Überlebensfähige Sicherheitskomponenten für Hochgeschwindigkeitsnetze

Entwurf und Realisierung am Beispiel einer Packet Screen

als Dissertation zur Erlangung des Doktorgrades
am Fachbereich Informatik der Universität Hamburg

vorgelegt von

Carsten Benecke

aus Hamburg

Hamburg 2002

Genehmigt vom Fachbereich Informatik der Universität Hamburg

auf Antrag von Prof. Dr. B.E. Wolfinger, Universität Hamburg
Prof. Dr. K. Brunnstein, Universität Hamburg
Dr. H.-J. Mück, Universität Hamburg

Hamburg, den 12. Februar 2002
(Tag der Disputation)

Prof. Dr. H.-S. Stiehl
Dekan

Vorwort des Herausgebers

Die zunehmende Bedeutung sicherer Informatiksysteme — und dabei insbesondere sicherer Kommunikations- und Rechnernetze (wie dem Internet) — wird inzwischen allseits anerkannt. Gleichwohl besitzen die gegenwärtig im Einsatz befindlichen Rechner und Netze nicht selten noch gravierende Sicherheitslücken.

Noch ungünstiger stellt sich die Situation dar, wenn nicht nur sichere sondern überlebensfähige Informatiksysteme gefordert werden, d.h. Systeme, die gleichzeitig ein hohes Maß an Sicherheit (z.B. Abhörsicherheit), Leistungsfähigkeit (z.B. garantierte Dienstgüte bei Rechnernetzdiensten) und Zuverlässigkeit (z.B. Verfügbarkeit) aufweisen sollen.

Die vorliegende Arbeit behandelt die Entwicklung überlebensfähiger Informatiksysteme am Beispiel von Sicherheitskomponenten in Netzen und legt dabei besonderes Gewicht auf eine ganzheitliche Sicht der Entwurfsproblematik für überlebensfähige Systeme. Im Gegensatz zu bisherigen Arbeiten, bei denen in der Regel nur Teilaspekte Berücksichtigung finden, schenkt der Autor den Aspekten Sicherheit, Leistungsfähigkeit und Zuverlässigkeit gleichermaßen Beachtung.

Dabei erzielt Herr Benecke wertvolle Beiträge in sehr unterschiedlichen Bereichen, u.a. durch

- Entwicklung von Richtlinien für einen, auf eine ganzheitliche Sicht ausgerichteten, Systementwurf mit Erarbeitung entwurfsunterstützender (semi-)formaler Beschreibungs- und Analysetechniken (insbesondere durch Erweiterung von “Message Sequence Charts — MSCs” sowie der Sprache “Specification and Description Language — SDL”);
- prototypische Umsetzung des vorgeschlagenen neuen Entwurfskonzeptes unter exemplarischer Anwendung der erarbeiteten Beschreibungs- und Analysehilfsmittel beim Entwurf einer parallelen “Packet Screen” (als einer zentralen Sicherheitskomponente in Rechnernetzen);
- analytische Modelle zur validen Leistungs- und Zuverlässigkeitsbewertung von parallelen “Packet Screens”;
- Entwurf und Implementierung eines leichtgewichtigen und sicheren Gruppenmanagement-Protokolls.

Ein besonderes Merkmal der Arbeit liegt dabei in ihrer starken Praxisorientierung und der konsequenten Nutzung der langjährigen Erfahrungen des Autors (z.B. basierend auf seinen vielbeachteten Projektarbeiten im “DFN-Firewall-Labor” des Fachbereichs Informatik der Universität Hamburg), um die Praxistauglichkeit von entwurfsunterstützenden Beschreibungs-, Modellierungs- und Analysetechniken anhand der konkreten Implementierung von Prototypen für Sicherheitskomponenten kritisch zu überprüfen und zu beurteilen.

Daher dürfte die vorliegende Arbeit nicht nur für Systementwickler sondern auch für Betreiber sicherer Kommunikationsnetze von beträchtlichem Interesse sein und (hoffentlich) wertvolle Anstöße für weitere FuE-Aktivitäten auf dem zur Zeit noch weiten Weg zu hochgradig überlebensfähigen Informatiksystemen liefern.

Hamburg, im Februar 2002 Bernd E. Wolfinger

Kurzfassung

Überlebensfähige Systeme zeichnen sich durch eine besonders gute Resistenz gegen verschiedenartige Störungen aus. Zu diesen Störungen gehören beispielsweise Ausfälle einzelner Komponenten aufgrund von Software- oder Hardwarefehlern oder auch Beeinträchtigungen, die durch Angriffe von Computer-Kriminellen verursacht werden.

Um ein überlebensfähiges System zu entwerfen und zu realisieren, sind daher eine Vielzahl verschiedenartiger Analyse- und Syntheseschritte erforderlich. Das Ziel der (wiederholten) Systemanalyse ist es, die unterschiedlichen Bedrohungen, Fehlermöglichkeiten und Leistungsengpässe zu identifizieren. Die Bewertung vorhandener Systeme erfordert daher in der Regel die Anwendung von Methoden aus verschiedenen Disziplinen, insbesondere aus den Bereichen des Fehler-, Leistungs-, und Sicherheitsmanagements. Bei der darauf folgenden Synthese müssen die gefundenen Defizite ausgeräumt werden. Das Hauptproblem bei der Synthese von überlebensfähigen Systemen liegt in der Notwendigkeit zur gleichzeitigen Optimierung teilweise gegenläufiger Ziele, die dazu führen können, daß beispielsweise die Erhöhung der Leistungsfähigkeit zu neuen, bisher nicht vorhandenen Sicherheitsrisiken führt.

Zur Zeit gibt es leider kaum praktische Erfahrungen und konkretes Methodenwissen im Entwurf von überlebensfähigen Systemen. In der vorliegenden Arbeit soll daher am Beispiel der Transformation einer wichtigen bekannten Firewall-Komponente untersucht werden, ob durch die sukzessive Verbesserung eines Systems mit Hilfe von Standardmethoden der Systemanalyse und -synthese schließlich ein überlebensfähiges Pendant entwickelt werden kann. In Fällen, bei denen vorhandene Analysemethoden/-werkzeuge nicht ausreichen, um die erforderlichen Untersuchungen durchzuführen, werden entsprechende Erweiterungen motiviert und angewendet. In der Arbeit werden insbesondere Erweiterungen an Meßwerkzeugen und Meßmethoden vorgestellt sowie verschiedene formale Spezifikations- und Analysemethoden miteinander kombiniert.

Neben der Diskussion der Defizite des vorhandenen Systems und der notwendigen Verbesserungen wird auch der “Tradeoff” zwischen verschiedenen Entwurfsalternativen diskutiert bzw. mit Hilfe von analytischen Modellberechnungen auf Basis von empirischen Meßergebnissen verdeutlicht. Die abschließende Analyse des Prototypen einer “überlebensfähigen Packet Screen” zeigt, daß Leistungsfähigkeit, Verfügbarkeit und Sicherheit durchaus gleichzeitig erreichbar sind und die gewählte Methode der “iterativen Adaption” erfolgreich für die Transformation eines Systems in ein überlebensfähiges Pendant eingesetzt werden kann.

Abschließend wird anhand eines Architekturbeispiels für ein überlebensfähiges Perimeternetz der Transfer von Ergebnissen auf andere wichtige Netzkomponenten vorgestellt. Es wird somit gezeigt, daß die am Beispiel gewonnenen Ergebnisse vielfach anwendbar sind, insbesondere auf andere Sicherheitskomponenten in Hochgeschwindigkeitsnetzen.

Abstract

Survivable systems are known to be resistant to different kinds of problems. Among these are failures due to software or hardware faults, but also attacks caused by computer criminals.

The design and implementation of survivable systems therefore requires a variety of different steps to support system analysis and synthesis. It is the goal of a repeatedly applied analysis of the system to identify all kinds of threats, errors, and performance bottlenecks. Thus, the evaluation of a system usually requires the combination of failure-, performance- and security management. During the synthesis of a survivable system in its design phase all recognized deficiencies have to be removed. The main problem of building survivable systems is the optimization of more than one competitive goals. For example, increasing the performance of the system may result into additional security risks.

At present, we are not only lacking experience in the design of survivable systems, but we are also in need of general methods which will ultimately transform a system into a survivable counterpart. This thesis demonstrates the migration of an important firewall building block, i.e. a packet screen, to a survivable system as a case study. The chosen method will adapt the packet screen step by step to the analyzed requirements. Doing this, standard methods will be applied whenever possible. These methods will be combined and extended whenever needed. It will be shown that also the extension of tools for the performance analysis is necessary. Moreover, it will be stated that a general security analysis requires the combination of several formal methods to increase the insight into the analyzed systems.

The tradeoff between design alternatives will be discussed as well and exemplified by means of analytical models and empirical load measurements. A final analysis will demonstrate that the chosen method results in a survivable packet screen: Performance, reliability and security can indeed be combined in one system.

Finally, an example of a survivable perimeter network will show that the results can be transfered to many other security building blocks for high speed networks. Thus, we conclude that our new method of “step by step adaption” as well as the chosen formal techniques are applicable to many different subsystems of communication networks.

Danksagung

An dieser Stelle möchte ich mich für die vielfältige Unterstützung bedanken, die mir die Anfertigung dieser Dissertationschrift erst ermöglichte.

Mein ganz besonderer, herzlicher Dank gilt Herrn Dr. Hans-Joachim Mück, der mir die langjährige Forschung in den zwei Drittmittelprojekten "Firewall-Labor für Hochgeschwindigkeitsnetze" (DFN-FWL) und "Sicherheit in ATM-Netzen" am Rechenzentrum des Fachbereichs Informatik der Universität Hamburg ermöglichte.

Für die überaus gute, fruchtbare Zusammenarbeit möchte ich den folgenden ehemaligen Kollegen aus dem Projekt DFN-FWL und den Partnerprojekten (DFN-CERT und DFN-PCA) danken, die mir immer mit Rat und Tat zu Seite standen: Dr. Uwe Ellermann (DFN-FWL), Stefan Kelm (DFN-PCA), Britta Liedtke (DFN-PCA), Wolfgang Ley (DFN-CERT), Les Schäfer (DFN-CERT) und Dr. Klaus-Peter Kossakowski (DFN-CERT), sowie Dr. Marcus Pattloch (DFN-Verein).

Für die fleißige Arbeit im Firewall-Labor danke ich Olaf Gellert, Stephan Holst, Axel Großklaus, Gregor Goldbach und Michael Krooß. Für die gute Kooperation danke ich meinen Kollegen vom Rechenzentrum, insbesondere dem Operating-Team, Reinhard Zierke und Andreas Lukas.

Für die vielen Hinweise und Anregungen zur schriftlichen Fassung dieser Arbeit danke ich Dr. Uwe Ellermann, Dr. Klaus-Peter Kossakowski, Wolfgang Ley, Stephan Kelm, Dr. Hans-Joachim Mück, Heino Peters und Dr. Michaela Welk.

Für ihre Betreuung und die Bereitschaft ein Gutachten zu dieser Arbeit anzufertigen, danke ich Prof. Dr. Bernd E. Wolfinger und Prof. Dr. Klaus Brunnstein sowie Dr. Hans-Joachim Mück. Besonderen Dank schulde ich Prof. Dr. Bernd E. Wolfinger für seine jahrelange Betreuung meiner akademischen Arbeiten (Studien-, Diplomarbeit und Dissertationsschrift) und die hiermit verbundenen, vielen hilfreichen Hinweise und Anregungen, insbesondere im Bereich der Modellbildung.

Abschließend danke ich meiner Familie für ihre Geduld mit mir und meiner Arbeit.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Ziele der Arbeit 2

1.2 Weitere Vorgehensweise 4

2 Überlebensfähige Systeme: Notwendigkeit, Entwurfsempfehlungen und Mechanismen **7**

2.1 Risiken durch die gängige Entwurfspraxis 8

2.1.1 Fallbeispiel 1: Priorisierung des Leistungsmanagements 8

2.1.2 Fallbeispiel 2: Ignorieren des Sicherheitsmanagements 9

2.2 Überlebensfähige Systeme 10

2.2.1 Definition: Überlebensfähigkeit 11

2.2.2 Anforderungen an überlebensfähige Systeme 12

2.2.3 Primäre Eigenschaften von überlebensfähigen Systemen 15

2.2.4 Entwurfsempfehlungen für überlebensfähige Systeme 16

2.3 Überlebensfähigkeit am Beispiel der Packet Screen 18

2.3.1 Beispiel: Überlebensfähige Firewall-Komponenten 19

2.3.1.1 Firewalls für Zugriffskontrolle und Audit 20

2.3.1.2 Bedeutung der Firewalls für die Netzsicherheit 22

2.3.1.3 Bedeutung der Packet Screen für Firewall-Architekturen . 23

2.3.2 Notwendige Reduzierung der Komplexität 27

2.4 Zusammenfassung 28

3 Engpässe durch Packet Screens in Hochgeschwindigkeitsnetzen **31**

3.1 Leistungsbewertung von Rechen- und Kommunikationssystemen 32

3.2 Szenarien, Last-, Konfigurations- und Leistungskenngrößen für die Bewertung von Packet Screens . . . 33
3.2.1 Szenarien für die Leistungsbewertung von Packet Screens . . . 35
3.2.1.1 Allgemeiner Versuchsaufbau . . . 35
3.2.1.2 Versuchsaufbau für Fast Ethernet-Messungen . . . 38
3.2.2 Last-, Konfigurations- und Leistungskenngrößen für die Bewertung von Packet Screens . . . 39
3.2.2.1 Lastkenngrößen . . . 42
3.2.2.2 Konfigurationskenngrößen . . . 44
3.2.2.3 Leistungskenngrößen . . . 44
3.3 Anforderungen an Meßwerkzeuge . . . 45
3.3.1 Statistische Absicherung . . . 46
3.3.2 Trennung von Konfigurations- und Meß-Dienstzugangspunkt . . . 47
3.3.3 Synchronisation mehrerer Meßinstanzen . . . 47
3.4 Wahl geeigneter Parameter und Konfigurierung der Meßsysteme . . . 48
3.4.1 Verwendete Hardware für die Meßdurchführung . . . 48
3.4.2 Größe der "Service Data Unit" auf der Transportebene . . . 49
3.4.3 Puffergröße des "Service Access Points" . . . 50
3.4.4 Transportprotokoll . . . 50
3.4.5 Meßdauer, Wiederholungen und Vertrauensintervalle . . . 51
3.4.6 Anzahl der Filterregeln . . . 52
3.5 Theoretischer Durchsatz und Referenzmessungen . . . 53
3.6 Packet Screen als Engpaß in Hochgeschwindigkeitsnetzen . . . 55
3.6.1 Verarbeitung von Filterregeln bei Packet Screens . . . 55
3.6.2 Leistungsbewertung einer Fast Ethernet-basierten Packet Screen . . . 59
3.7 Zusammenfassung . . . 60

4 Parallele Packet Screen für Hochgeschwindigkeitsnetze 63

4.1 Leistungsanforderungen an eine Packet Screen für Hochgeschwindigkeitsnetze 64
4.2 Leistungssteigerung durch parallele Protokollverarbeitung . . . 66
4.2.1 Gängige Klassifikationen im Bereich der Parallelverarbeitung . . . 66

4.2.1.1 Klassifikation bezüglich Rechnerarchitektur 67
4.2.1.2 Klassifikation bezüglich Zuordnung 67
4.2.2 Klassifikation bezüglich Protokollverarbeitung 69
4.3 Entwurf einer parallelen Packet Screen 74
4.3.1 Statische und dynamische Parallelität 76
4.3.2 Parallele Packet Screen auf Basis der Paketparallelität 78
4.3.3 Lastverteilung bei der Paketparallelität 79
4.3.4 Dezentrale Zuordnung der Pakete zu Prozessoren 80
4.3.5 Dezentrale Selektion von Paketen 81
4.3.6 Selektion innerhalb des IP-Filter Moduls 85
4.3.7 Selektion innerhalb des IP-Kernel Moduls 86
4.3.8 Selektion innerhalb der Geräte-Treiber 86
4.4 Realisierung von Hochleistungs-Packet Screens 87
4.4.1 Parallele Packet Screen für “shared” Ethernet 87
4.4.2 Verbesserte parallele Packet Screen für “switched” Ethernet 89
4.5 Leistungsbewertung der parallelen Packet Screen 90
4.5.1 Parallele Packet Screen in “shared” Fast Ethernet Netzen 92
4.5.1.1 Messung ohne konfigurierte Filterregeln 92
4.5.1.2 Messung mit 50 konfigurierten Filterregeln 94
4.5.1.3 Bidirektionale Messung mit zwei Lastgeneratoren 94
4.5.2 Parallele Packet Screen in “switched” Fast Ethernet Netzen 98
4.5.3 Zusammenfassung: Meßergebnisse Parallele Packet Screen 99
4.6 Analytische Bewertung der Parallelen Packet Screen 100
4.6.1 Modell einer parallelen Packet Screen 100
4.6.2 Analytische Berechnung des Paketdurchsatzes 103
4.6.3 Bestimmung der Werte für eine realistische Modellparametrisierung 105
4.6.4 Modellbasierte Analyse des Gewinns durch Parallelverarbeitung . . 108
4.6.5 Auswirkungen des Einsatzes schnellerer Hardware 109
4.6.6 Auswirkungen des Einsatzes aufwendiger Filteroperationen 110
4.6.7 Auswirkungen der Verbindungsparallelität 112
4.7 Zusammenfassung 113

5 Variationen des Selektionsalgorithmus als Basis für eine flexible Parallelverarbeitung 115

5.1 Unterschiedliche Leistungsfähigkeit der Prozessoren 115

5.2 Dynamisches Filtern und Verbindungsparallelität 117

5.2.1 Dynamisches Filtern für "Remote Procedure Call"-Zugriffskontrolle 118

5.2.2 Dynamisches Filtern für "Intrusion Detection System"-Rückkoppelung 119

5.2.3 Anforderung: Unterstützung dynamischer Filter 121

5.2.4 Algorithmus zur Unterstützung dynamischer Filter 121

5.2.5 Weitere Anwendungsmöglichkeiten der Verbindungsparallelität . . . 124

5.2.6 Mehrstufige Selektion 124

5.3 Zusammenfassung 125

6 Ausfallsicherheit der parallelen Packet Screen 127

6.1 Zuverlässigkeit der Packet Screen und Auswirkungen durch den Ausfall von Prozessoren 128

6.1.1 Zuverlässigkeit und Kenngrößen 128

6.1.2 Modelle zur Fehleranalyse 129

6.1.3 Ausfall einer Instanz der parallelen Packet Screen 131

6.2 Fehlermodell und Fehlersemantik 134

6.2.1 Fehlermodell für die parallele Packet Screen 134

6.2.1.1 Potentielle Hardware- und Softwarefehler 135

6.2.1.2 Fehlersemantik der parallelen Packet Screen 137

6.2.2 Bewertung der Fehlersemantik 137

6.3 Anforderungen an eine fehlertolerante Packet Screen 138

6.3.1 Anforderungen an die Fehlerbehandlung 139

6.3.2 Weitere Anforderungen 139

6.4 Abhängigkeiten der Mechanismen im Bereich der fehlertoleranten Systeme 140

6.5 Verwaltung von Gruppenzugehörigkeit 143

6.6 Das zeitgesteuerte, asynchrone Systemmodell 144

6.7 Zusammenfassung 145

7 Leichtgewichtiges, sicheres Gruppenmanagement-Protokoll 147

7.1 Funktionsbedingter Aufwand beim Gruppenmanagement 148

7.2 Entwurf eines leichtgewichtigen Gruppenmanagement-Protokolls 152

7.2.1 Dienstspezifikation Gruppenmanagement-Dienst 153

7.2.2 Technische Annahmen über die Umgebung 157

7.2.3 Automatische Erkennung eines Ausfalls 157

7.2.3.1 “Heartbeat”-Signale als “Multicasts” 159

7.2.3.2 Heuristik für die Ausfallserkennung 159

7.2.4 Semiformale Beschreibung der Funktionsweise des Protokolls 161

7.2.4.1 Verhalten bei erkennbaren Fehlern 162

7.2.4.2 Erhöhen der Robustheit I 163

7.2.4.3 Aufnahme in die Gruppe 163

7.2.4.4 Erhöhen der Robustheit II 164

7.2.5 Protokollspezifikation 166

7.2.5.1 Protokollautomat 166

7.2.5.2 Zeitlicher Ablauf 170

7.2.5.3 “Protocol Data Unit”-Format 174

7.2.5.4 SDL-Spezifikation 175

7.3 Validation vs. Sicherheitsanalyse 181

7.3.1 Validation des leichtgewichtigen Gruppenmanagement-Protokolls . . 181

7.3.2 Vollständigkeit 181

7.3.2.1 Syntaktische Vollständigkeit 182

7.3.2.2 Semantische Vollständigkeit 182

7.3.3 Keine Verklemmungen (“Deadlocks”) 182

7.3.4 Keine “Livelocks” 183

7.3.5 Kein ungewollter Abbruch 183

7.4 Erreichte Zuverlässigkeit 183

7.4.1 Verfügbarkeit der parallelen Packet Screen 184

7.4.2 Verfügbarkeit bei Leistungsvorgaben 184

7.5 Zusammenfassung 186

8 Sicherheitsanalyse für das leichtgewichtige GM-Protokoll 189

8.1 Sicherheitsrelevante Annahmen über die Umgebung 190

8.2 Sicherheitsanforderungen an das Gruppenmanagement-Protokoll 192

8.3 Zusätzliche Entwurfsschritte für Protokolle in überlebensfähigen Systemen 193

8.4 Sicherheitsanalyse für Protokolle 194

8.4.1 BAN-Logik und deren Weiterentwicklungen 195

8.4.1.1 Schlußregeln der BAN-Logik 196

8.4.1.2 Wichtige Ergänzungen der BAN-Logik 198

8.4.2 Automatenmodell-basierte Ansätze 199

8.4.3 Hybride Methoden 200

8.4.4 Offene Probleme und Schlußfolgerungen 200

8.5 Sicherheitsprobleme, Bedrohungspotential und erforderliche Sicherheitsmechanismen 203

8.5.1 Zu SA1: Nur autorisierte Prozessoren dürfen in die Gruppe aufgenommen werden 204

8.5.2 Zu SA2: Autorisierten Prozessoren darf die Aufnahme in die Gruppe nicht verweigert werden 207

8.5.3 Zu SA3: Die Integrität des Gruppenkonsens muß gewährleistet sein 212

8.5.4 “Denial of Service”-Angriff durch IP-Dateneinheiten 213

8.5.4.1 “Denial of Service”-Angriff: Erfolgsaussichten und Auswirkungen 214

8.5.4.2 Worst-Case Szenario 216

8.5.5 “Denial of Service”-Angriff auf das Gruppenmanagement-Protokoll . 217

8.5.5.1 Erkennen des “Denial of Service”-Angriffs 219

8.5.5.2 Abwehren des Angriffs 220

8.6 Zusammenfassung der zusätzlichen Mechanismen für ein sicheres GM-Protokoll 223

8.6.1 “Keyed Message Authentication Code” für Integrität und Authentizität 223

8.6.2 Streng monoton steigende Sequenznummer 223

8.6.3 Authentifikationsanfrage an Bewerber 224

8.6.4 Alumni-Liste und stabiler Speicher 224

8.6.5 Zusätzlicher Zustand für DoS-Angriffe 224

8.7 Zusammenfassung 225

9 Implementationsaspekte und Test des Gruppenmanagement-Protokolls 227

9.1 Eingesetze Hardware und Software 228

9.2 Maßnahmen für geringen Speicherverbrauch 229

9.2.1 Erkennen und Vermeiden von Speicherlecks 229

9.2.2 Zusätzliche Maßnahmen 230

9.3 Identifikation der Prozessoren 231

9.4 Analyse eintreffender Dateneinheiten 231

9.5 Implementation der Zeitgeber 232

9.6 Test des Timings 233

9.6.1 Verwendung von “Tracing”-Techniken 234

9.6.2 Unterstützung der Fehlersuche 234

9.6.3 Konsequente Modularisierung 235

9.7 Host-Sicherheit 235

9.8 Leistungsbewertung einer abgesicherten parallelen Packet Screen 236

9.9 Zusammenfassung 237

10 Ein überlebensfähiges Perimeternetz für Hochgeschwindigkeitsnetze 239

10.1 Eine Beispielarchitektur 240

10.2 Überlebensfähige Firewall-Komponenten 240

10.2.1 Bastionen und Proxies 240

10.2.2 “Virtual Private Network”-Gateways 242

10.2.3 “Sniffer” für “Intrusion Detection”-Systeme 242

10.2.4 Audit-Trails 243

10.3 Überlebensfähige Informationsdienste 245

10.3.1 DNS-Service 245

10.3.2 HTTP-Server 246

10.4 Zusammenfassung 247

11 Resümee und Ausblick 249

11.1 Zusammenfassung der Ergebnisse 249

11.2 Offene Fragestellungen 252

A Spezifikation mit "Message Sequence Charts" und "Specification and Description Language" 255

A.1 Einführung in die graphische Notation der "Message Sequence Charts" und der "Specification and Description Language" 255

A.1.1 Einführung in "Message Sequence Charts" 256

A.1.2 Einführung in die "Specification and Description Language" 257

A.2 "Message Sequence Charts" als standardisierte, semiformale Methode zur Sicherheitsanalyse von Protokollen 261

A.2.1 "Message Sequence Charts" und BAN-Logik 263

A.2.2 "Message Sequence Charts" und "Strands" 264

A.2.3 "Message Sequence Charts" und Verarbeitungskosten 266

A.2.4 Gleichzeitige Integration mehrerer Annotationen 272

A.3 Zusammenfassung 274

B Protokollspezifikation mit der "Specification and Description Language" 275

B.1 Hilfsspezifikation: "Broadcast"-Medium 275

B.2 Spezifikation eines Gruppenmanagement-Protokolls 277

B.2.1 Systemstruktur 277

B.2.2 Blockstruktur 277

B.2.3 Prozeßstruktur (Protokollinstanz) 279

B.2.4 Prozeßstruktur (Admin-Prozeß) 293

B.2.5 Diverse Hilfsfunktionen 293

B.3 Zusammenfassung 303

C Durchführung von Messungen und Erweiterungen an Meßwerkzeugen und Gerätetreibern 305

C.1 Fallstricke bei der Meßdatenerfassung 305

C.1.1 Fehlerquelle: "Plug and Play"-Netzanschluß 306

C.1.2 Fehlerquelle: falsche und ungleichmäßige Konfigurierung von Monitor, Lastgenerator und der Systeme unter Last 306

C.1.3 Fehlerquelle: Störlast im Testnetz 307

C.1.4 Fehlerquelle: Anzahl der Prozessoren auf den Meßsystemen 307

C.2 Erweiterungen an den Meßwerkzeugen 309

C.2.1 Statistische Absicherung 309
C.2.2 Trennung der Wegwahl für gleichzeitige Kommunikationsbeziehungen 309
C.2.3 Synchronisation mehrerer Meßsysteme 310
C.2.4 Leistungsvorgaben 311
C.3 Erweiterungen an den Gerätetreibern 312
C.3.1 Erweiterungen für die verteilte Selektion 313
C.3.2 Erweiterungen an der Flußkontrolle 316
C.3.3 Erweiterungen an der Fehlererkennung 319
C.4 Zusammenfassung 322

Abkürzungsverzeichnis **323**

Literaturverzeichnis **327**

Kapitel 1

Einleitung

In den vergangenen 20 Jahren hat sich ein deutlicher Wandel innerhalb der Gesellschaft vollzogen. Es gibt eine zunehmende Abhängigkeit von der Informationstechnologie (IT). Die IT-basierte Informationsverarbeitung durchdringt immer mehr gesellschaftliche Bereiche und schafft neue Abhängigkeiten. Keine Industrienation ist heute mehr in der Lage, auf den Einsatz von Informationstechnologien zu verzichten; in einigen Geschäftszweigen, insbesondere im Finanzsektor und in der öffentlichen Verwaltung, gibt es bereits eine vollständige Abhängigkeit von der Computer-gestützten Informationsverarbeitung. Dieser Trend wird sich auch weiterhin fortsetzen. Immer komplexere Arbeitsabläufe werden IT-gestützt abgewickelt und erfordern im gleichen Zuge immer leistungsfähigere Rechner und Netzwerke. Wo früher noch kleine Übertragungsgeschwindigkeiten für Text-basierte elektronische Post ausreichte, wird heute ein "breitbandiges" Hochgeschwindigkeitsnetz mit Übertragungsgeschwindigkeiten jenseits von 100 Mbit/s benötigt, um ganze Arbeitsabläufe im Bereich der Bürokommunikation mit Unterstützung von verteilten Systemen effizienter zu gestalten, oder um einer ständig steigenden Anzahl von Benutzern den Zugriff auf multimediale Daten zu ermöglichen (Internet-Radio, "Video on Demand", elektronische Marktplätze mit interaktiven Produktpräsentationen usw.).

Diese Abhängigkeit führt bei einem Versagen der IT zu ganz erheblichen Problemen. Die möglichen Folgen fangen an bei Ärgernis erregenden Alltäglichkeiten, wie beispielsweise Datenverlust durch einen Programmabsturz. Erheblich drastischer sind finanzielle Verluste durch Ausfall von Rechnern, über die Banktransaktionen abgewickelt werden. Und sogar Leib und Leben sind in Gefahr, wenn es zu Fehlfunktionen beispielsweise bei Computer-basierten medizinischen Geräten kommt.

Ein weiteres zu beobachtendes Phänomen, das mit der steigenden Verbreitung der IT in vielen Bereichen der Gesellschaft einhergeht, ist die Zunahme der Computer-Kriminalität. Defacto hat die Computer-Kriminalität in den letzten Jahren ständig zugenommen, wie die gesammelten Vorfallsdaten von Computer-Notfallteams und vergleichbaren Institutionen belegen [138]. Ein aktuelles Beispiel sind die jüngsten Erpressungsversuche amerikanischer "E-Commerce-Betreiber". Ukrainische und russische Computer-Kriminelle brachen in die

Systeme ein und stahlen vertrauliche Daten, u.a. Kreditkarteninformationen. Die betroffenen Unternehmen sollten gezwungen werden, bei den "Hackern" einen kostenpflichtigen "Service" in Anspruch zu nehmen, damit sich derartige Vorfälle nicht wiederholen würden (vgl. [38]). Ein weiteres Beispiel ist der Diebstahl von Daten: "Softwarepiraten" gelang ein spektakulärer Einbruch in einen Microsoft Rechner und der anschließende Diebstahl von Quellcodes (vgl. [221]). Die Beispiele verdeutlichen, daß es für die von der IT abhängigen Institutionen unerheblich ist, ob das Versagen ihrer Systeme durch natürliche Katastrophen, unsachgemäße Bedienung, technische Fehlfunktion oder Sabotage herbeigeführt wird. Die Auswirkungen können in allen Fällen verheerend sein.

Es ist somit in vielen Bereichen eine "Überlebensfähigkeit" der IT bzw. der IT-gestützten Arbeitsabläufe trotz widriger Umstände erforderlich. Die Überlebensfähigkeit (für die Definition vgl. Abschnitt 2.2) stellt hohe technische Anforderungen an die Informationstechnologie: Das Versagen der IT muß im Idealfall ausgeschlossen werden. Ist dies nicht möglich, müssen die negativen Folgen begrenzt werden, und es muß eine möglichst schnelle Wiederherstellung eines ordnungsgemäßen Betriebsablaufs erfolgen. In vielen Fällen erfordert eine solche Überlebensfähigkeit aus technischer Sicht[1] eine Kombination von Maßnahmen aus den Teilbereichen *Fehlermanagement, Sicherheitsmanagement und Leistungsmanagement.*

Auf der einen Seite gibt es einen Bedarf nach gleichzeitig sicheren, hoch verfügbaren und leistungsfähigen Systemen. Andererseits gibt es leider für die Analyse und den Entwurf solcher Systeme noch keine breit akzeptierten Methoden. Erste Fallbeispiele für überlebensfähige Systeme zeigen darüber hinaus, daß diese Systeme hoch spezialisiert sind (vgl. Kapitel 2). Es lassen sich daher kaum Rückschlüsse auf die allgemeine Anwendbarkeit der gewählten Methoden ziehen. Die entwickelten Systeme sind darüber hinaus nur für vereinzelte Anwendungsfälle geeignet, so daß dem hohen Entwicklungsaufwand ein vergleichsweise geringer Nutzen gegenübersteht.

1.1 Ziele der Arbeit

In dieser Arbeit soll daher ein etwas anderer, als der in Kapitel 2 skizzierte Ansatz, gewählt werden. Es soll versucht werden, mit etablierten Standardmethoden aus den Bereichen des Leistungs-, Fehler- und Sicherheitsmanagements ein vielseitig verwendbares System, das gleichzeitig prototypisch für eine ganze Klasse vergleichbarer Komponenten steht, in

[1] Neben den technischen Maßnahmen sind auch personelle und organisatorische Maßnahmen erforderlich. Beispiele für organisatorische Maßnahmen sind Risiko-Management, Notfallplanung oder auch Abschluß von Versicherungen. Beispiele für personelle Maßnahmen sind Schulung von Mitarbeitern oder Ernennung eines Sicherheitsbeauftragten.

ein überlebensfähiges System zu überführen. In Kapitel 2 wird die Wahl einer "Firewall-Komponente[2]", der sogenannten "Packet Screen[3]", zu diesem Zweck begründet.

Gelingt diese Transformation, ist zumindest sichergestellt, daß durch die Wahl des Systems ein breites Anwendungsfeld eröffnet wird und die Ergebnisse möglicherweise leicht an andere Systeme der Klasse adaptiert werden können. Dadurch könnten wesentliche Netz-Komponenten für viele verteilte Systeme überlebensfähig ausgelegt werden.

Der Einsatz von Standardmethoden bei der Analyse und Synthese ist ebenfalls von Vorteil. Unabhängig davon, ob die Überführung des Systems in ein überlebensfähiges Pendant gelingt, können Probleme und Vorteile des Einsatzes dieser Standardmethoden aufgezeigt werden. Durch die Verwendung von Standardtechniken bzw. verbreiteten Analyse- und Synthesemethoden würden von positiven Ergebnissen eine Vielzahl von Entwicklern profitieren. Mißlingt der Einsatz dieser Techniken, kann möglicherweise aufgezeigt werden, worin die dafür ursächlichen Defizite liegen.

Das Hauptziel ist somit die exemplarische Migration einer weit verbreiteten Firewall-Komponente zu einem überlebensfähigen System, mit begleitender Evaluierung durch heute gängige, empfohlene Methoden.

Dabei werden verschiedene Unterziele verfolgt:

- Es muß der Begriff der "Überlebensfähigkeit" definiert und der Stand der Forschung in diesem Bereich aufgezeigt werden. Dies beinhaltet auch eine Diskussion der Frage, warum es heute trotz des hohen Bedarfs noch so wenig überlebensfähige Systeme gibt.

- Es muß die Wahl der Packet Screen als Beispiel für die prototypische Überführung in ein überlebensfähiges System begründet werden. Es muß festgelegt werden, was das vorrangige Ziel einer Überführung in ein überlebensfähiges Pendant ist, falls eine gleichzeitige Optimierung aller Anforderungen nicht möglich sein sollte. Es muß eine nachvollziehbare, möglichst einfache Methodik für die weiteren Analyse- und Syntheseschritte gefunden werden.

- Bei der anschließenden Überführung sollen Standardmethoden verwendet werden. Dabei sind Defizite an vorhandenen Konzepten, Methoden und Werkzeugen aufzuzeigen und falls möglich zu beseitigen bzw. zu verringern. Priorität muß dabei die Kombination bekannter Methoden vor der Entwicklung neuer Ansätze haben, da so eher eine breite Akzeptanz der Erkenntnisse erwartet werden kann.

- Durchzuführende Erweiterungen an der ausgewählten Beispielkomponente müssen so allgemein wie möglich gehalten werden, um eine Adaption an andere, vergleichbare

[2] Als Vorabdefinition für Firewall reicht an dieser Stelle: Ein Firewall ist ein Konzept für die Zugriffskontrolle in Netzen.

[3] Vorabdefinition: Eine "Packet Screen" führt eine grobe Zugriffskontrolle, basierend auf Informationen in Dateneinheiten, innerhalb eines Firewalls durch. Aufgrund der häufigen Nennung von "Firewall" und "Packet Screen", werden diese Begriffe im folgenden ohne Anführungszeichen verwendet.

Komponenten zu erleichtern. Dabei sollten möglichst nachhaltige Verbesserungen vorgenommen werden, und es muß diskutiert werden, wie die Optimierungsziele die Auswahl unter alternativen Erweiterungsmöglichkeiten beeinflussen.

- Die Verbesserungen/Erweiterungen müssen ihrerseits mit den vorhandenen Analysemethoden bewertet werden. Dabei muß der "Tradeoff" zwischen verschiedenen Erweiterungen dargestellt werden, da zu erwarten ist, daß die gleichzeitige Optimierung so verschiedener Ziele wie Leistungsfähigkeit, Verfügbarkeit und Sicherheit nicht immer gleichermaßen möglich ist.

- Es muß diskutiert werden, ob und wie sich Ergebnisse auf andere Systeme übertragen lassen. Es ist zu untersuchen, in wieweit die Lösungsansätze und Methoden auf andere Systeme übertragen werden können, um möglicherweise Methodenwissen für die Analyse und den Entwurf von überlebensfähigen Systemen ableiten zu können.

1.2 Weitere Vorgehensweise

Kapitel 2 verdeutlicht an zwei Beispielen, daß heutige Systeme in der Regel nicht überlebensfähig sind und sich dieser Mißstand meist unmittelbar durch die heute gängige Entwurfspraxis ergibt. Diese Beispiele sind prototypisch für viele Systeme, die nicht in Hinblick auf Überlebensfähigkeit entworfen wurden. Es wird der Begriff der "Überlebensfähigkeit" definiert und bereits erarbeitete Erkenntnisse aus diesem Bereich dargestellt. Anschließend wird die Wahl einer Packet Screen für die Überführung in ein überlebensfähiges System dargestellt, und es wird diskutiert, welche Annahmen für die weitere Vorgehensweise gemacht werden müssen, um möglichst allgemeine Erkenntnisse gewinnen zu können.

In Kapitel 3 wird die Leistungsfähigkeit einer vorhandenen Workstation-basierten Packet Screen untersucht. Es werden Kriterien für die Leistungsbewertung für die Packet Screen und einer ganzen Klasse weiterer Netzkomponenten hergeleitet. Es werden Defizite und notwendige Erweiterungen an vorhandenen Analysewerkzeugen für die systematische Bewertung dieser Komponenten dargestellt. Die Darstellung der typischen Szenarien für den Einsatz von Packet Screens und ähnlichen Netzkomponenten führt zu Szenarien für die systematische Leistungsbewertung dieser Komponenten. Anschließende Leistungsbewertungen von Packet Screens zeigen die Engpässe bei deren Einsatz in Hochgeschwindigkeitsnetzen auf.

In Kapitel 4 wird gezeigt, wie diese Engpässe beseitigt werden können. Es wird Parallelverarbeitung eingesetzt, um eine Beschleunigung der Packet Screen herbeizuführen. Dabei wird ein bekanntes Konzept gewählt, das gut skalierbar ist und auch an noch höhere Leistungsanforderungen angepaßt werden kann. Die Leistungssteigerung wird erneut mit den Methoden aus Kapitel 3 bewertet, um die erreichte Leistungssteigerung zu dokumentieren. Abschließend wird gezeigt, wie sich der Einsatz anderer Hardware und anderer Zugriffskontrollen auf die Leistungsfähigkeit auswirken würden.

In Kapitel 5 werden die Auswirkungen von alternativen Strategien bei der Parallelverarbeitung untersucht. Es wird gezeigt, daß der in Kapitel 4 gewählte Ansatz an heterogene Hardware und häufig gestellte, erweiterte Anforderungen an Packet Screens angepaßt werden kann.

In Kapitel 6 wird mit weiteren Standardmethoden die verbesserte Packet Screen erneut analysiert. Es wird gezeigt, daß durch die Leistungssteigerung die Zuverlässigkeit der Packet Screen verringert wird. Es wird diskutiert, welche Standardmethoden zur Erhöhung der Verfügbarkeit im Bereich des Fehlermanagements eingesetzt werden und wie diese Erkenntnisse auf die parallelisierte Packet Screen übertragen werden können. Es wird schließlich die Notwendigkeit für ein Gruppenmanagement-Protokoll (GMP) hergeleitet, um die verringerte Verfügbarkeit wieder auszugleichen.

In Kapitel 7 wird dargestellt, daß die vorhandenen Lösungen für ein Gruppenmanagement nicht anwendbar sind, da sie im Widerspruch zu den Leistungsanforderungen stehen oder die Sicherheit der parallelen Packet Screen reduzieren würden. Es folgt der Entwurf eines leichtgewichtigen GMPs, das nicht im Widerspruch zu den hohen Leistungsanforderungen an die Packet Screen steht. Wiederum werden für diesen Entwurf Standardmethoden eingesetzt. Es wird eine Grobspezifikation mit Hilfe eines endlichen Automatenmodells vorgenommen und schließlich eine vollständige, genaue Protokoll- und Dienstspezifikation mit Hilfe der “Specification and Description Language” (SDL) durchgeführt.

Das spezifizierte GMP wird in Abschnitt 8 einer eingehenden Sicherheitsanalyse unterzogen. Wiederum werden Standardmethoden eingesetzt, deren Defizite und notwendigen Erweiterungen diskutiert. Es werden Sicherheitsrisiken aufgezeigt, die erst durch den Einsatz des GMPs entstehen. Dies führt zu einer Integration zusätzlicher Mechanismen in das GMP, um es gegen die erkannten Angriffe weitgehend abzusichern. Dabei wird die Wahl der Mechanismen in Hinblick auf die Leistungsfähigkeit und Verfügbarkeit des Gesamtsystems (Packet Screen) vorgenommen.

Auf Grundlage der Spezifikation und der erforderlichen, zusätzlich notwendigen Mechanismen, erfolgt in Kapitel 9 die Darstellung der Implementierung eines sicheren, leichtgewichtigen GMPs. Erneut wird der Einsatz von Standardmethoden, in diesem Fall für die Implementierung und den anschließenden Test des Prototypen, diskutiert. Es wird mit einer abschließenden Leistungsbewertung gezeigt, daß die zusätzlichen Maßnahmen zur Erhöhung der Verfügbarkeit und der Sicherheit die Leistungsfähigkeit kaum beeinträchtigen.

In Kapitel 10 wird schließlich gezeigt, wie sich die Erkenntnisse auf andere Netzkomponenten auf Basis der in Kapitel 5 diskutierten Alternativen übertragen lassen. Es wird gezeigt, daß sich die Ergebnisse auf ähnliche Komponenten übertragen lassen, so daß auch diese Komponenten mit relativ geringem Aufwand in überlebensfähige Systeme überführt werden können.

Ein Resümee und eine Diskussion offener Fragestellungen wird abschließend in Kapitel 11 gegeben.

Abbildung 1.1 stellt die Vorgehensweise noch einmal graphisch dar und zeigt die wesentlichen Schritte bei der Überführung einer Packet Screen zu einer überlebensfähigen Netz-Komponente.

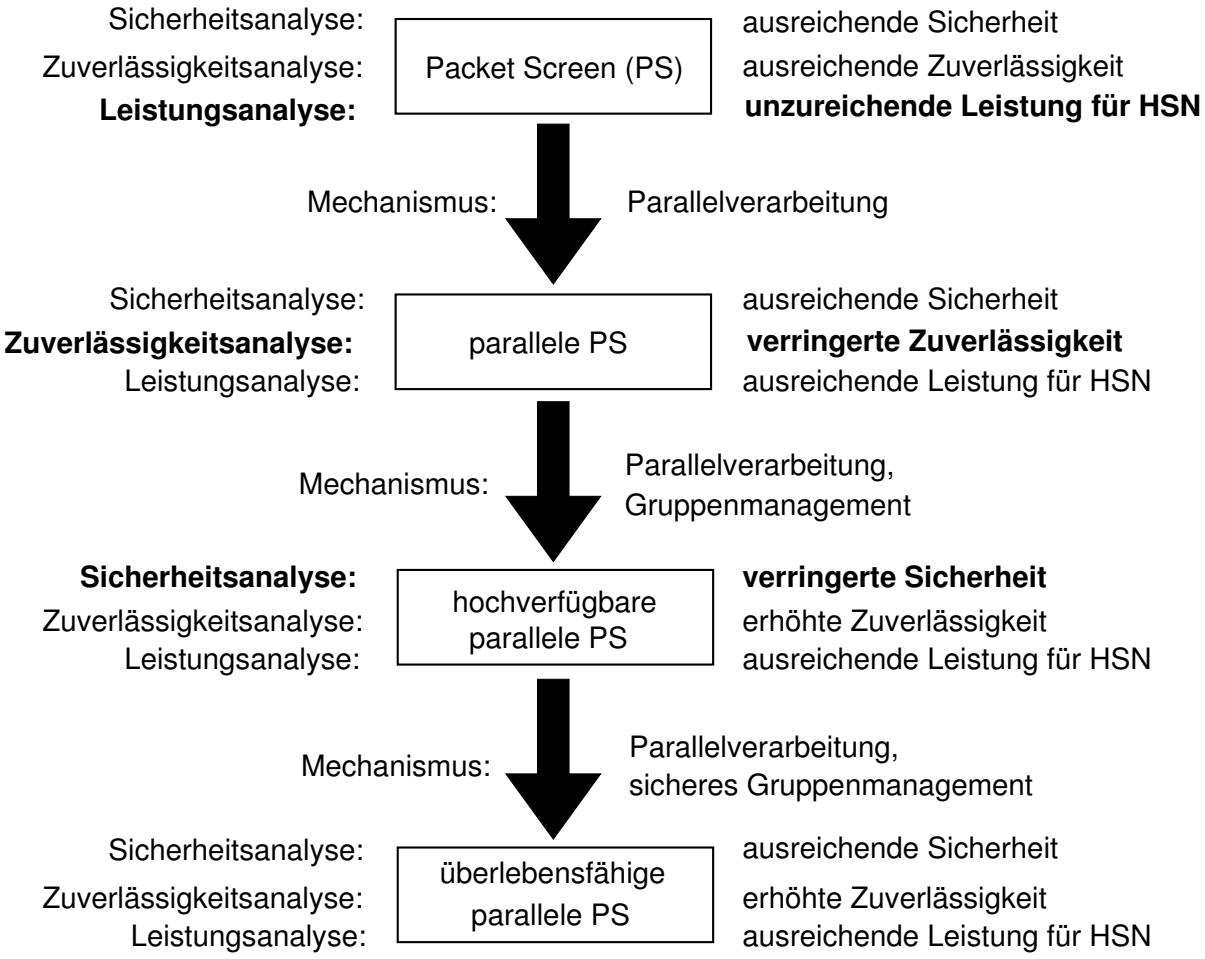


Abbildung 1.1: Überführung einer Packet Screen in eine überlebensfähige Netz-Komponente

Kapitel 2

Überlebensfähige Systeme: Notwendigkeit, Entwurfsempfehlungen und Mechanismen

Es gibt durch die zunehmende Abhängigkeit der Industrieländer von der Informationstechnologie eine steigende Anzahl an Vorfällen, bei denen Schäden durch das Versagen der Informationssysteme entstehen. Sowohl die Art der Schäden[1] als auch deren Ursachen[2] sind vielfältig. Dieses Versagen der Informationssysteme kann in vielen Fällen nicht toleriert werden, insbesondere dann nicht, wenn das Versagen lebensbedrohend ist (beispielsweise [142]) oder großer finanzieller Schaden durch die erhebliche Störung der Arbeitsabläufe entsteht.

In Abschnitt 2.1 wird an zwei Beispielen dargestellt, daß viele ursächliche Probleme durch die heute übliche Entwurfspraxis entstehen.

Anschließend wird in Abschnitt 2.2 der Begriff der *überlebensfähigen Systeme* genauer motiviert und es wird dargestellt, daß alle nicht trivialen verteilten Systeme eigentlich als überlebensfähige Systeme ausgelegt (entworfen und implementiert) werden sollten. Von der Forderung nach überlebensfähigen Systemen bis zu ihrem Einsatz ist es jedoch ein sehr langer Weg. Die Probleme liegen vor allem darin, daß es noch keine allgemein akzeptierten Methoden gibt, die beim Entwurf von überlebensfähigen Systemen helfen können. Derartige Methoden befinden sich zur Zeit in der Entwicklung [79, 155].

---

[1] Beispiele für Schäden durch das Versagen von Informationssystemen sind: physischer Schaden an Personen oder Geräten, finanzielle Schäden oder auch nicht quantifizierbare Schäden wie "Imageverlust", Vertrauensverlust usw.

[2] Beispiele für derartige Ursachen sind: Ausfall einer Komponente, Fehlbedienung, Sabotage durch Computervirus oder Angriff durch einen "Hacker".

Es wird daher am Beispiel einer Packet Screen (Baustein für Firewalls im Bereich der Netzsicherheit) gezeigt, wie der praktische Entwurf und die Implementierung zu einem überlebensfähigen System führen kann, das nicht nur sehr leistungsfähig, sondern auch hochgradig verfügbar und sicher ist. Sowohl die Beschränkung auf dieses hoch spezialisierte System, als auch die gewählte Vorgehensweise (vgl. Abschnitt 2.3), ermöglichen das Aufzeigen von Problemen, die erst bei der Implementierung von Systemen auftreten und sich somit oft den akzeptierten Entwurfs- und Validierungsmethoden entziehen. Gleichzeitig ermöglicht die Erstellung einer überlebensfähigen Netzkomponente auch deren empirische Bewertung. Eine derartige Bewertung des resultierenden Systems ist wichtig, um zu zeigen, daß die gewählte Vorgehensweise als auch die angewendeten Analyse- und Synthesemethoden praxistauglich sind.

2.1 Risiken durch die gängige Entwurfspraxis

Das Netzmanagement unterscheidet sechs Bereiche, die konzeptionelle Teilaspekte der zu verwaltenden Aufgaben in verteilten Systemen benennen. In [226, S.14] werden diese Bereiche auch Subsysteme genannt; diese Bezeichnung verdeutlicht, daß die einzelnen Teilbereiche das Netzmanagement konstituieren. Es handelt sich hierbei um *Abrechnungs-, Planungs-, Konfigurations-, Leistungs-, Fehler-* und *Sicherheitsmanagement*.

Bei der Spezifikation von neuen Anwendungen, Diensten und Kommunikationsprotokollen im Bereich der verteilten Systeme werden häufig einzelne Aspekte beim Netzmanagement zugunsten anderer bevorzugt. Die Ursachen für diese Entwicklung sind vielschichtig und werden hier nur am Rande behandelt. Die erkennbaren Auswirkungen dieser Vorgehensweise — wie beispielsweise häufige Totalausfälle wegen fehlender Redundanz oder Datendiebstahl wegen mangelnder Absicherung bei der Übertragung — sind jedoch in vielen Bereichen inakzeptabel, in fast allen Bereichen jedoch mindestens störend.

2.1.1 Fallbeispiel 1: Priorisierung des Leistungsmanagements

Ein Beispiel für die Priorisierung des Leistungsmanagements gegenüber dem Fehlermanagement ist die Entwicklung der verbreiteten LAN-Technologien seit den 90er Jahren, die von 'shared media'-Netzen zu vollständig auf 'switching' beruhenden Netzen mit einem sog. "collapsed backbone" (vgl. [18, S.516ff]) übergehen. Es ergab sich wegen der ständig steigenden Nachfrage nach höherer Übertragungsgeschwindigkeit in den einzelnen Subnetzen auch eine Nachfrage nach höherer Übertragungsgeschwindigkeit in den von allen Subnetzen genutzten 'Backbone-Netzen'. Vielerorts führte dies zur Ersetzung von FDDI-Backbones[3] durch die 'Fast Ethernet'-Technologie, mit der an einem zentralen Punkt, bei-

[3] FDDI ist durch die zwei unabhängigen Übertragungsmedien und dazugehörender Protokollfunktionalität zur Fehlererkennung und automatischer Rekonfiguration ein redundant ausgelegtes Protokoll für die OSI-Schichten 1 und 2 (vgl. [128]).

spielsweise im Rechenzentrum, der 'Backbone' in einem einzigen Switch realisiert werden kann. Durch diese Maßnahme ist in der Tat ein vergleichsweise hoher, akkumulierter Gesamtdurchsatz (in Mbit/s) möglich; durch den Ausfall dieser einen zentralen Komponente fällt jedoch mit einem Schlag der gesamte 'Backbone' aus, da Fast-Ethernet nicht über die bei FDDI vorhandenen Redundanzmechanismen verfügt.

Einhergehend mit der zunehmenden Dezentralisierung und dem Trend, immer größere verteilte Systeme zu erstellen, ist der Verzicht auf Redundanz im Kommunikationssubsystem besonders unverständlich. Einerseits beruhen die Vorteile der verteilten Datenverarbeitung immanent auf den Netzdiensten, andererseits wird gerade das unverzichtbare Netz[4] nicht-redundant ausgelegt. Es findet also eine Priorisierung des Leistungsmanagements (höherer Datendurchsatz über das Netzwerk) gegenüber dem Fehlermanagement (Zuverlässigkeit/Verfügbarkeit des Netzes) statt.

2.1.2 Fallbeispiel 2: Ignorieren des Sicherheitsmanagements

Leider ist es auch gängige Entwurfspraxis, das Sicherheitsmanagement zu vernachlässigen bzw. zu ignorieren. Im Bereich der Protokollverarbeitung ist dies besonders gut an Protokollspezifikationen der "Internet Engineering Task Force (IETF)" zu beobachten. Das Resultat der Arbeitsgruppen sind Entwürfe ("Drafts") und Protokollspezifikationen ("Requests for Comments", RFCs), bei denen häufig Sicherheitsaspekte ignoriert oder nur unzureichend betrachtet werden.

In [25] beschreibt der Autor am Beispiel der Angriffsmöglichkeiten auf einen Dienst für die Adreßauflösung in ATM-Netzen (" No special security mechanisms have been added[...]", [148, S.15]) einige der protokollinhärenten Schwächen. Diese Schwächen hätten zu einem Großteil durch eine auf Sicherheit bedachte Spezifikation vermieden werden können. Die IETF verzichtete jedoch auf entsprechende protokollspezifische Sicherheitsanalysen (vgl. Abschnitt 8). Es resultierte hieraus eine nachweislich hochgradig unsichere Protokoll- und Dienstspezifikation (vgl. [25, 119]). Das Ignorieren des Sicherheitsmanagements bei der Spezifikation von Netzprotokollen ist keine Seltenheit, wie beispielsweise in [23] nachgewiesen wird.

Zusätzlich ist zu beachten, daß durch diese Vorgehensweise ein unsicheres Protokoll bzw. ein unsicherer Dienst etabliert wird. Auch nach einer Überarbeitung (wie in [149] geschehen) wird dieses Protokoll noch Jahre lang eingesetzt werden. Darüber hinaus sah sich die IETF gezwungen, mit der überarbeiteten Version weitgehend kompatibel zur ursprünglichen Version zu bleiben. Daher sind in der neuen Version bereits einige Sicherungsmöglichkeiten angedacht, aus Kompatibilitätsgründen jedoch nicht als zwingend vorgeschrieben.

[4] Das Netz ist hier als Subsystem des verteilten Systems im Verhältnis zu anderen Komponenten des verteilten Systems zu sehen. Jedem ist klar, daß er einen anderen Drucker, einen anderen Rechner und eventuell sogar eine andere kompatible Software benutzen kann, sollte der lokale Drucker oder Rechner ausgefallen sein oder die Lieblingsanwendung wegen Speichermangel nicht mehr funktionieren. Was aber passiert, wenn das nicht-redundante Netz ausfällt?

2.2 Überlebensfähige Systeme

Das oben allgemein dargestellte und am Beispiel verdeutlichte Problem einer Vernachlässigung bestimmter Teilbereiche des Netzmanagements wird im Bereich des Sicherheitsmanagements besonders deutlich: Der Wunsch nach leichterem Zugriff (neudeutsch: "Usability") auf verteilte Ressourcen gepaart mit der Annahme, das Internet würde seinen 'kooperativen und forschungsorientierten' Charakter auch längerfristig beibehalten, führte zu einer Vielzahl neuer, nicht abgesicherter Dienste, die den Zugriff auf das Netz der Netze zwar erleichterten, damit aber auch einer neuen Klientel den Zugriff ermöglichten. Heutzutage ist das ursprüngliche Interesse an einem Forschungsnetz nur eines von vielen verschiedenen Interessen der Internet-Benutzer. Die in der Gründungszeit vorhanden gewesene, kooperative Zusammenarbeit einiger Forscher ist nur eine mögliche Handlungsform im heutigen Internet.

Eine andere, neue Handlungsform im Internet ist der elektronische Handel (E-Commerce), bei dem erstmalig die direkte Wertschöpfung aus dem Internet im Vordergrund steht. Gerade aber weil hier für jedermann ersichtlich, 'reale (Geld-)Werte' übertragen werden, steigen die Anforderungen an die Sicherheitsdienstgüte im Netz. Mit dem steigenden Interesse an Sicherungsmöglichkeiten steigt auch der (Nachhol-)Bedarf an angewandter Forschung im Teilbereich "Sicherheitsmanagement".

Es gibt nun wiederum mehrere Möglichkeiten, wie auf die Erkenntnis reagiert werden kann, daß das Sicherheitsmanagement in den vergangenen Jahren entweder zu kurz gekommen ist oder in eine falsche Richtung lief. Zwei in der Literatur vertretene, nach Meinung des Autors aber nicht angemessene Vorgehensweisen sind:

Bevorzugung von Sicherheitsmanagement: Aus der Vernachlässigung des Sicherheitsmanagements in vielen Bereichen wird teilweise der Umkehrschluß gezogen, Sicherheit müsse fortan vor allen anderen Netzmanagementaufgaben wahrgenommen werden. Dieses Vorgehen ist sicherlich in vielen Fällen übertrieben und wird dem eigentlichen Bedarf der Benutzer nur selten gerecht.

Bagatellisieren der vorhandenen Defizite: Hierbei handelt es sich eigentlich um zwei aufeinander aufbauende Aspekte. Es werden die Sicherheitsprobleme als 'neutitulieren altbekannter Phänomene' abgetan, um anschließend mit bereits vorhandenen bzw. bereits bekannten Lösungsmethoden das Sicherheitsmanagement einer anderen Teildisziplin unterzuordnen. Am Beispiel der Diskussion aus [19] kann nachvollzogen werden, wie versucht wird, Sicherheitsbedrohungen als Teilaspekte eines allgemeinen Fehlermanagements darzustellen. Es setzen sich jedoch zunehmend differenziertere Meinungen durch. Anstatt die Übertragung von Methoden aus anderen Managementdisziplinen auf das Sicherheitsmanagement zu fordern, werden die Möglichkeiten einer Adaption bewährter Methoden aus dem Bereich des Fehlermanagements für das Sicherheitsmanagement diskutiert. Es wird jedoch nicht der Anspruch erhoben, durch diese Adaption alle Sicherheitsprobleme lösen zu können [167,164]. Eine solche Vorgehensweise ist deutlich pragmatischer.

Es stellt sich somit die Frage, ob es überhaupt eine Möglichkeit gibt, a priori für alle Systeme festzulegen, welche Managementaufgaben besonders wichtig sind. Diese Frage kann wahrscheinlich nur verneint werden. Zumindest aber für die am Anfang des Kapitels erwähnten Informationssysteme, durch deren Versagen erhebliche Schäden angerichtet werden können, scheinen sich drei besonders wichtige Managementbereiche herauszubilden.

2.2.1 Definition: Überlebensfähigkeit

In einem systematischen Ansatz wurde in [181] herausgearbeitet, daß neben *Zuverlässigkeit* und *Leistungsfähigkeit Sicherheit* die Hauptanforderung an nicht triviale Systeme darstellt. Sicherheit ist somit ein wichtiger Aspekt von drei zentralen Anforderungen an sog. *überlebensfähige Systeme*.

Der Begriff "Überlebensfähigkeit", wie er hier verwendet wird, ist in der Informatik noch nicht lange bekannt. Es gibt auch noch keine einheitliche Definition für diesen Begriff. Zum Zeitpunkt der Erstellung dieser Dissertationsschrift sind allerdings zwei sehr ähnliche Definitionen bekannt, die als Ausgangsbasis für die vorliegende Arbeit dienen können und daher näher betrachtet werden sollen:

Definition (2.1): **Survivability (a)**

> "We define survivability as the capability of a system to fulfill its mission, in a timely manner, in the presence of attacks, failures, or accidents. We use the term system in the broadest possible sense, including networks and large-scale systems of systems." (vgl. [80])

Eine dieser Definition sehr ähnliche Aussage ist:

Definition (2.2): **Survivability (b)**

> "[...] survivability is the ability of a computer-communication system-based application to satisfy and to continue to satisfy certain critical requirements (e.g., specific requirements for security, reliability, real-time responsiveness, and correctness) in the face of adverse conditions. Survivability must be defined with respect to the set of adversities that are supposed to be withstood." (vgl. [181])

Die Kernaussage beider Definitionen ist, daß ein (verteiltes) System auch unter erschwerten Bedingungen weiterhin in der Lage sein muß, bestimmten Aufgaben nachzukommen. Während Definition (2.1) explizit Angriffe, Fehler und Unfälle als erschwerte Bedingungen nennt, beschränkt sich Definition (2.2) auf Anwendungen in verteilten Systemen ("computer-communication system-based application") und ist etwas pragmatischer, da sie eine Angabe

der Faktoren ermöglicht, gegenüber denen die Überlebensfähigkeit aufrecht erhalten bleiben soll.

Definition (2.1) ist somit strenger, da sie keine Einschränkung auf bestimmte Faktoren ("adversities") zuläßt, sondern die bedingungslose Diensterbringung im Sinne einer Zielvorgabe ("mission") fordert.

Zu beiden Definitionen ist auch noch anzumerken, daß sich die Überlebensfähigkeit auf die jeweilige Aufgabenerfüllung ("mission") bzw. auf die vom System auszuführende Anwendung ("application") bezieht, nicht jedoch auf das System als solches. Dies ist eine relativ neue[5] Interpretation des Begriffes "überlebensfähiges System".

Im folgenden soll im wesentlichen nach der Definition (2.2) vorgegangen werden. Die bedingungslose Erfüllung einer Aufgabe laut Definition 2.1 erscheint als nicht praktikabel und ist eher als ideales Ziel vorstellbar.

2.2.2 Anforderungen an überlebensfähige Systeme

Aus den obigen Definitionen lassen sich direkt Anforderungen an überlebensfähige Systeme ableiten. Um Angriffen von Computer-Kriminellen (vgl. [64]) widerstehen zu können, müssen die Systeme über Sicherungsfunktionen verfügen. Fehlern durch Unfälle, Verschleiß oder technisch bedingten Problemen wird traditionell durch eine fehlertolerante Auslegung der Systeme begegnet. Die zeitnahe Reaktion ("timely manner", "real-time responsiveness") ist letztendlich eine Anforderung an hohe Verarbeitungsgeschwindigkeiten bei der Auftragserfüllung.

Überlebensfähige System müssen daher grundsätzlich (also weitgehend unabhängig von der Aufgabe) in Hinblick auf Leistungsfähigkeit, Fehlertoleranz und Sicherheit optimiert werden. Diese grundsätzlichen Anforderungen werden an die untergeordneten "Disziplinen" gestellt ("subdisciplines" in [179]):

[5] Bereits 1983 wurde von H.M. Deitel in [63, S.453f] der Begriff "survivable system" folgendermaßen definiert: "A survivable computer system is one that continues to operate even after one or more of its components fail". Interessant hieran ist zweierlei: Erstens bezieht sich diese Definition direkt auf das jeweilige System und nicht auf die Aufgabe bzw. die "Mission". Zweitens wurde in der Neuauflage der Publikation in [62, S.536] der Begriff "survivable system" durch "fault-tolerant system" ersetzt. In der Tat beschreibt die Definition von Deitel fehlertolerierende Systeme. Seine Motivation für die Diskussion von Sicherheitsaspekten war jedoch bereits damals, die "andauernde und zuverlässige Funktionsfähigkeit und Verfügbarkeit" von Systemen zu gewährleisten. Wahrscheinlich bestanden schon 1984 erhebliche Zweifel daran, daß dies alleine durch Mechanismen des Fehlermanagements erreicht werden kann.

Sicherheit: Sicherheit wird hier im Sinne von "Sicherheit vor Angriffen" ([138]) verstanden.[6] Der Begriff "Sicherheit" wird in der vorliegenden Arbeit ebenfalls im Sinne der "Sicherheit vor Angriffen" verwendet.

Leistungsfähigkeit: Leistungsfähigkeit wird hier ausschließlich unter dem Aspekt der hohen Verarbeitungsgeschwindigkeit bzw. der Effizienz bei der Auftragsbearbeitung betrachtet. Diese Einschränkung auf "geleistete Arbeit pro Zeiteinheit" wird durch die explizite Nennung von "Echtzeitanforderungen" in beiden Definitionen der Überlebensfähigkeit deutlich ("timely manner", "real-time responsiveness"). In der vorliegenden Arbeit wird der Begriff der "Leistungsfähigkeit" ebenfalls in diesem Sinn verwendet. Funktionale Aspekte (Leistungsfähigkeit im Sinne der Funktionsvielfalt) werden nicht betrachtet, da sie auch nicht unabhängig von der Aufgabe angegeben werden können.

Zuverlässigkeit: Das System bleibt trotz auftretender Fehler verfügbar. Der Ausfall einer Komponente führt nicht notwendigerweise zum Ausfall des gesamten Systems. Eine möglichst große Anzahl von Fehlern kann somit "toleriert" werden.

Diese "subdisciplines" können jedoch nicht unabhängig voneinander betrachtet werden. Vielmehr bedingen sich die Anforderungen an Sicherheit, Verfügbarkeit und Leistungsfähigkeit in vielfacher Hinsicht gegenseitig.

Eine hohe Verfügbarkeit der einzelnen Komponenten eines Systems ist nicht ausreichend, wenn durch einen Angriff der Zugriff auf das System oder die Daten verhindert werden kann, beispielsweise indem ein Angreifer das System für legitime Benutzer sperrt. Die Verfügbarkeit stellt auch Mindestanforderungen an die Leistungsfähigkeit. Der Zugriff auf Informationen muß beispielsweise so schnell möglich sein, daß die Arbeitsabläufe nicht gestört werden. Zu lange Wartezeiten beim Abruf von Informationen schränken die Verfügbarkeit ein. Leistungsfähigkeit erfordert jedoch auch die Verfügbarkeit. Fallen zu viele Komponenten gleichzeitig aus, kann die Auftragsverarbeitung möglicherweise nicht mehr zeitnah erfolgen. Weitere Beispiele für diese gegenseitigen Abhängigkeiten werden in [181, §3] gegeben.

Überlebensfähige Systeme erfordern jedoch mehr als nur die Summe bestimmter Eigenschaften aus den Bereichen Sicherheit, Fehlertoleranz und Leistungsfähigkeit. Es ist insbesondere zu nennen (vgl. [80, 155, 181]):

Robustheit bei Angriffen Das Sicherheitsmanagement sieht als Maßnahmen traditionell die Erkennung und die Verhinderung von Angriffen vor. Das Absichern (Härten) von Systemen ist jedoch nicht hinreichend für überlebensfähige Systeme. Vielmehr

[6] Das deutsche Wort "Sicherheit" umfaßt hier die englischen Begriffe "Safety" und "Security". Eine bekannte Unterscheidung aus [40] bezeichnet beispielsweise ein System als 'safety critical', falls ein Fehler in diesem System unmittelbar zu einem Schaden führt. Das gleiche System ist 'security critical' falls ein Fehler im System die Möglichkeit bietet, mit diesem System (intentional) Schaden anzurichten.

sollen die Systeme trotz des Angriffes weiterhin (eventuell mit gewissen Einschränkungen) benutzbar sein. Das Verhindern oder Erkennen eines Angriffes darf also nicht dazu führen, daß die gesamten Betriebsmittel des Systems für die Aufgaben des Sicherheitsmanagements gebunden werden. Es darf höchstens zu einer Einschränkung ("degradation") der normalen Auftragsbearbeitung während eines Angriffs kommen. Darüber hinaus ist zu fordern, daß sich das überlebensfähige System nach dem Ende eines Angriffs hiervon wieder erholt.

Zuverlässigkeit trotz provozierter Fehler Das Fehlermanagement setzt traditionell auf Basis von Fehlermodellen bestimmte Mechanismen ein, die die Auswirkungen der als "wahrscheinlich" erkannten Fehler bis zu einem gewissen Grad auffangen können (hierzu mehr in Kapitel 6). Die grundsätzliche Vorgehensweise ist bei dieser Methode, das Auftreten von Fehlern durch Zufallsvariablen zu beschreiben. Die aufgrund des Fehlermodells als sehr unwahrscheinlich (vernachlässigbar) eingestuften Fehler werden dann nicht weiter betrachtet. Ein absichtliches Herbeiführen eines oder mehrerer dieser als vernachlässigbar klassifizierten Fehler kann dennoch zu einem Ausfall eines Systems führen. Diese durch einen neugierigen Benutzer oder einen Angreifer herbeigeführten Fehler entziehen sich ergo vollständig dem klassischen Fehlermanagement. Beispielsweise ist die redundante Auslegung (als typischer Mechanismus des Fehlermanagements) dann nicht ausreichend, da derart herbeigeführte Fehler beliebig oft auf den redundanten Komponenten wiederholt werden können.

Zeitnahe Reaktion Die Leistungsanforderungen beziehen sich typischerweise auf die Abarbeitung von Benutzer-Aufträgen. Ein überlebensfähiges System erfordert darüber hinaus auch die zeitnahe Reaktion auf erkannte Angriffe, Einbrüche und Ausfälle. Da die Sicherheit eines Systems nie absolut garantiert werden kann, ist im Sinne einer Schadensminimierung bei Erkennen eines erfolgreichen Angriffs eine schnelle Reaktion erforderlich. Das überlebensfähige System muß somit auch in der Lage sein, zeitnah auf erkannte Angriffe reagieren zu können. Dies erfordert neben der Erkennung von Angriffen auch im Voraus definierte Reaktionen, die dann automatisch ablaufen können.

Klassifizierung der Funktionalität Es stellt sich unmittelbar die Frage, wie die "mission" (vgl. Definition 2.1) trotz potentieller Einschränkungen durch Angriffe, Fehler oder erforderliche Abwehrreaktionen erfüllt werden kann. Beim Entwurf von überlebensfähigen Systemen ist es daher erforderlich, die essentiellen Funktionen von den unkritischen Funktionen zu unterscheiden. Diese Einteilung kann auch mehrstufig sein, von sehr wichtiger Funktionalität zu optionaler Funktionalität. Durch diese Priorisierung innerhalb verschiedener Auftragsklassen kann erreicht werden, daß ein überlebensfähiges System sich bei Auftreten der "widrigen Umstände" (vgl. Definition 2.2) auf die Erfüllung der essentiellen Funktionalität konzentriert. Werden Betriebsmittel ggf. für die Reaktion (Abwehrmaßnahmen, aufwendigeres Logging, Reorganisation, usw.) benötigt, können diese den Prozessen entzogen werden, die mit weniger wichtigen Aufgaben betraut sind. Im Idealfall bezieht sich somit eine

Schwächung ("degradation") nie auf die essentiellen Funktionen, sondern höchstens auf die optionalen Funktionen.

Die Aufgabe ("mission") ist indes systemspezifisch. Die Klassifikation der Funktionalität kann daher nur im jeweiligen Einzelfall erfolgen.

2.2.3 Primäre Eigenschaften von überlebensfähigen Systemen

Die oben dargestellten, allgemeinen Anforderungen an überlebensfähige Systeme kann man wie in [80] auch über Schlüsseleigenschaften (primäre Eigenschaften) herleiten. Hierbei handelt es sich um die Eigenschaften, die die Überlebensfähigkeit eines Systems konstituieren:

Widerstand (Resistance) Diese Eigenschaft wird im wesentlichen durch die bekannten Mechanismen aus dem Bereich des Sicherheitsmanagements herbeigeführt. Die Widerstandsfähigkeit macht einen Großteil der Robustheit aus, da mit den bekannten Mechanismen aus dem Bereich des Sicherheitsmanagements bereits viele Angriffe abgefangen werden können.

Erkennung (Recognition) Hier werden klassische Monitoring-Verfahren zur Überwachung des Systems aus dem Bereich des Fehlermanagements eingesetzt und mit neuen Verfahren aus dem Bereich des Sicherheitsmanagements kombiniert. Insbesondere werden Verfahren zur Erkennung von Einbrüchen (intrusion detection system, IDS) eingesetzt. Die Erkennung ergänzt die Widerstandsfähigkeit. Auch dies ist eine bekannte Technik aus dem Sicherheitsmanagement, bei der schon immer die zwei Schwerpunkte Verhinderung und Erkennung von Angriffen propagiert wurden [187]. Die Erkennung muß auch in der Lage sein, das Ende eines Angriffs zu erkennen, damit möglichst automatisch ein "Recovery" eingeleitet werden kann.

Wiederherstellung (Recovery) Ist ein Angriff beendet, dann muß das System in der Lage sein, sich von diesem Angriff schnellstmöglich zu erholen. Das System muß ausgefallene oder kompromittierte Komponenten und Daten erkennen und durch geeignete Maßnahmen versuchen, einen ordentlichen Betrieb wieder herzustellen. Ein Beispiel hierfür ist die automatische Löschung kompromittierter Systemsoftware mit anschließender Neuinstallation.

Anpassung und Entwicklung (Adaption and Evolution) Während eines Angriffs muß ein System versuchen, sich an diesen Angriff anzupassen. Beispielsweise könnten weitere Komponenten zur Abwehr und Überwachung eingeschaltet werden, unwichtigeren Programmen könnten zur Unterstützung der Abwehr Betriebsmittel entzogen werden, oder es könnten automatisch die Administratoren benachrichtigt werden, falls das System eine Gefahr erkennt, für die keine automatische Gegenmaßnahme spezifiziert wurde.

Das System muß also ständig an neue Bedrohungen angepaßt werden. Dies erfordert eine sorgfältige Auswertung der Angriffe aber auch der technisch bedingten Fehler. Im

Idealfall findet eine regelmäßige Evaluierung des Systems in Hinblick auf neu erlangte Erkenntnisse (neue Bedrohungen, zusätzliche Anforderungen, veränderte Umgebung) statt.

2.2.4 Entwurfsempfehlungen für überlebensfähige Systeme

Ausgangspunkt für den Entwurf eines überlebensfähigen Systems ist eine Analyse der "Ist-Situation". Bestehende Systeme (bzw. die als essentiell oder auch als "mission critical" klassifizierten Dienste [155]) werden daraufhin untersucht, welche Eigenschaften sie bezüglich "Resistance", "Recognition" und "Recovery" aufweisen. In [79] wird empfohlen, dann die sog. "usage scenarios" und "intrusion scenarios" auf Grundlage der "Ist-Situation" zu evaluieren. Bei den "usage scenarios" handelt es sich um die erwünschte Dienst/Systembenutzung. Bereits hier können Unzulänglichkeiten der "Ist-Situation" auffallen (beispielsweise mangelnde Fehlererkennung bei Ausfällen von Komponenten).

Die Anwendung der "intrusion scenarios" soll hingegen Aufschluß über die Defizite der "Ist-Situation" in Hinblick auf mögliche Angriffe geben. Schließlich resultiert dieser Prozeß in einem Maßnahmenkatalog mit Empfehlungen ("architecture strategies"), wie die erkannten Defizite in den drei Bereichen "Resistance", "Recognition" und "Recovery" auszugleichen sind und in welcher Form eine Anpassung und Entwicklung des Systems erfolgen sollte.

Kritisch ist hierbei die Erstellung der sog. "scenarios", also die Antizipation möglicher Angriffe ("intrusion scenarios") auf das System bzw. auf die essentiellen Dienste, sowie die Analyse möglicher Fehler bei den Arbeitsabläufen ("usage scenarios"), die (ungewollt) zu Schaden führen können. Werden hier potentiell erfolgreiche Angriffsmöglichkeiten und Fehlerpotentiale innerhalb der Arbeitsabläufe nicht erkannt, oder fälschlicherweise als vernachlässigbar klassifiziert, finden sie keine Berücksichtigung bei den Verbesserungsvorschlägen. Es können also durchaus Verwundbarkeiten und Fehlerquellen übrig bleiben. Hilfreich könnten bei der Erstellung der "scenarios" Methoden aus dem Bereich der klassischen Risikoanalyse sein (vgl. [187, 102]).

Beim anschließenden Entwurf müssen die empfohlenen Mechanismen umgesetzt werden, um das System in ein überlebensfähiges Pendant zu überführen. Für diesen Schritt gibt es eine Reihe von Entwurfsempfehlungen, die hier kurz vorgestellt werden (vgl. [181]):

Einsatz formaler Methoden Formale Methoden helfen bei der Spezifikation und Validation von Systemen [211, 144]. Typischerweise zeigt man mit formalen Methoden, daß eine Spezifikation konform mit den Anforderungen ist bzw. daß bestimmte unerwünschte Seiteneffekte nicht auftreten können. Formale Methoden können somit auch als Ergänzung zu Tests (s.u.) eingesetzt werden, um die Abwesenheit von bestimmten Fehlern zu zeigen.

Verantwortungsvolle Softwareentwicklung Bei der Entwicklung von Software sollten Hochsprachen eingesetzt werden (strenge Typprüfung). Eine gute Dokumentation

der Software ist unerläßlich. Darüber hinaus sollte im Zuge systematischer Softwaretests versucht werden, möglichst viele Fehler systematisch auszuschließen und nicht — wie heute gängige Praxis — ausschließlich zufällig aufgefundene Fehler zu beseitigen (vgl. [154, S.220], "debugging code into existence"). Eine Technik hierfür ist die inkrementelle Softwareentwicklung. Es sollten aber auch Werkzeuge für das automatische Testen von Programmen eingesetzt werden, beispielsweise um die Reaktion auf mögliche Fehleingaben zu prüfen.

Modularisierung Kleine, modular aufgebaute Subsysteme sollten so entworfen werden, daß sie möglichst einfach miteinander kombiniert werden können. Idealerweise lassen sich so aus gut verstandenen, nachweisbar fehlerfreien Komponenten größere Systeme zusammenstellen. Der modulare Aufbau kann auch die Rekonfiguration der Systeme —beispielsweise im Zuge eines erkannten Angriffs— vereinfachen, indem unsichere Komponenten (oder ganze Subsysteme) dynamisch aus dem laufenden System entfernt werden.

Offenlegung von Quellcodes Die Verfügbarkeit von Quellcodes ist in mehrfacher Hinsicht wichtig für überlebensfähige Systeme. Erstens können andere Experten[7] zur Fehleranalyse und Fehlerfindung beitragen. Zweitens können gefundene Fehler schnell beseitigt werden, da nicht auf eine Fehlerbehebung durch den Hersteller gewartet werden muß. Vielmehr kann jeder — geeignete Kenntnisse vorausgesetzt — einen Fehler beheben. Drittens kann die Güte der eingesetzten Algorithmen beurteilt werden. Die Offenlegung der Quellcodes ist somit eine gute Methode, um vertrauenswürdige Software zu entwickeln. Im Gegensatz hierzu ist es heutzutage üblich, Software einzusetzen, deren Quellcodes nicht eingesehen werden können. Keiner der oben genannten Vorteile trifft auf diese Art von Software zu. Allerdings spricht gegen die Offenlegung von Quellcodes, daß dadurch auch Computer-Kriminelle in die Lage versetzt werden, Schwachstellen zu analysieren und gezielt auszunutzen.

Neben diesen Entwurfsempfehlungen, die unmittelbar zum Systemdesign eingesetzt werden sollten, werden in [181] weitere Empfehlungen gegeben, die mittelbar der Erstellung von überlebensfähigen Systemen dienen. Ein Beispiel dafür ist die "gute Ausbildung und Schulung" von Programmierern und Entwicklern.

Bei der vorgeschlagenen Vorgehensweise bleiben viele Fragen ungeklärt. Zwar wird postuliert, daß die "Ist-Situation" eines Systems evaluiert werden soll. Es ist jedoch weitgehend unklar, wie dies zu erfolgen hat. Es wird nicht gesagt, wie die sogenannten "intrusion scenarios" aufgestellt werden. Möglicherweise ergeben sie sich als Teil einer Risikoanalyse (vgl. [187, 102, 138]). Für die Synthese werden formale Methoden empfohlen. Wünschenswert sind hierfür durchgängige Methoden, die sowohl Analyse, Synthese als auch Tests in den drei Bereichen Fehler-, Sicherheits- und Leistungsmanagement unterstützen. Zur Zeit

[7] Mit "anderen Experten" sind hier alle erfahrenen, nicht an der Erstellung der entsprechenden Software beteiligten Personen, gemeint.

sind jedoch keine formalen Methoden bekannt, die Leistungsfähigkeit und Sicherheit oder Verfügbarkeit und Sicherheit gleichermaßen betrachten. Es gibt lediglich Methoden, bei denen einige Bereiche des Fehlermanagements mit dem Leistungsmanagement kombiniert werden. In der Literatur wird diese Vorgehensweise häufig mit dem Schlagwort "Performability" (vgl. beispielsweise [227]) bezeichnet. Der Einsatzschwerpunkt dieses Ansatzes scheint bei der Analyse von Echtzeitsystemen zu liegen (vgl. beispielsweise [235]). Die häufig für die gleichzeitige Analyse von Zuverlässigkeit und Leistungsfähigkeit verwendeten Markov-Modelle und stochastischen Petrinetz-Modelle/Prozeßalgebra-Modelle scheinen wenig geeignet zu sein, gleichzeitig auch noch Sicherheitsaspekte zu modellieren. Der Grund hierfür ist, daß sich Sicherheit (im Sinne einer Abwehr von Angriffen, s.o.) kaum durch zufällige Prozesse beschreiben läßt, die häufig die Grundlage für die Modellierung von "Performability"-Aspekten sind (vgl. beispielsweise [110, 174, 227]).

Bei den vorzunehmenden Anpassungen ist unklar, wie widersprüchliche Optimierungsziele aufgelöst werden können. Es wurde zwar dargestellt, daß in vielen Systemen sowohl die Leistungsfähigkeit, die Sicherheit als auch die Verfügbarkeit wünschenswert sind, sich sogar teilweise gegenseitig bedingen; es ist jedoch genauso gut denkbar, daß eine Leistungsoptimierung im Widerspruch zu Sicherheitsanforderungen steht oder die Verfügbarkeit unter zu restriktiver Sicherheit leidet.

Es scheint erforderlich zu sein, derartige Fragestellungen, insbesondere die Darstellung und Auflösung von "Tradeoffs" zwischen den verschiedenen Optimierungszielen am Beispiel zu diskutieren.

2.3 Überlebensfähigkeit am Beispiel der Packet Screen

Der Entwurf und die Realisierung überlebensfähiger Systeme ist zwar konzeptionell durchaus verstanden und intuitiv plausibel, die tatsächliche Umsetzung und Bewertung solcher Systeme jedoch selten (ein Fallbeispiel wird in [79] gegeben). Ein Grund für diesen Mangel an Erfahrung ist sicherlich die Komplexität der erforderlichen Analyse- und Entwurfsschritte (vgl. Abschnitt 2.2.4), die u.a. darauf zurückzuführen ist, daß Wissen und Erfahrungen aus verschiedenen Teildisziplinen (mindestens Sicherheitsmanagement, Fehlermanagement, Leistungsmanagement) erforderlich sind. Ein anderer Grund mag sein, daß die vorgeschlagene Methode (vgl. Abschnitt 2.2.4) immer in Hinblick auf ein konkretes System und dessen Aufgabe ("mission") angewendet wird. Dies hat zur Folge, daß die vorgenommenen Analyse- und Syntheseschritte (Systemanpassungen) sehr speziell und wenig allgemein sind.

Damit die potentiell sehr aufwendigen Analyse- und Anpassungsschritte im Rahmen einer Dissertation bewältigt werden können, muß für die Untersuchung ein kleines (Sub-)System gewählt werden, das nicht zu komplex ist. Nur so kann erreicht werden, daß die notwendigen Analysen nicht oberflächlich ausfallen und Zeit für Anpassungen vorhanden ist. Gleichzeitig sollte dieses System für viele Einsatzbereiche relevant sein. So besteht zu-

mindest die Aussicht, daß die erarbeiteten Ergebnisse teilweise unabhängig vom speziellen Einsatzbereich sind bzw. sich leichter an ähnliche Anwendungsfälle adaptieren lassen.

Die vorzunehmenden Untersuchungen sollten möglichst in Hinblick auf allgemeinere Szenarien, d.h. nicht in Hinblick auf eine spezielle "Mission", durchgeführt werden. Dadurch sind die erarbeiteten Ergebnisse möglicherweise an eine Vielzahl spezieller Szenarien mit vergleichsweise geringem Aufwand adaptierbar.

Diese allgemeiner ausgelegten Untersuchungen an einem wichtigen Subsystem ermöglichen dann den Einsatz dieses Subsystems in vielen speziellen Szenarien, da das Subsystem aufgrund seiner eigenen Überlebensfähigkeit in vielen Umgebungen besser in andere Systeme integriert werden kann, als ein nicht-überlebensfähiges Subsystem. Soll nämlich auch das Gesamtsystem überlebensfähig sein, dann sind solche Komponenten besonders wertvoll, die innerhalb ihres Aufgabenbereichs bereits überlebensfähig sind und nicht erst durch die Zusammenstellung mit anderen Subsystemen überlebensfähig gemacht werden sollen. Bei der Zusammenstellung der überlebensfähigen Subsysteme kann man sich daher auf die bei der Zusammenstellung entstehenden Probleme konzentrieren (vgl. [117, 116, 1]). Mit anderen Worten: Überlebensfähigkeit im kleinen (Subsystem) fördert nicht nur die Überlebensfähigkeit des Gesamtsystems, sondern auch dessen Zusammenstellung ("composing").

2.3.1 Beispiel: Überlebensfähige Firewall-Komponenten

Als Beispiel wird eine sehr wichtige Firewall-Komponente, die sog. "Packet Screen", ausgewählt. Es handelt sich hierbei um ein essentielles Element aus dem Bereich der Netzsicherheit, das in vielen gängigen Firewall-Architekturen eingesetzt wird und die Aufgabe hat, eine grobe Zugriffskontrolle durchzuführen (für eine Präzisierung des Begriffs "Packet Screen", vgl. Definition 2.6).

Die Entscheidung, eine Packet Screen in ein überlebensfähiges System zu überführen, hat zwei wesentliche Gründe. Im folgenden Abschnitt wird verdeutlicht, daß Firewalls einer der wesentlichen Zugriffskontrollmechanismen im Bereich der Netzsicherheit sind. Diese Motivation alleine reicht aus, um jede mögliche Firewall-Komponente dahingehend zu untersuchen, ob sie in ein überlebensfähiges System überführt werden kann. In Abschnitt 2.3.1.3 wird daher motiviert, daß es besonders sinnvoll ist, von den vielen verschiedenen Firewall-Komponenten gerade die Packet Screen (Subsystem vieler Firewalls) in ein überlebensfähiges System zu überführen.

Der zweite Grund ist, daß eine typische Packet Screen ein System fast ohne Benutzerinteraktionen[8] ist. Viele allgemeine Betrachtungen bezüglich der oben skizzierten Anwendungsfälle ("usage scenarios") brauchen daher nicht durchgeführt zu werden. Eine typische Packet Screen ist für Benutzer vielmehr fast immer transparent. Der Analyseaufwand verringert sich dadurch erheblich.

[8] Die Administration ist hier der einzige zu betrachtende Benutzerzugriff auf die Packet Screen.

2.3.1.1 Firewalls für Zugriffskontrolle und Audit

Die folgende Definition für den Begriff "Firewall" ist sehr allgemein gehalten und wird anschließend noch verdeutlicht:

Definition (2.3): **Firewall**

> Firewalls sind logische Komponenten im Bereich der Netzsicherheit, deren Aufgabe es ist, den gesamten Netzverkehr zwischen zwei oder mehreren Sicherheitsdomänen entsprechend festgelegter Richtlinien (Policies) zu überwachen. Die Überwachung hat mindestens die beiden Teilaufgaben Zugriffskontrolle (Verhindern unerwünschter Kommunikation) und Audit (Aufzeichnen sicherheitsrelevanter Informationen).

Diese allgemeine Definition ist weitgehend identisch mit etablierten Definitionen aus [46, 47, 74, 75, 24] und berücksichtigt darüber hinaus die heute zunehmend voranschreitende Dezentralisierung der Firewall-Architekturen [76,71]. Die Verwendung des Begriffs "Sicherheitsdomäne" soll verdeutlichen, daß die Zugriffskontrolle auf Basis der Richtlinien erfolgen soll. In der Praxis ergibt sich hieraus oft die auch in [74, S. 10] formulierte "Schwelle zwischen zwei (Rechner-)Netzen". Gerade aber bei verteilten Firewall-Installationen ist diese "Schwelle" nicht mehr physikalisch, sondern nur noch logisch ausgelegt (vgl. [71, S.199f]); die Sicherheitsdomäne ist hingegen in beiden Fällen durch die einheitlichen Richtlinien in ihrem Geltungsbereich definiert:

Definition (2.4): **Sicherheitsdomäne**

> Eine Sicherheitsdomäne ist der Geltungsbereich einheitlicher (Sicherheits)-Richtlinien (Policies) (in Anlehnung an [83, 82]).

Eine genauere Betrachtung der verschiedenen Definitions- und Interpretationsmöglichkeiten für "Sicherheitsdomäne" erfolgt beispielsweise in [234].

In der Definition (2.3) wird ein Firewall als logische Komponente bezeichnet. Damit soll verdeutlicht werden, daß es sich bei einem Firewall um ein Konzept für Zugriffskontrolle und Audit handelt. Dieses Konzept kann in verschiedenartigen Ausprägungen umgesetzt (implementiert) werden. Hierzu werden einzelne Firewall-Komponenten zu einer Firewall-Installation (Implementierung eines Firewalls) zusammengeführt, um gemeinsam die Richtlinien durchzusetzen. Die Vielfalt der Firewall-Installationen ist eine direkte Konsequenz aus den individuellen Richtlinien, die jeweils umgesetzt werden müssen, und den verschiedenen Alternativen für die Umsetzung einer spezifischen Richtlinie.

Es gibt jedoch unter den verschiedenartigen Firewall-Installationen ein paar grundlegendere Firewall-Architekturen, die häufig direkt für eine Installation übernommen werden oder nur noch geringfügig an die individuellen Bedürfnisse angepaßt werden müssen:

Definition (2.5): **Firewall-Architektur**

> Eine Firewall-Architektur ist eine Schablone (Template) für den Aufbau einer Firewall-Installation. Die Firewall-Architektur besteht aus einer oder mehreren Firewall-Komponenten, die zusammen die individuellen Richtlinien umsetzen. Typische Firewall-Komponenten sind Packet Screens und Bastionen sowie Sicherheits-Proxies, Log/Audit-Server und Schlüssel-Server.

Die am häufigsten verwendeten Firewall-Komponenten sind Packet Screen, Bastion und Sicherheits-Proxy:

Definition (2.6): **Packet Screen**

> Eine Packet Screen ist eine Firewall-Komponente. Die Zugriffskontrolle der Packet Screen erfolgt auf Grundlage von Filterregeln. Die Regeln beschreiben durchzuführende Aktionen (typischerweise: Verwerfen, Umleiten, Weiterleiten, Protokollieren), die auf die empfangenen Dateneinheiten ("Protocol Data Units", PDUs) in Abhängigkeit von Attributen dieser PDUs (typischerweise: Adreßangaben, Dienstzugangspunkt, Längenangaben, PDU-Typ) anzuwenden sind.

Definition (2.7): **Sicherheits-Proxy**

> Ein Sicherheits-Proxy (kurz Proxy) ist ein Prozeß, der stellvertretend für einen Dienst von Client und Server kontaktiert wird und eine Dienst-spezifische Zugriffskontrolle und ein Audit durchführt. Der Proxy ist der Endpunkt der Kommunikationsbeziehungen und verhindert somit eine direkte Datenübertragung zwischen Client und Server.

Durch die Anwendungsnähe ist ein Proxy in der Lage, eine sehr detaillierte Zugriffskontrolle und ein ebenso ausführliches Audit durchzuführen. Damit bieten Proxies in der Regel ein höheres Maß an Sicherheit als der alleinige Einsatz einer Packet Screen.

Definition (2.8): **Bastion**

> Eine Bastion ist eine Firewall-Komponente, die als Ausführungsumgebung (Plattform) für Sicherheits-Proxies dient. Sie stellt den einzig möglichen Übergang für Kommunikationsbeziehungen zwischen Client und Server verschiedener Sicherheitsdomänen bezüglich der durch die Proxies vertretenen Dienste dar. Aufgrund dieser exponierten Position müssen Bastionen besonders gut gesichert sein (vgl. [24]).

Für eine Diskussion der Funktionalität und Aufgaben der oben genannten Firewall-Komponenten vgl. [46, 47, 72, 24]. In Abschnitt 2.3.1.3 werden häufig verwendete Firewall-Architekturen vorgestellt.

2.3.1.2 Bedeutung der Firewalls für die Netzsicherheit

Der Firewall ist der wichtigste Zugriffskontrollmechanismus im Bereich der Netzsicherheit. Ein Firewall erlangt seine Bedeutung dadurch, daß er wesentliche Teile der Richtlinien einer Sicherheitsdomäne stellvertretend für andere Netzkomponenten dieser Sicherheitsdomäne umsetzt. Der Firewall ermöglicht somit die Durchsetzung von Richtlinien innerhalb einer Sicherheitsdomäne, falls die individuelle, lokale Absicherung jeder einzelnen Netzkomponente administrativ zu aufwendig oder nicht möglich ist und ergänzt darüber hinaus andere Schutzmechanismen.[9]

Wichtige Beispiele für die Notwendigkeit von Firewalls sind (vgl. [24]):

- Begrenzen der Auswirkungen von Administrationsfehlern und Softwarefehlern: Durch Administrations– und Softwarefehler kann es zu unerwünschter Kommunikation einzelner Rechner einer Sicherheitsdomäne kommen, die dann durch den Firewall lokal begrenzt werden. Hierzu gehört auch der klassische Einsatz von Firewalls zur Zugriffskontrolle auf nicht weiter gesicherte Rechner. Die Möglichkeit zur Vernachlässigung der Administration der einzelnen Endgeräte war lange die Hauptmotivation für den Einsatz von Firewalls. Diese Vernachlässigung bzw. unzureichende Administration ist ein Sonderfall der Administrationsfehler.

- Begrenzen der Auswirkungen von 'Denial of Service' (DoS)-Angriffen auf Rechner in der Sicherheitsdomäne: Insbesondere dann, wenn Rechner innerhalb einer Sicherheitsdomäne aufwendige Dienste anbieten, oder die angebotenen Dienste bekanntermaßen anfällig gegen DoS-Angriffe sind, kann ein Firewall bei Erkennen solcher Angriffe stellvertretend die Abwehr übernehmen und somit die Auswirkungen minimieren (beispielsweise [207, 103, 115, 209]).

- Durchsetzen von Verschlüsselungsrichtlinien: Die Möglichkeit des Einsatzes von kryptographischen Verfahren zur Absicherung der Kommunikation durch entsprechend installierte Hard- und Software garantiert noch lange nicht deren Verwendung bei

[9] Zusätzlich zur Zugriffskontrolle durch einen Firewall können einzelne Kommunikationsbeziehungen durch kryptographische Verfahren abgesichert werden. Dadurch ist es möglich, die Vertraulichkeit der übertragenen Daten und der Kommunikationsbeziehung, die Integrität der übertragenen Dateneinheiten und die Authentizität der Dateneinheiten und Kommunikationspartner bei der Kommunikation zu wahren bzw. Verstöße gegen diese Sicherungsziele zu erkennen.
Der Einsatz von kryptographischen Verfahren ist jedoch weitestgehend orthogonal zum Einsatz von Firewalls. Keiner der Mechanismen kann alle Sicherungsziele alleine durchsetzen.

der Kommunikation. Beispielsweise könnten Benutzer durch Unwissenheit oder Unaufmerksamkeit (Fahrlässigkeit) auf den Einsatz von kryptographischen Mechanismen verzichten, obwohl dies laut Unternehmensrichtlinie vorgeschrieben ist. Zumindest am Übergang zwischen den Sicherheitsdomänen kann ein Firewall jedoch die Verwendung oder auch Unterlassung von kryptographischen Mechanismen entsprechend einer Richtlinie durchsetzen. Die Integration eines 'virtuell private network'-Endpunktes in einen Firewall ist sicherlich das bekannteste Beispiel für das Durchsetzen von Verschlüsselungsrichtlinien im Firewall (vgl. [139]).

- Anonymisierung der Inter-Netzkommunikation: Die zunehmende Benutzung von Internet-Werkzeugen (beispielsweise WWW-Browser) geht einher mit "digitalen Spuren", die die Benutzer hiermit auf kontaktierten Servern hinterlassen. Manche Server verlangen u.U. sogar eine starke (kryptographisch sichere) Authentifikation vom Benutzer mittels Zertifikaten. Ein Firewall kann als zusätzlicher "Hop" zwischen Client und Server diese Authentifikation stellvertretend vornehmen und somit zu einer Teil-Anonymisierung beitragen. Dies kann beispielsweise durch ein Firmenzertifikat erfolgen, das der Firewall stellvertretend für die Benutzer an die Server weiterleitet. Auf dem Server kann durch das Zertifikat die Spur nur bis zur Firma, nicht jedoch bis zum individuellen Benutzer, zurückverfolgt werden.

Die oben genannten Beispiele (obwohl nicht vollständig) für Einsatzmöglichkeiten der Firewalls reichen aus, um die Bedeutung dieser gut verstandenen und in den letzten Jahren zunehmend eingesetzten Zugriffskontrollmechanismen zu verdeutlichen.

2.3.1.3 Bedeutung der Packet Screen für Firewall-Architekturen

Wie oben angedeutet besteht ein Firewall typischerweise aus mehreren Komponenten (Bausteinen), wobei fast immer mindestens eine Packet Screen dazugehört. In diesem Abschnitt wird erläutert, warum gerade die Packet Screen besonders wichtig für viele Firewall-Installationen ist. Dies ist letztendlich auch der Grund, eine Packet Screen in eine überlebensfähige Firewall-Komponente zu überführen.

Eine Packet Screen führt eine eher grobe Zugriffskontrolle durch, da die Regeln typischerweise Bezug auf die Attribute in den "Protocol Data Unit"-Köpfen (PDU-Köpfen) nehmen und den Inhalt (Benutzerdaten, "Service Data Unit", SDU) nicht weiter auswerten. Die Regeln werden üblicherweise nach einem der beiden folgenden Prinzipien konfiguriert (vgl. [73]):

- Nicht explizit verbotene Kommunikation ist erlaubt. Hierbei ist die Standard-Aktion somit das Weiterleiten der PDUs.

- Nicht explizit erlaubte Kommunikation ist verboten. Hierbei ist die Standard-Aktion somit das Verwerfen der PDUs. Dies ist eine Ausprägung des allgemeinen "least privileges"-Prinzips.

Wie bereits weiter oben angedeutet, lassen sich viele Firewall-Installationen auf eine Basis-Architektur zurückführen. Auffällig ist, daß bei den am häufigsten[10] zum Einsatz kommenden Firewall-Architekturen mindestens eine Packet Screen als Firewall-Komponente integriert ist. Hierbei ist besonders zu beachten, daß die Packet Screen in allen vorgestellten Firewall-Architekturen immer im Hauptdatenpfad zwischen den Netzen eingesetzt wird. Die Packet Screen muß somit alle Dateneinheiten verarbeiten, die den Firewall passieren sollen. Die Gründe hierfür werden bei der folgenden Diskussion dieser Architekturen verdeutlicht.

Die folgenden Firewall-Architekturen sind [24, 60] entnommen und somit konsistent mit den bisher eingeführten Begriffen. Es werden zu jeder Architektur auch die Verweise auf andere häufig verwendete Bezeichnungen vorgestellt. Es wird hier nur am Rande auf die jeweiligen Vor- und Nachteile der einzelnen Architekturen eingegangen. Eine detaillierte Diskussion dieser Aspekte erfolgt in der zitierten Literatur.

Packet Screen

Die einfachste Möglichkeit, zwei Netze durch einen Firewall zu trennen, ist der Einsatz einer Packet Screen. Die Architektur besteht somit nur aus dieser einen Komponente (vgl. Abbildung 2.1). Offensichtlich muß der gesamte Verkehr zwischen den beiden Netzen über die Packet Screen geleitet werden, da sie die einzige Verbindung zwischen diesen Netzen darstellt. In der Literatur wird diese Architektur häufig auch als “dual homed firewall” bezeichnet. Die Bezeichnung ist jedoch nicht eindeutig, da sie auch auf “gateway firewalls” zutrifft.

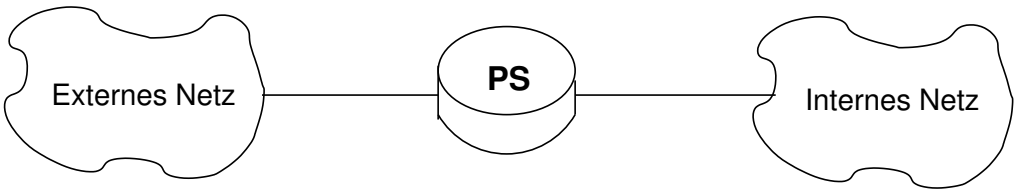


Abbildung 2.1: Packet Screen (PS)

[10] In den letzten drei bis fünf Jahren gibt es eine deutliche Zunahme der Anschlußzahlen von individuellen Rechnern oder auch kleinen Firmennetzen über relativ langsame Übertragungswege (beispielsweise ISDN). Auch diese kleinen ‘small office - home office’-Installationen werden zunehmend über Firewall-Installationen abgesichert, die meist eine Mischform der diskutierten Architekturen darstellen. Diese Mischformen ergeben sich durch die Anforderung der häufig völlig überforderten Benutzer, Sicherheit ohne Administrationsaufwand als ‘Black Box’ geliefert zu bekommen. Die Hersteller kommen dieser Nachfrage dann mit entsprechenden Standardlösungen nach, deren Zugriffskontroll-Mechanismen jedoch kaum auf Basis von individuellen Richtlinien der Benutzer erfolgen. Bei Unternehmen, die ernsthaft an Sicherheitslösungen auf Basis ihrer eigenen Richtlinien interessiert sind, kommen jedoch die hier beschriebenen Architekturen zum Einsatz.

Packet Screen mit Bastion

Die folgenden Architekturen kombinieren Packet Screens mit weiteren Firewall-Komponenten, um eine feingranulare Zugriffskontrolle auf der Bastion (vgl. Definition 2.8) zu ermöglichen. Bei allen Architekturen gilt:

- Die Packet Screen ist immer im Hauptdatenpfad zwischen den beiden Netzen. Alle Dateneinheiten, die von dem einen in das andere Netz gesendet werden, müssen die Packet Screen passieren. Es gibt keinen Weg an der Packet Screen vorbei.
- Die Packet Screen hat die Aufgabe, eine Vorauswahl zu treffen und Kommunikationsbeziehungen, die in jedem Fall im Widerspruch zu den Richtlinien stehen, unmittelbar zu blockieren.[11] Die verbleibenden, zu möglicherweise erlaubten Kommunikationsbeziehungen gehörenden Dateneinheiten werden der Bastion für eine detailliertere Zugriffskontrolle überlassen. Typischerweise verhindert die Packet Screen in allen Szenarien die Umgehung der Bastion, um eine feinere Zugriffskontrolle auf der Bastion zu erzwingen. Dies geschieht entweder durch Umleiten der noch nicht verworfenen Dateneinheiten an die Bastion oder durch Verwerfen von Dateneinheiten, die nicht an die Bastion gerichtet sind bzw. nicht von dieser gesendet wurden.

Die folgenden Architekturen werden in der Literatur unter dem Sammelbegriff "screened bastion firewall" behandelt. In Abbildung 2.2 ist eine Architektur mit externer Bastion dargestellt. "Extern" bezieht sich auf die Richtung der Kommunikation in Bezug auf die vom Firewall geschützte Sicherheitsdomäne. Die Packet Screen (PS) ist dichter an der zu schützenden Sicherheitsdomäne und schützt diese auch dann noch, falls die vorgelagerte Bastion in ihrer Funktion versagt. Abbildung 2.3 zeigt die gespiegelte Firewall-Architektur. In diesem Fall ist die Packet Screen "extern" und die Bastion dichter an der Sicherheitsdomäne. Bei dieser Firewall-Architektur wird die Bastion zusätzlich durch die Packet Screen gegen Angriffe von außerhalb der Sicherheitsdomäne geschützt. Kombiniert man die beiden oben gezeigten Firewall-Architekturen miteinander, so erhält man eine beidseitig abgeschirmte Bastion. Die Bastion wird so gegen Angriffe aus beiden Sicherheitsdomänen geschützt. Bei dieser Architektur erhält man ein weiteres Subnetz, das zwischen den Sicherheitsdomänen etabliert wird. Dieses in der Literatur als "perimeter network" oder auch als "demiliterized zone" (DMZ) bezeichnete Übergangsnetz ist eine dritte Sicherheitsdomäne. Die Richtlinien gestatten häufig Zugriffe auf die hier zusätzlich zur Bastion angesiedelten Server. Die DMZ kann man sich daher auch als "semi-öffentliches" Netz vorstellen. Da zwischen den beiden Packet Screens die DMZ als ein weiteres Netz eingefügt wird, das hauptsächlich dem Schutz des internen Netzes dient, wird es häufig auch als Schutznetz oder "Perimeternetz" bezeichnet. Bei allen bisher diskutierten Architekturen ist die Packet Screen somit essentieller Bestandteil. Darüber hinaus liegt mindestens eine Packet Screen immer im Hauptdatenpfad von Kommunikationsbeziehungen zwischen Teilnehmern aus verschiedenen angrenzenden Domänen.

[11] Zusätzlich zu jeder Aktion, die eine Packet Screen mit den Dateneinheiten durchführt, kann auch ein Audit durchgeführt werden.

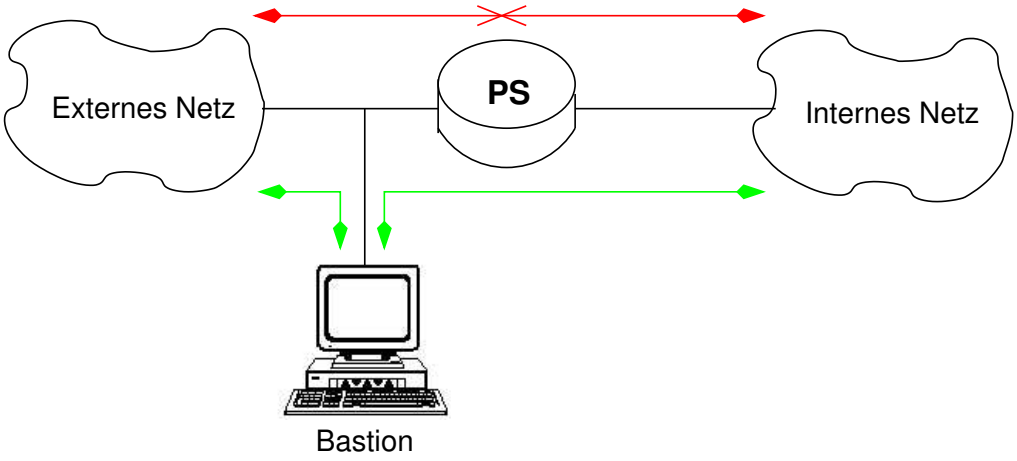


Abbildung 2.2: Packet Screen mit externer Bastion

Hieraus alleine lassen sich bereits hohe Leistungsanforderungen an die Paketverarbeitung auf der Packet Screen ableiten. Die Packet Screen muß jedoch insbesondere dann besonders leistungsfähig sein, wenn sie noch weitere Aufgaben übernimmt. Typischerweise ist eine Packet Screen in IP-basierten Netzen gleichzeitig ein Vermittlungsrechner und somit neben der Zugriffskontrolle und dem Audit häufig auch zuständig für:

- Vermittlung von Dateneinheiten. Die Wegwahlfunktionen erfordern häufig das zusätzliche Bearbeiten von Routing-Protokollen wie beispielsweise OSPF oder RIP.[12] Dies ist meist der Fall, wenn die Packet Screen als Subsystem eines Vermittlungsrechners realisiert ist. Die Bearbeitung von Wegwahl-Protokollen auf der Packet Screen ist aus Sicherheitsgründen zu vermeiden.

- Umsetzen von Netzadressen. Falls in einem angeschlossenen Netz private IP-Adressen verwendet werden, dann muß beim Übergang zwischen den Domänen auch eine Übersetzung der privaten IP-Adressen in öffentliche IP-Adressen erfolgen. Diese Funktion kann kombiniert werden mit Funktionen zur Lastverteilung. Die Packet Screen führt ein "load balancing" z.B. für Server in der DMZ durch (vgl. [71, 76]).

- Durch die Kenntnis der Netztopologie ist die Packet Screen (in ihrer Funktion als Vermittlungsrechner) auch besonders gut geeignet, als Endpunkt für die Kommunikation in einem "virtual private network" (VPN) zu fungieren, was das Ausführen aufwendiger kryptographischer Berechnungen auf allen Dateneinheiten nach sich zieht, die Teil einer VPN-Kommunikationsbeziehung sind (vgl. [24, 139]).

[12] Es handelt sich hierbei um zwei bekannte Routing-Protokolle. Das relativ leichtgewichtige "Routing Information Protocol" (RIP, vgl. [157]) und das aufwendigere, hierarchische Routing-Protokoll "Open Shortest Path First Interior Gateway Protocol" (OSPF, vgl. [175]).

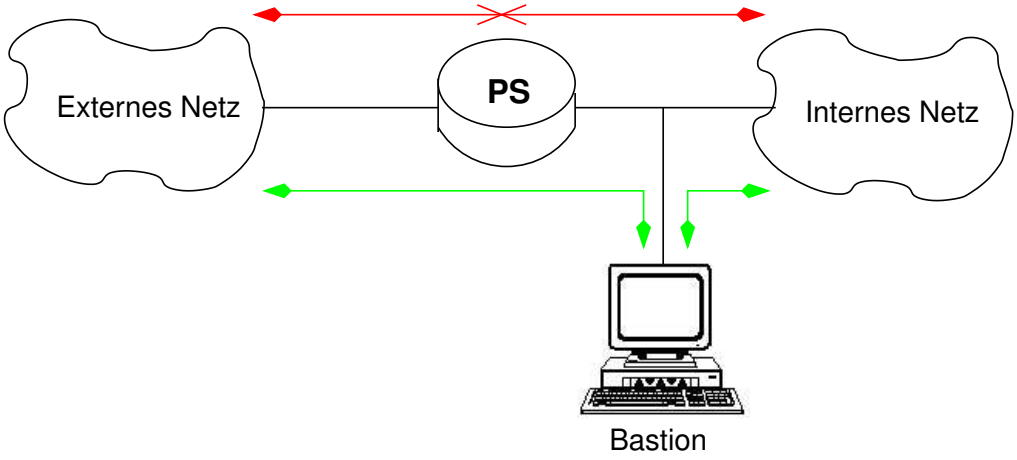


Abbildung 2.3: Packet Screen mit interner Bastion

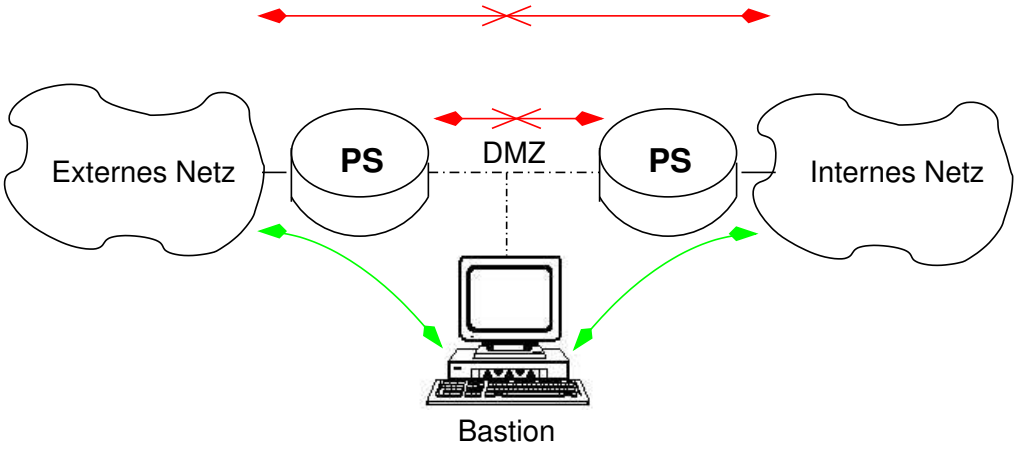


Abbildung 2.4: Beidseitig durch Packet Screens abgeschirmte Bastion

Für die weiteren Untersuchungen soll aufgrund der großen Bedeutung der Packet Screen für die Netzwerksicherheit am Beispiel der Packet Screen die Überführung in ein überlebensfähiges System studiert werden.

2.3.2 Notwendige Reduzierung der Komplexität

Nachdem die Packet Screen als System für eine Überführung in ein überlebensfähiges System ausgewählt wurde, müssen weitere Einschränkungen gemacht werden.

Um die Komplexität beim Entwurf von überlebensfähigen Systemen bewältigen zu können, ist nach Auffassung des Autors eine schrittweise Anpassung an die gleichzeitigen Anforderungen nach Verfügbarkeit, Leistungsfähigkeit und Sicherheit erforderlich. Hierbei sollte

vom primären Defizit ausgegangen werden. Nach jeder Anpassung, die aufgrund einer Bestandsaufnahme des jeweiligen Systemzustandes erfolgt (als Resultat der Evaluierung eines "usage scenarios" oder eines "intrusion scenarios"), muß geprüft werden, ob diese Anpassung nicht neue Sicherheitsrisiken, Verfügbarkeitsprobleme oder Leistungsengpässe mit sich bringt, bevor weitere Anpassungen erfolgen. Vor jeder weiteren Anpassung wird somit eine aktualisierte "Ist-Situation" festgestellt. So kann vermieden werden, daß umfangreiche Änderungen vorgenommen werden, die evtl. wieder rückgängig gemacht werden müssen. Gleichzeitig soll versucht werden, diese Evaluierung möglichst allgemein, losgelöst von speziellen Szenarien durchzuführen. Die Evaluierung muß zur Gewinnung von allgemeineren Aussagen auf die primären Anforderungen an überlebensfähige Systeme konzentriert werden: Es wird die Leistungsfähigkeit, die Sicherheit und die Verfügbarkeit evaluiert.

Um die Komplexität weiter zu reduzieren, sind an dieser Stelle folgende Annahmen zur "Ist-Situation" der Packet Screen erforderlich:

- Die Sicherheit der Packet Screen ist ausreichend. Es wird für die weitere Argumentation angenommen, daß die Sicherheit der konfigurierbaren Zugriffskontrolle ausreichend ist. Es gibt also keine Fehler in der Software für Zugriffskontrolle und Audit.
- Die Verfügbarkeit der Packet Screen ist ausreichend. Es wird angenommen, daß die Ausfallwahrscheinlichkeit der Packet Screen gering genug ist, so daß keine Anpassungen für die Erhöhung der Verfügbarkeit erforderlich sind.
- Es ist zu vermuten, daß die Packet Screen aufgrund der in Abschnitt 2.3.1 formulierten, hohen Leistungsanforderungen zum Engpaß bei der Kommunikation werden kann. Es muß demnach evaluiert werden, ob die Zugriffskontrolle auf der Packet Screen zum Engpaß bei der Kommunikation werden kann.

Ausgangspunkt für die weitere Betrachtung ist ein potentieller Leistungsengpaß auf der Packet Screen. Die erste Evaluierung muß daher aufzeigen, ob Packet Screens zum Engpaß bei der Kommunikation werden können.

2.4 Zusammenfassung

In diesem Abschnitt wurden an zwei Beispielen die Defizite der heute gängigen Entwurfspraxis bei der Erstellung von verteilten Systemen dargestellt. Es wurde argumentiert, daß die sogenannte Überlebensfähigkeit für viele Anwendungen ein übergeordnetes Entwurfsziel sein sollte. Bei der Suche nach generischen Entwurfszielen stellte sich heraus, daß besonders die vielfältigen Abhängigkeiten der Aufgabenerfüllung ("mission") des Systems von den drei wünschenswerten Eigenschaften *Leistungsfähigkeit, Sicherheit und Verfügbarkeit* zu beachten sind. Die Aufgabenerfüllung eines individuellen Systems ist zwar immer von den jeweiligen Anforderungen und Rahmenbedingungen abhängig, jedoch scheinen Leistungsfähigkeit, Sicherheit und Verfügbarkeit in vielen Fällen gleichzeitig erforderlich zu

sein. Anscheinend handelt es sich bei diesen Anforderungen um "generische Anforderungen" an überlebensfähige Systeme.

Es wurde anschließend aufgezeigt, daß es kaum Erfahrungen mit dem Entwurf dieser Systeme gibt. Besonders wichtig scheint die zielgerichtete Evaluierung und Anpassung der Systeme zu sein, wobei dieser Prozeß nicht einmal, sondern wiederholt durchlaufen werden muß, um neue Randbedingungen und Aufgaben berücksichtigen zu können. Die erneute Evaluierung eines Systems nach einer vorgenommenen Anpassung scheint auch deshalb erforderlich zu sein, weil die drei genannten "generischen Anforderungen" zu unterschiedlichen, möglicherweise gegenläufigen Optimierungen bzw. Anpassungen am System führen können.

Es ist daher erforderlich, an einem Beispiel die Anpassung eines Systems in ein überlebensfähiges System vorzunehmen, um zu zeigen, welche Probleme bei gegenläufigen Optimierungen auftreten können, und wie sich der "Tradeoff" zwischen verschiedenen erforderlichen, möglicherweise gleichwertigen, Anpassungen darstellt. Als Beispiel wurde die Packet Screen ausgewählt, da es sich hierbei um eine in vielen Fällen verwendete, sehr wichtige Firewall-Komponente handelt. Aufgrund ihrer zentralen Stellung bei der Koppelung verschiedener Sicherheitsdomänen werden an Packet Screens hohe Anforderungen an Verfügbarkeit, Leistungsfähigkeit und Sicherheit gestellt. Sie scheint daher besonders gut für eine Überführung in ein überlebensfähiges System geeignet zu sein. Da die Packet Screen in vielen verschiedenen Szenarien bei der Absicherung von Netzen eingesetzt wird, besteht die Aussicht, daß die erarbeiteten Ergebnisse auf viele Szenarien übertragen werden können.

Da es aufgrund mangelnder Erfahrung für den Entwurf von überlebensfähigen Systemen keine verbindlichen, allgemein akzeptierten Empfehlungen gibt, soll hier eine mehrstufige Vorgehensweise konsequent getestet werden. Durch die sukzessive Analyse und Verbesserung eines nicht-überlebensfähigen Systems soll schließlich eine Migration zu einem überlebensfähigen System erreicht werden. Nach jeder Iteration muß geklärt werden, welche Probleme beseitigt wurden und welche (neuen) Probleme im folgenden Iterationsschritt zu lösen sind. Zur Reduzierung der Komplexität bei dieser mehrstufigen Vorgehensweise wurde als Ausgangspunkt die Annahme formuliert, daß die Packet Screen als ausreichend sicher und verfügbar eingestuft werden kann. Lediglich die Leistungsfähigkeit ist möglicherweise unzureichend und soll daher in einer ersten Evaluierung untersucht werden.

Kapitel 3

Engpässe durch Packet Screens in Hochgeschwindigkeitsnetzen

Im vorigen Kapitel wurde motiviert, daß die Überführung von wichtigen Netzkomponenten (und somit auch von Firewall-Komponenten) zu überlebensfähigen Subsystemen eine vordringliche Aufgabe ist, da es sich hierbei um wichtige Infrastrukturkomponenten für die gesamte EDV handelt, die in der Regel nicht redundant ausgelegt sind. Es wurde auch gezeigt, daß die Packet Screen (PS) als ein Firewall-Baustein, über den der gesamte Netzverkehr (zwischen Sicherheitsdomänen) geleitet wird, besonders leistungsfähig sein muß. Dies ist insbesondere dann erforderlich, wenn weitere Aufgaben – wie beispielsweise "Routing" – zusätzlich von der Packet Screen übernommen werden.

Für andere Firewall-Komponenten, die aufwendigere Zugriffskontrollen als die Packet Screen durchführen, wurde bereits gezeigt, daß sie zum Engpaß bei der Kommunikation zwischen verteilten Anwendungen werden können (vgl. [77, 78, 24, 76, 71]), und es wurde darauf hingewiesen, daß mit zunehmenden Übertragungsgeschwindigkeiten auch die Packet Screen zu einem Engpaß werden kann. Da Packet Screens zu den am besten verstandenen und am häufigsten eingesetzten Firewall-Bausteinen gehören, besteht ein großes Interesse daran, PS auch in Hochgeschwindigkeitsnetzen einzusetzen. Wichtig ist, daß die Benutzer von der hohen Übertragungsgeschwindigkeit der neuen Netze profitieren können, d.h. die hohen Übertragungsgeschwindigkeiten auf der Netzwerkebene müssen im Idealfall auch auf der Anwendungsebene in den Endsystemen zur Verfügung stehen. Das Ziel muß demnach eine Absicherung sein, die leistungsfähig genug ist, um trotz der erforderlichen Zugriffskontrolle nicht zum Engpaß bei der Kommunikation zwischen den verteilten Anwendungen zu werden. Dies ergibt sich auch unmittelbar aus den generischen Anforderungen an überlebensfähige Systeme, die sowohl sicher als auch leistungsfähig sein sollen.

Eine Sicherheitskomponente, die durch die ständig steigenden Übertragungsgeschwindigkeiten in den Hochgeschwindigkeitsnetzen zum Engpaß bei der Kommunikation wird, behindert die normalen Arbeitsabläufe zusätzlich. Spätestens wenn die ersatzlose Streichung der Sicherheitskomponente zugunsten einer schnelleren Kommunikation ansteht, sollten an-

dere Lösungsmöglichkeiten gesucht werden. Bevor Lösungsmöglichkeiten aufgezeigt werden können, müssen zwei wichtige Unterziele erreicht werden:

1. Nachweisen von Engpässen: Es muß gezeigt werden, daß durch die Zugriffskontrolle auf der Packet Screen tatsächlich Engpässe bei der Kommunikation zwischen verteilten Anwendungen entstehen können.

2. Aufzeigen der Ursachen: Es muß analysiert werden, warum und unter welchen Umständen eine Packet Screen zum Engpaß wird. Das Verstehen der Ursache ermöglicht die Suche nach Lösungen für die Beseitigung der Engpässe.

Für den Nachweis von Engpässen ist eine Leistungsanalyse erforderlich. In Abschnitt 3.1 werden verschiedene Alternativen zur Durchführung einer Leistungsanalyse vorgestellt. Es wird begründet, daß vergleichende Messungen die beste Alternative darstellen und daher für die Engpaßanalyse herangezogen werden sollen.

In Abschnitt 3.2 wird dargestellt, welche Szenarien bei der Analyse von Netzkomponenten betrachtet werden können, und welche Kenngrößen für eine Beurteilung der Leistungsfähigkeit einer Packet Screen ermittelt werden müssen.

Aus den zu ermittelnden Kenngrößen und den Szenarien ergeben sich wiederum Anforderungen an Meßwerkzeuge, mit denen die vergleichenden Messungen durchgeführt werden. In Abschnitt 3.3 werden diese Anforderungen dargestellt und es wird gezeigt, wie Meßwerkzeuge erstellt bzw. erweitert werden können, um diesen Anforderungen zu genügen.

Mit diesen Werkzeugen können schließlich Messungen durchgeführt werden, um die Leistungskenngrößen in Abhängigkeit vom Szenario zu ermitteln. Der Vergleich der Ergebnisse eines Szenarios ohne Zugriffskontrolle durch eine Packet Screen mit einem Szenario, bei dem eine Zugriffskontrolle durch die Packet Screen stattfindet, kann dann Aufschluß über Engpässe durch die Packet Screen geben. Diese Analyse wird in Abschnitt 3.6 durchgeführt.

3.1 Leistungsbewertung von Rechen- und Kommunikationssystemen

Konzeptionell ist die Leistungsbewertung von Rechnern, Rechnernetzen und verteilten Systemen seit langem gut verstanden. Soll die Leistungsfähigkeit eines Systems bestimmt werden, gibt es grundsätzlich zwei verschiedene Möglichkeiten: Ein vorhandenes System kann mit Hilfe von *Meßmethoden* direkt evaluiert werden oder es kann eine *modellbasierte Bewertung* des Systems durchgeführt werden. Bei der modellbasierten Bewertung wird noch einmal zwischen der analytischen und der simulativen Bewertung der Systemmodelle unterschieden.

Diese Alternativen haben jeweils Vor- und Nachteile, die hier kurz zusammengefaßt werden (ausführlicher beispielsweise in [129, 104, 109]):

Messungen haben den entscheidenden Vorteil, ein vorhandenes System direkt "auf Herz und Nieren" testen zu können. Darüber hinaus ermöglichen die gewonnenen Leistungsdaten dieses Tests einen direkten Vergleich mit den Leistungsdaten aus gleichartigen Tests und somit den Vergleich verschiedener Systeme. Der Hauptnachteil dieser Methode liegt darin, daß das zu untersuchende System bereits existieren muß und die benötigten Schnittstellen für die Meßdatenerfassung zugänglich sein müssen (s.u.).

Simulative Bewertung ist besonders geeignet zur Optimierung von Systemen durch ein verfeinertes Design. Ist ein Simulations-Modell von einem System erstellt und validiert[1], dann können die Auswirkungen verschiedenartiger Variationen der Modellparameter aufwandsarm studiert werden. Das Simulations-Modell ermöglicht somit eine Optimierung durch Variieren und Ausprobieren.

Analytische Bewertung ist besonders gut geeignet, um Prognosen zum Beispiel im Rahmen einer Kapazitätsplanung zu *berechnen*. Das analytische Modell liegt als eine oder mehrere Formeln vor, in die durch Ersetzen der Modellparameter durch numerische Werte die Auswirkungen direkt berechnet werden können. Die Ergebnisse sind in der Regel sehr viel schneller zu berechnen als durch simulative Bewertung. Der Hauptnachteil der analytischen Modelle liegt in der häufig sehr aufwendigen und mathematisch anspruchsvollen Modellbildung. Dies führt in der Praxis dazu, daß meist nur sehr einfache analytische Modelle erstellt werden können.

Da das Ziel der Nachweis von Engpässen ist, müssen empirische Meßdaten zur Leistungsfähigkeit von Packet Screens erhoben werden. Es ist daher vorerst lediglich möglich, Messungen an vorhandenen Packet Screens durchzuführen. Auf Grundlage dieser Meßergebnisse können dann weiterführende Bewertungen durchgeführt werden. Beispielsweise könnten die Meßergebnisse für die Kalibrierung eines analytischen oder simulativen Modells verwendet werden.

3.2 Szenarien, Last-, Konfigurations- und Leistungskenngrößen für die Bewertung von Packet Screens

Zum Aufzeigen von Engpässen ist es erforderlich, die noch festzulegenden Leistungskenngrößen für mindestens zwei Szenarien zu ermitteln. Im ersten Szenario werden die Kenngrößen für die direkte Kommunikation zwischen verteilten Anwendungen erhoben. Im zweiten Szenario werden die Kenngrößen bei indirekter Kommunikation, über eine zwischen den beiden Kommunikationspartnern eingefügte Packet Screen, erhoben. Ein Engpaß ist genau

[1] Die Modellerstellung sollte validiert werden. Erste Ergebnisse einer Simulation können beispielsweise mit Messungen am Originalsystem verglichen werden oder mit den Ergebnissen eines analytischen Modells. Durch derartige Vergleiche kann man die Adäquatheit der Modellierung belegen.

dann gefunden bzw. nachgewiesen, wenn die gemessene Leistung bei der Kommunikation durch den Einsatz der Packet Screen deutlich schlechter ist als bei der direkten Kommunikation.

Ist ein Engpaß gefunden, kann darüber hinaus untersucht werden, von welchen Faktoren die Leistungsfähigkeit der Packet Screen abhängt. Für die Beseitigung eines Engpasses, der von mehreren Faktoren abhängt, sollte erst die maßgebliche (dominante) Ursache behandelt werden. Um jedoch die dominanten Faktoren bestimmen zu können, sind weitere Messungen durchzuführen, bei denen die Auswirkungen verschiedener Last- und Konfigurationskenngrößen aufgezeigt werden.

Der angestrebte Vergleich der Systeme durch Beurteilen der mit gleichartigen Messungen an verschiedenen Systemen gewonnenen Leistungskenngrößen wird auch als "Benchmarking" bezeichnet.

Ein Vergleich ist jedoch nur möglich, wenn die bei verschiedenen Messungen benutzten Lastprofile (gekennzeichnet durch die Aufträge/Auftragsklassen und deren Attribute) identisch oder zumindest annähernd identisch sind. Für das Benchmarking sind daher Messungen in realen Umgebungen, zum Beispiel Messungen an Netzkomponenten in Produktionsnetzen, nicht geeignet, da die reale Last hohen Schwankungen unterliegt. Systematische Messungen für ein Benchmarking sind nur in speziellen Testumgebungen mit Lastgeneratoren möglich, die in der Lage sind, eine *synthetische Last* präzise zu reproduzieren. Die genaue Reproduktion dieser synthetischen Last/Lastprofile ist die eigentliche Schwierigkeit bei systematischen Messungen an Kommunikationssystemen und in der Praxis ein hartnäckiges Problem.

Die Lastgeneratoren und Meßwerkzeuge für die Systemleistung werden häufig als Programme erstellt, die auf dem zu untersuchenden System ausgeführt werden. Typische Vertreter dieser Art von Meßwerkzeugen sind die von der "Standard Performance Evaluation Corporation" (SPEC) herausgegebenen Lastgeneratoren (vgl. `http://www.spec.org`). In diesem Fall übernimmt ein Programm sowohl die Generierung der Last als auch die Beobachtung/Aufzeichnung der erreichten Leistung. In vielen Fällen ist diese Integration jedoch nicht möglich. Ein allgemeineres Modell für die systematische Meßdatenerfassung muß in Anlehnung an [104] und [37, §6] einen Generator logisch von einem Monitor unterscheiden. Generator und Monitor kooperieren bei der Durchführung der Leistungsanalyse (vgl. Abbildung 3.1). Der Generator übergibt dem zu untersuchenden System (System unter Last, SuL) an einer Schnittstelle Aufträge, die festzulegenden Lastkenngrößen genügen. Das System verarbeitet die Aufträge und wird dabei vom Monitor beobachtet, der an einer weiteren Schnittstelle (systemabhängige) Leistungskenngrößen aufzeichnet. Diese Art von Lastmessung ermöglicht somit auch die vergleichende Beurteilung von verteilten Systemen. Das in Abbildung 3.1 vorgestellte Modell verallgemeinert darüber hinaus die in [146, S.37ff] diskutierten Meßmethoden für verteilte Systeme, bei denen Lastgenerator und Monitor nicht voneinander unterschieden werden und somit die durchführbaren Untersuchungen auf das Client-Server-Modell beschränkt sind.

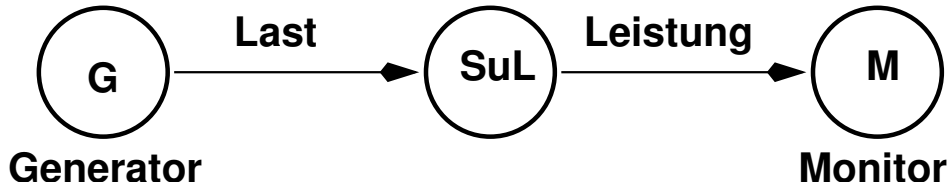


Abbildung 3.1: Allgemeines Modell für die systematische Meßdatenerfassung

3.2.1 Szenarien für die Leistungsbewertung von Packet Screens

Die Leistungsbewertungen der Packet Screen-Architekturen werden auf Basis von Fast Ethernet durchgeführt, da diese Netztechnologie weit verbreitet ist. In den drei folgenden Unterabschnitten wird der allgemeine Versuchsaufbau für die Messungen dargestellt, sowie die jeweilige Verfeinerung für "shared medium" und "switched" Fast Ethernet-Netze. Die verschiedenen Versuchsaufbauten entsprechen den möglichen Einsatzszenarien, die sich bei der Wahl von Fast Ethernet als Netzwerkprotokoll und der Packet Screen als "SuL" nach Abbildung 3.1 ergeben.

3.2.1.1 Allgemeiner Versuchsaufbau

Der allgemeine Versuchsaufbau ergibt sich aus *Lastgeneratoren*, die entsprechend einem Untersuchungsziel *Aufträge* generieren und an *Monitore* senden. Zwischen diesen lastgenerierenden und empfangenden Systemen können keine, ein oder mehrere *zu vermessende Systeme* bzw. *Systeme unter Last, SuL* angeordnet werden. Die Generatoren stimulieren die Auftragsbearbeitung auf dem SuL durch die Aufträge. Das SuL bearbeitet die Aufträge und die dabei erreichte Leistung wird von den Monitoren erfaßt.

Zur Verdeutlichung der in den folgenden Abschnitten verwendeten Begriffe dienen die Abbildungen 3.2 — 3.6.

Abbildung 3.2 zeigt den schematischen Versuchsaufbau einer direkten Messung zwischen Lastgenerator (G) und Monitor (M). Meßreihen unter diesem Szenario dienen als *empirische Vergleichswerte* für weitere Messungen, bei denen eine Packet Screen-Architektur als SuL zwischen Generator und Monitor eingefügt wird.

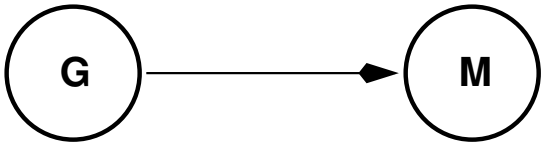


Abbildung 3.2: Versuchsaufbau: "Generator zu Monitor"-Messung

Abbildung 3.3 zeigt den schematischen Versuchsaufbau einer Messung, bei der ein zu vermessendes System (SuL) zwischen Generator und Monitor angeordnet ist. Mit diesem Versuchsaufbau können Aussagen über das zu vermessende System (SuL) im Vergleich zu Messungen nach Abbildung 3.2 gemacht werden. Da sich die folgenden Untersuchungen immer auf Protokolle der TCP/IP-Familie beziehen, könnte man den Versuchsaufbau mit dem zusätzlichen "Hop" über das zu vermessende System auch als "Host to Router to Host"-Messung (HRH) bezeichnen. Dies ergibt sich aus der Notwendigkeit, das zu vermessende System innerhalb der TCP/IP-Familie als Vermittlungsrechner (Router) zu konfigurieren. Es gibt zwar andere Möglichkeiten, Daten vom Lastgenerator zum Monitor über ein drittes System weiterzuleiten, die Verwendung eines Routers ist jedoch der "natürliche" Ansatz bei TCP/IP.



Abbildung 3.3: Versuchsaufbau: "Lastgenerator zu SuL zu Monitor"-Messung

Da es das Ziel ist, mit den Experimenten auch die Leistungsfähigkeit von parallelen Lösungen zu bewerten, ergibt sich ein weiterer allgemeiner Versuchsaufbau wie in Abbildung 3.4 dargestellt. Der Generator sendet an mehrere (n) parallele Instanzen ($SuL_0, \cdots, SuL_{n-1}$) eines zu vermessenden Gesamtsystems (SuL). In diesem Fall können Aussagen über das Gesamtsystem gemacht werden. Die einzelnen Instanzen sind wiederum als Router zu verstehen. Jede einzelne Instanz leitet Dateneinheiten vom Generator an den Monitor weiter. Dieser Versuchsaufbau könnte somit auch als "Host to Parallel Router to Host" (HparH) bezeichnet werden. In [158] werden für die Unterscheidung zwischen Subsystemen ($SuL_0, \ldots, SuL_{n-1}$) und Gesamtsystem (SuL) verschiedene Begriffe verwendet. Letzteres wird als "System under Test" bezeichnet, während eine Komponente als "Device under Test" bezeichnet wird.

Die bisher vorgestellten Versuchsaufbauten dienen der Erhebung von Meßdaten, bei denen von einem Generator unidirektional an einen Monitor gesendet wird. In der Praxis kommen aber auch bidirektionale Kommunikationsbeziehungen vor. Für die Bewertung von Packet Screens ist es daher erforderlich, weitere Versuchsaufbauten bereitzustellen, mit denen auch bidirektionale Messungen durchgeführt werden können.

Es hat sich bei der Durchführung von verschiedenen Messungen gezeigt, daß für derartige Versuche unabhängige Lastgeneratoren und Monitore eingesetzt werden sollten. Generator und Monitor werden auf beiden "Seiten" des zu vermessenden Systems repliziert (vgl. Abbildung 3.5).

Der Vorteil dieses Vorgehens gegenüber einer Verschmelzung der Sende- und Empfangsfunktionen in einem kombinierten Generator/Monitor liegt in der Vergleichbarkeit der Ergebnisse mit denen der unidirektionalen Versuchsaufbauten. Ein unabhängiger Monitor

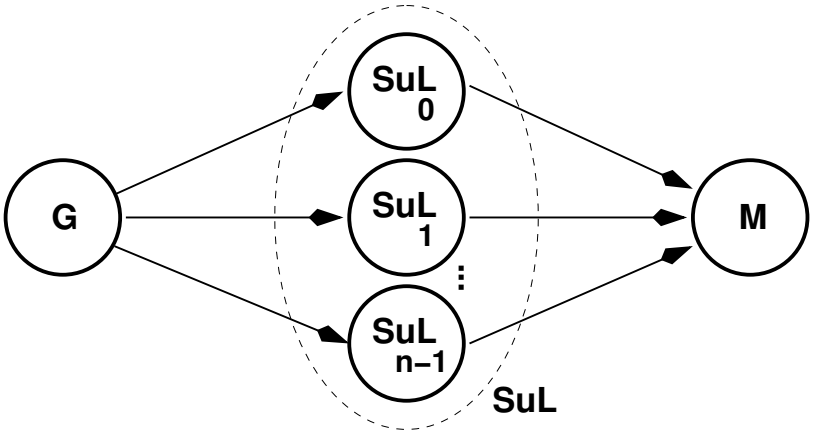


Abbildung 3.4: Versuchsaufbau: "Lastgenerator zu parallelem SuL zu Monitor"-Messung

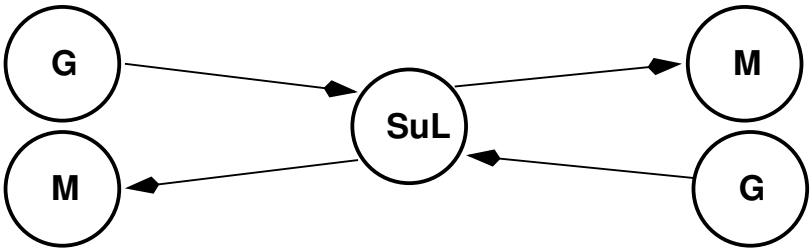


Abbildung 3.5: Versuchsaufbau: bidirektionale Messung, unabhängige Meßwerkzeuge

verhält sich genauso wie in den unidirektionalen Versuchsaufbauten aus Abbildung 3.2—3.4. Eine Verdoppelung der Funktionen (gleichzeitiges Senden und Empfangen) in derselben Instanz (vgl. Abbildung 3.6) führt zu einem anderen, nicht vergleichbaren Verhalten der Meßwerkzeuge. Für die Vergleichbarkeit der Messungen untereinander ist es daher

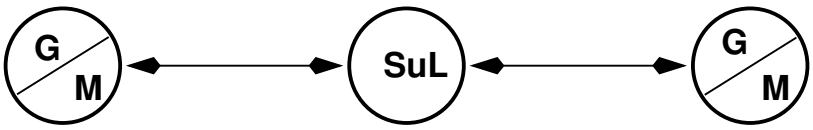


Abbildung 3.6: Versuchsaufbau: bidirektionale Messung, Abhängigkeiten zwischen Lastgenerator und Monitor

erforderlich, unabhängige Lastgeneratoren und Monitore für die bidirektionalen Messungen einzusetzen. Diese werden mit den in Abschnitt 3.3.3 diskutierten Mechanismen synchronisiert.

Einen entsprechenden Versuchsaufbau gibt es auch für bidirektionale Messungen an parallelen Instanzen eines zu vermessenden Systems.

Neben Meßdaten/Meßwerten, die auf Grundlage eines der dargestellten Versuchsaufbauten erhoben werden, gibt es für einzelne Leistungskenngrößen auch ableitbare Werte. In vielen Graphiken mit Meßergebnissen sind zusätzlich auch abgeleitete Werte eingezeichnet. Dies ist in jedem Fall in der Legende zu den Kurvenverläufen vermerkt.

3.2.1.2 Versuchsaufbau für Fast Ethernet-Messungen

Fast Ethernet (FE) ist eine Weiterentwicklung des bekannten Ethernet-Protokolls (IEEE 802.3). Die Zugriffskontrolle erfolgt weiterhin nach dem CSMA/CD-Verfahren ("Carrier Sense Multiple Access with Collision Detection"). FE ist daher auch als "Shared-Medium"-Netz zu verwenden, in dem alle sendewilligen Stationen um das Übertragungsmedium konkurrieren. Das gemeinsame Medium wird bei dieser Betriebsart des FE-Netzes durch Verstärker ("Repeater") mit mehreren Schnittstellen für den Anschluß der Stationen realisiert. Diese "Multiport-Repeater" (Hubs) bilden eine "Broadcast"-Domäne. Alle eintreffenden Signale werden auf alle anderen Schnittstellen weitergeleitet. Der zum Modell in Abbildung 3.3 analoge Aufbau für dieses FE-Szenario ist in Abbildung 3.7 graphisch dargestellt. Die beiden Meßsysteme sind jeweils an einen Hub angeschlossen. Die Hubs sind wiederum indirekt über die Packet Screen miteinander gekoppelt.

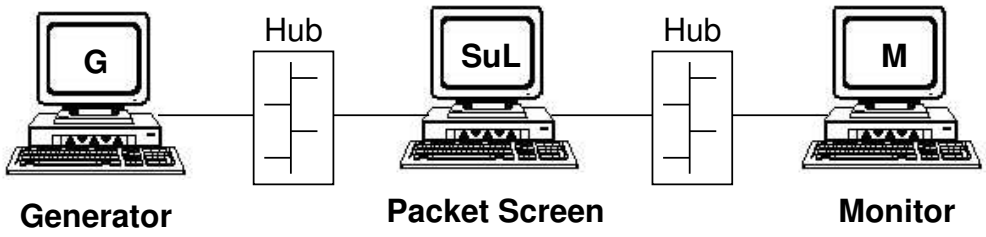


Abbildung 3.7: Physikalischer Versuchsaufbau für FE-Messungen (Hub)

Auch bei FE setzt sich zunehmend die Switching-Technologie durch. Ein FE-Switch bietet gegenüber einem FE-Hub den Vorteil, jedes Ethernet-Segment an einer Schnittstelle physikalisch von den anderen Schnittstellen zu trennen. FE-Switches bilden im Gegensatz zu Hubs somit kein "Shared-Medium" nach. Durch die Trennung der Schnittstellen ist es möglich, die eintreffenden Daten selektiv weiterzuleiten und nur noch auf den Ausgang zu kopieren, an den der Empfänger der Dateneinheit angeschlossen ist. Dieses Vorgehen führt zu einer dramatischen Verringerung der Kollisionen und somit zu einer Durchsatzsteigerung. Die an einem Switch angeschlossenen Stationen können paarweise miteinander mit der maximalen Übertragungsgeschwindigkeit kommunizieren. Jede Schnittstelle des Switches stellt die gesamte Bandbreite von 100 Mbit/s zur Verfügung. Durch diese Trennung in verschiedene Netzsegmente ist es außerdem möglich, die Sende- und Empfangsleitungen bei "Unshielded Twisted Pair" (UTP)-Kabeln zu trennen. Über eine Switch-Schnittstelle (Switch-IF) kann dadurch ein akkumulierter Durchsatz von 200 Mbit/s erreicht werden.

Der Nachteil der Switching-Technologie bei FE liegt darin, daß durch die eingesetzte "Store and Forward"-Technik die Verzögerung eines Ethernet-Rahmens bei der Übertragung im Switch proportional zur Länge des Rahmens ist.

Der Versuchsaufbau für viele der durchgeführten Messungen in einem "geswitchten" FE-Netz ist analog zu dem Versuchaufbau mit FE-Hubs aus Abbildung 3.7. In der folgenden Abbildung (vgl. Abbildung 3.8) ist daher der Versuchsaufbau für Messungen entsprechend dem Modell aus Abbildung 3.5 dargestellt. Es ist außerdem angedeutet, daß die Synchronisation der Meßwerkzeuge über einen dritten Rechner ("Sync") und ein separates Konfigurationsnetz (gestrichelte Verbindungen) erfolgt. Man erkennt deutlich die in der Praxis erforderlichen Erweiterungen gegenüber dem theoretischen Modell.

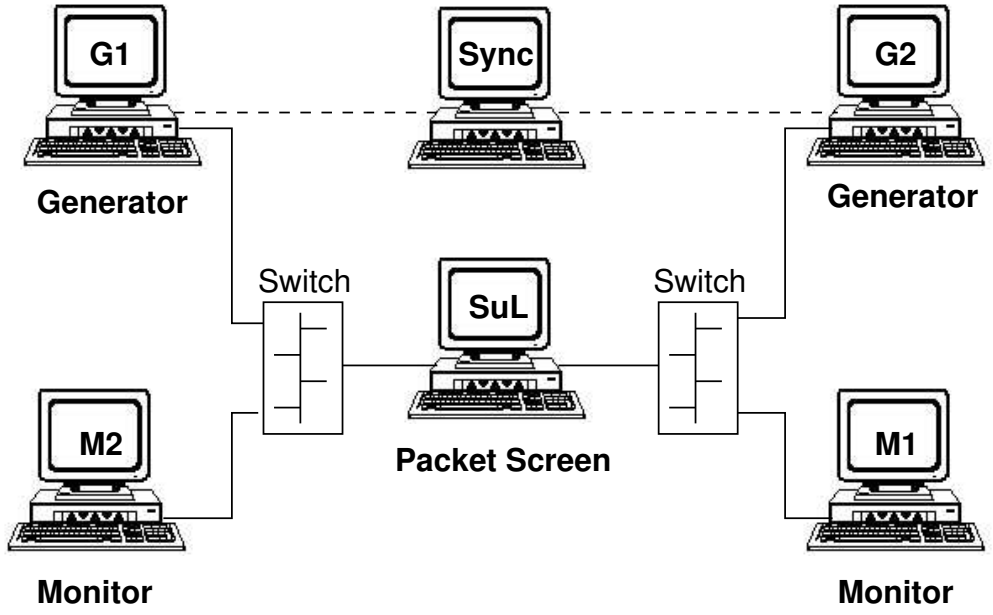


Abbildung 3.8: Physikalischer Versuchsaufbau für FE-Messungen (Switch)

3.2.2 Last-, Konfigurations- und Leistungskenngrößen für die Bewertung von Packet Screens

Das globale Untersuchungsziel ist im folgenden die Bewertung der Leistungsfähigkeit einer Packet Screen. Es sind im allgemeinen Fall viele Kenngrößen denkbar, die im Zuge einer Evaluierung eines Systems betrachtet werden können. Bei einer vergleichenden Bewertung auf Grundlage von Messungen geht es aber immer darum, solche Größen zu untersuchen, die quantitativ bewertbar sind, um sie anschließend in Beziehung zu den Ergebnissen auf anderen Systemen setzen zu können. Es werden hier somit alle Bewertungen bezüglich der

Funktionalität und der Qualität ausgeklammert.[2] Darüber hinaus werden Untersuchungen zur Zuverlässigkeit bzw. Verfügbarkeit erst in späteren Abschnitten durchgeführt, so daß hier im wesentlichen die Durchsatzkenngrößen und damit zusammenhängende Kenngrößen, wie beispielsweise Verzögerungszeiten, übrigbleiben.

Im allgemeinen Fall bezieht sich der erreichbare Durchsatz immer auf eine bestimmte Art von Aufträgen (formal einer Auftragsklasse a), die in einer bestimmten Zeit bearbeitet werden. Der Durchsatz (d) gibt dann die Anzahl der vom System bearbeiteten Aufträge pro Zeiteinheit an. Hierbei ist insbesondere der mittlere Durchsatz interessant, der in vielen Fällen ausreicht, um von den in der Praxis häufig vorkommenden kleinen Schwankungen in der Verarbeitungszeit für einzelne Aufträge zu abstrahieren:

$$
\begin{aligned}
\Delta t, &\qquad \text{betrachtetes Zeitintervall} \\
\Delta d(t, \Delta t), &\qquad \text{bearbeitete Aufträge im Intervall} \quad (t - \Delta t, t] \\
D_a(t, \Delta t) = \frac{\Delta d_a(t, \Delta t)}{\Delta t} &\qquad \text{mittlerer Durchsatz zum Zeitpunkt} \quad t \\
&\qquad\qquad\qquad \text{an Aufträgen der Klasse} \quad a
\end{aligned}
\tag{3.1}
$$

Bei der Bewertung von Paket Screens stellt sich nun die Frage, welche Durchsatzkenngrößen am Besten die Leistungsfähigkeit beschreiben und welche Auftragsklassen ein Lastgenerator hierfür generieren muß. Da es sich bei der Packet Screen um eine Netzkomponente handelt, die als Nadelöhr den gesamten Verkehr zwischen zwei oder mehreren Netzen kontrollieren muß, kommen grundsätzlich alle Durchsatzkenngrößen in Frage, die quantitative Aussagen zum Kommunikationsaufkommen über die Packet Screen erlauben:

- Durchsatz in Megabit pro Sekunde (Mbit/s). Es könnte in Abhängigkeit vom jeweiligen Netztyp untersucht werden, wieviele Mbit/s an Nutzdaten über die Packet Screen gesendet werden können. Intuitiv wäre eine Packet Screen um so leistungsfähiger, je größer dieser Durchsatz ist.

- Pakete pro Sekunde. Ähnlich dem Durchsatz in Mbit/s könnte angegeben werden, wieviele Pakete pro Sekunde von der Packet Screen verarbeitet werden können. Diese Leistungskenngröße berücksichtigt die Tatsache, daß bei allen verbreiteten Protokollen die Informationen strukturiert –in Form von "Protocol Data Units (PDUs)"– übertragen werden. Außerdem läßt sich hieraus direkt eine Lastkenngröße für den Lastgenerator ableiten: Der Lastgenerator muß eine Sequenz von Dateneinheiten generieren, die sich wiederum in Klassen bezüglich ihrer Länge und ihrer sonstigen Attribute einteilen lassen.

[2] Auch in diesem Fall muß davon ausgegangen werden, daß die ursprüngliche Packet Screen sowohl den funktionalen Anforderungen als auch den qualitativen Anforderungen genügt. Andernfalls hätte vor einer Überführung in ein überlebensfähiges System eine den funktionalen und qualitativen Anforderungen genügende Zugriffskontrolle ausgewählt werden müssen (vgl. Abschnitt 2.3.2).

- Verbindungen pro Sekunde. Die meisten im WAN-Bereich eingesetzten Protokolle arbeiten verbindungsorientiert. Eine Packet Screen, die typischerweise auch LAN-zu WAN-Verkehr filtern muß, könnte daraufhin untersucht werden, wieviele Verbindungen pro Sekunde durch sie hindurch vermittelt werden können. Auch hier wäre eine mögliche Lastkenngröße unmittelbar für einen Generator ableitbar. In diesem Fall müßte der Generator die für den Verbindungsaufbau und -abbau erforderlichen Dienstelemente generieren können. Falls der Lastgenerator aus Gründen der Effizienz die Dienstschnittstelle des lokalen Systems umgeht, müßte er die Protokolldateneinheiten direkt generieren.

- Transaktionen pro Sekunde. Typischerweise sendet ein Kommunikationspartner ein oder mehrere Anfragen und erhält vom anderen Kommunikationspartner eine oder mehrere Antworten. Sämtliche für einen solchen Frage-Antwort-Ablauf versendeten Dateneinheiten zwischen zwei kommunizierenden Instanzen könnte man auch als eine zusammengehörende "Transaktion" auffassen. Eine mögliche Leistungskenngröße wäre somit auch der erreichbare Transaktionsdurchsatz, der nicht mit dem Transaktionsdurchsatz aus dem Bereich der Datenbanksysteme verwechselt werden darf. Das bekannteste Beispiel für diese Art von Transaktionen ist wahrscheinlich die auf dem HTTP-Protokoll [27, 87] basierende Anfrage eines WWW-Browsers mit der dazugehörenden Antwort eines WWW-Servers. Die Gewinnung dieser Leistungskenngröße stellt erhebliche Anforderungen an Lastgenerator und Monitor, eine detaillierte Beschreibung für den Entwurf und die Implementierung für derartige Meßwerkzeuge wird beispielsweise in [26] gegeben.

- Verzögerungszeiten und Latenzzeiten. Der Generator könnte neben den mittleren Verzögerungszeiten, die häufig aus den obigen Durchsatzkenngrößen abgeleitet werden können, auch die Verzögerungszeiten für einen Verbindungsauf/abbau oder die Latenzzeit zwischen Empfang einer Dateneinheit durch die Packet Screen und Weiterleiten dieser Dateneinheit an den Monitor nach einer Ruhephase messen. Daraus lassen sich Aussagen über die Unterschiede in der Auftragsbearbeitung während der Anlaufphase einer Packet Screen und dem eingeschwungenen (stationären) Zustand ableiten.

Die richtige Wahl der geeigneten Leistungskenngrößen ergibt sich teilweise durch spezielle Untersuchungsziele. Im einfachsten Fall sollte der Durchsatz an Paketen pro Sekunde auf einer Packet Screen gemessen werden. Hierfür gibt es mehrere Gründe:

1. Jede Packet Screen kennt als kleinste zu untersuchende Einheiten ein oder mehrere PDU-Typen. Nur eine Untermenge von Packet Screens ist darüber hinaus in der Lage, verbindungsorientiert zu filtern. Die Untersuchung zu Verbindungen pro Sekunde sind daher nicht in jedem Fall möglich.

2. Wesentlichen Einfluß auf die Leistungsfähigkeit von Packet Screens haben die noch zu diskutierenden Konfigurationskenngrößen (s.u.). Die wichtigste Konfigurationskenn-

größe einer Packet Screen ist die Anzahl von konfigurierten Filterregeln. Diese Filterregeln beziehen sich wiederum in fast allen Fällen auf Attribute von PDUs [162]. Der Zusammenhang zwischen der Anzahl der Filterregeln und dem erreichbaren Durchsatz läßt sich daher am Besten durch die Leistungskenngröße "Pakete pro Sekunde" beschreiben.

3. Eine Packet Screen ist in Anlehnung an [36, 158] ein "Network Interconnect Device", bei dem eine wesentliche Kenngröße die "Forwarding Rate" (FR) ist. Sie bezeichnet die Anzahl der erfolgreich weitergeleiteten Dateneinheiten pro Sekunde, die zuvor über eine Schnittstelle empfangen wurden. In [37, 159] wird auch empfohlen, die FR für verschiedene Paketlängen zu bestimmen, da bei größeren Paketen im Mittel weniger "Overhead" anfällt als bei kleineren Paketen.

3.2.2.1 Lastkenngrößen

Aus der vorangegangenen Diskussion ist ersichtlich, daß ein Lastgenerator für das Benchmarking von Packet Screens in der Lage sein muß, PDUs verschiedener Protokolle mit verschiedenen Attributen zu generieren. Das wichtigste Attribut ist in diesem Zusammenhang die PDU-Länge, also die Größe der versendeten Dateneinheit. Es bietet sich daher an, die PDU-Größe als ein Kriterium für die Ableitung von Auftragsklassen einzuführen. Man erhält damit verschiedene Auftragsklassen aufgrund der verschiedenen PDU-Längen.

Weitere wichtige Kriterien, die zu neuen Auftragsklassen führen könnten, sind PDU-Typ, Adreßinformationen und SAP-Informationen. Dies sind die am häufigsten verwendeten PDU-Attribute, die von einer Packet Screen ausgewertet werden, so daß in Abhängigkeit von der Wahl dieser Attribute und der Konfigurationskenngrößen der Packet Screen Auswirkungen auf die Leistungsfähigkeit zu erwarten sind.

Eine weitere wichtige Lastkenngröße ist die Rate, mit der Aufträge der oben angedeuteten Auftragsklassen an das zu vermessende System abgegeben werden können. Wichtig ist in diesem Zusammenhang, daß die maximale Rate des Lastgenerators unabhängig von der Auftragsklasse deutlich höher ist, als die erwartete Bedienrate des zu untersuchenden Systems. Andernfalls ist es fraglich, ob die Messung das zu untersuchende System charakterisiert oder lediglich Aussagen über die Leistungsfähigkeit des Lastgenerators erlaubt. Ähnliche Überlegungen gelten auch für die Leistungsfähigkeit des Monitors (vgl. [26]).

Die wichtigste Randbedingung ist jedoch, daß die Lastkenngrößen mit den vom Generator erstellten Aufträgen möglichst direkt korrespondieren. Diese Randbedingung ergibt sich aus der Notwendigkeit, bei der Verwendung von Universalrechner-basierten Meßwerkzeugen, die Aufträge an einer anderen Schnittstelle zu generieren und zu messen, als an der Schnittstelle, an der das SuL die Aufträge für die Bearbeitung entgegennimmt und nach der Bearbeitung ausliefert. In Anlehnung an [239] wird von den Generatoren eine primäre Last an der primären Schnittstelle (IF_{pG}) generiert. Das SuL wird jedoch durch transformierte Aufträge stimuliert, die als sekundäre Last an einer anderen, sekundären Schnittstelle (IF_{sG}) beim SuL eintreffen.

Die Generatoren übergeben — durch Aufrufen von Systemfunktionen — Dienstelemente an der UDP- oder TCP-Schnittstelle der Endsysteme, um Dateneinheiten mit bestimmten Attributen zu versenden. Das SuL empfängt diese Dateneinheiten in der Regel nach einer Transformation durch den Protokollstapel des Endsystems an seiner Ethernet-Schnittstelle. Nach der Bearbeitung auf der Packet Screen wird die Dateneinheit über die Ethernet-Schnittstelle des Monitors empfangen und nach einer weiteren Transformation wieder an der UDP/TCP-Schnittstelle des Endsystems an den Monitor-Prozeß übergeben.

Es dürfen daher nur solche Aufträge generiert werden, die zu einer deterministischen, idealerweise sehr einfachen Transformation dieser Aufträge und ihrer Attribute an allen Schnittstellen führen. Nur so kann sichergestellt werden, daß die beim Monitor beobachtete Leistung am IF_{pM} tatsächlich Rückschlüsse auf die vom SuL an der Schnittstelle IF_{sM} erbrachten Leistung zuläßt, die das SuL charakterisiert. Abbildung 3.9 zeigt die benannten Schnittstellen in einer Verfeinerung der Abbildung 3.1, in Anlehnung an [239]. Die sekundäre Last wird auf dem physikalischen Übertragungsmedium (hier Fast Ethernet, FE) an das SuL angelegt. Zusammenfassend ergibt sich folgender Zusammenhang zwischen den Schnittstellen und den Last- bzw. Leistungskenngrößen: Die Last wird an der Schnittstelle IF_{pG} angefordert. Die tatsächlich generierte Last liegt an der Schnittstelle IF_{sG} an. Die vom SuL erreichte Leistung liegt an der Schnittstelle IF_{sM} an und kann vom Monitor an der Schnittstelle IF_{pM} indirekt gemessen werden.

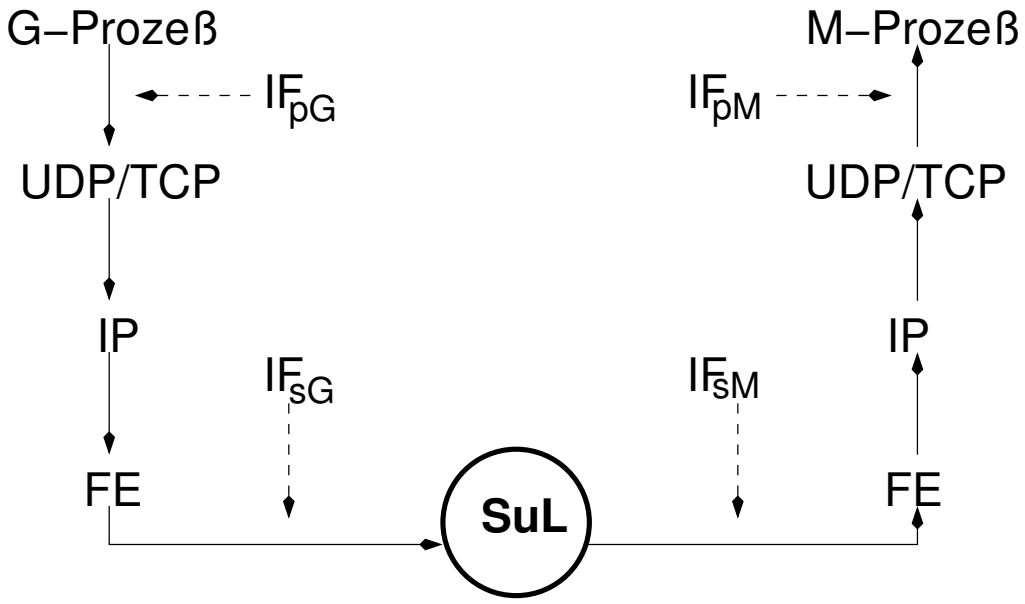


Abbildung 3.9: Indirekte Beobachtung, der an Schnittstelle IF_{sM} nicht meßbaren Leistung, an der Monitor-Schnittstelle IF_{pM}

3.2.2.2 Konfigurationskenngrößen

Konfigurationskenngrößen werden auf dem SuL vor der Durchführung einer Messung eingestellt. Neben Art und Typ der eingesetzten Packet Screen (Hard- und Software-Architektur) sind hier im wesentlichen die Anzahl der konfigurierten Filterregeln (vgl. [37, §7]), die pro Auftrag abgearbeitet werden müssen, zu nennen. Intuitiv erwartet man von der Anzahl der anzuwendenden Filterregeln Auswirkungen auf die Leistungsfähigkeit der Packet Screen. Ein weiteres wichtiges Konfigurationskriterium ist aber auch die Anzahl der eingesetzten Instanzen (Subsysteme bzw. "Devices under Test", s.o.), beispielsweise bei den Untersuchungen in Abschnitt 4.5.

Im Zusammenhang mit dem Benchmarking ist es wichtig, daß aufgrund der potentiell sehr großen Anzahl zu testender Konfigurationen die Meßwerkzeuge weitestgehend in der Lage sind, die für eine Meßreihe erforderlichen Konfigurationsschritte auf dem zu untersuchenden System selbständig (automatisch) vorzunehmen. Erstens geschieht dies in der Regel schneller als durch manuelle Konfigurationsschritte und zweitens ist diese Vorgehensweise weniger fehleranfällig als die manuelle Konfigurierung.[3] Die typischen Konfigurationskenngrößen werden in Abschnitt 3.4 im Detail diskutiert.

3.2.2.3 Leistungskenngrößen

Als wesentliche Leistungskenngröße für die Beurteilung der Leistungsfähigkeit von Packet Screens wird aufgrund der obigen Diskussion der Paketdurchsatz pro Sekunde angesehen. Andere Leistungskenngrößen können u.U. dann von dieser primären Größe abgeleitet werden:

- Der Durchsatz in Mbit/s (1 Mbit=10^6Bit) ergibt sich bei einer Einteilung der Auftragsklassen aufgrund der PDU-Größe unmittelbar aus dem Paketdurchsatz für Aufträge dieser Auftragsklasse multipliziert mit der PDU-Größe in Bit*10^{-6}. Voraussetzung dafür ist eine hierauf abgestimmte Konfigurierung der zu vermessenden Packet Screen, da der maximale Durchsatz nur dann gemessen werden kann, wenn alle Dateneinheiten von der Packet Screen auch tatsächlich weitergeleitet werden.

- Die Bedienzeit/Verzögerungszeit ergibt sich für den stationären Zustand des SuL als Kehrwert des maximalen Durchsatzes ("Maximum Forwarding Rate", MFR, vgl. [158]).

[3] Ein weiterer nicht zu unterschätzender Vorteil ist die zusätzliche Dokumentation der Konfigurationskenngrößen mit Hilfe automatischer Konfigurationsschritte. Letztendlich müssen alle Konfigurationskenngrößen für eine automatische Konfigurierung in einer maschinell interpretierbaren Form aufgeschrieben (programmiert) werden. Die Erstellung dieser Konfigurationswerkzeuge dokumentiert somit im Detail die hiermit später durchgeführten Einstellungen für die Meßreihen. Sie dokumentieren natürlich nicht die Motivation für die Durchführung der jeweiligen Meßreihe.

3.3 Anforderungen an Meßwerkzeuge

Die Leistungsfähigkeit eines Systems kann durch kooperierende Meßwerkzeuge bewertet werden, die zu bearbeitende Aufträge an das System senden und bearbeitete Aufträge vom System empfangen. Dieser Ansatz ermöglicht es, das zu beurteilende System als 'Black Box' zu betrachten. Es ist nicht erforderlich, das System für die durchzuführenden Messungen zu *instrumentalisieren*. Dadurch entfällt die unerwünschte Beeinflussung der Leistungsfähigkeit des zu untersuchenden Systems durch die Meßdatenerhebung.

Über eine oder mehrere Transformationen[4], der während einer Messung aufgezeichneten Rohdaten, kann dann auf die Leistungsfähigkeit des zu bewertenden Systems geschlossen werden.

Die indirekte Beobachtung eines Systems ist in diesem Fall nur möglich, weil die zu beobachtende Auftragsbearbeitung auf diesen Systemen nach außen sichtbar ist. Die Aufträge werden über das Kommunikationssubsystem empfangen und die Ergebnisse werden wiederum über dieses Kommunikationssubsystem versendet. Zu beachten ist hierbei, daß es sich um die normale Protokollverarbeitung auf den zu vermessenden Systemen handelt. Der Empfang der Aufträge (hier zu filternde Dateneinheiten) und das Versenden der Ergebnisse (die gefilterten Dateneinheiten) ist der normale Betriebszustand der Systeme. Die Messung kann daher ohne jeden Eingriff in die Auftragsabarbeitung erfolgen.

Da die Leistungsfähigkeit indirekt an der Schnittstelle IF_{pM} (vgl. Abbildung 3.9) beobachtet wird, müssen die Meßwerkzeuge an die Schnittstellen des zu beobachtenden Systems angepaßt werden. Die Meßwerkzeuge müssen in der Lage sein, die an den Schnittstellen des zu vermessenden Systems (IF_{sM}) vorkommenden Aufträge nachzubilden und zu analysieren. Typischerweise können an derselben physikalischen Schnittstelle Aufträge verschiedener Auftragsklassen oder Aufträge mit verschiedenen Attributen übergeben werden. Bei Messungen an Kommunikationssystemen entsprechen diese verschiedenen Auftragsklassen beispielsweise den verschiedenen Nachrichtentypen oder verschiedenen Nachrichtenlängen, die auf einem bestimmten Medium übertragen werden können (vgl. [22]).

Zusammenfassend müssen Meßwerkzeuge für ein systematisches Benchmarking den folgenden Anforderungen genügen:

- Der Lastgenerator muß die Generierung unterschiedlicher Auftragsklassen unterstützen. Insbesondere müssen bei der Analyse von Netzkomponenten Typ, Länge und ggf. der Inhalt von Dateneinheiten variiert werden können.
- Der Lastgenerator und der Monitor müssen in der Lage sein, die für eine statistische Absicherung der Ergebnisse erforderliche Anzahl von Versuchswiederholungen selbständig durchzuführen und die daraus resultierenden Vertrauensintervalle zu berechnen. Dies ist für die Ergebnisanalyse erforderlich, um statistisch signifikante Unterschiede zwischen Meßreihen von zufälligen Schwankungen unterscheiden zu können.

[4] Eine Transformation leitet aus Rohdaten weitere Meßergebnisse ab. Beispielsweise kann der Durchsatz in Megabit pro Sekunde aus der Anzahl der empfangenen Pakete und der Meßdauer errechnet werden.

- Die Werkzeuge müssen einen Export der Meßergebnisse in einem maschinell leicht weiter zu verarbeitenden Format unterstützen. So lassen sich aus den Meßergebnissen die für eine Interpretation hilfreicheren, graphischen Darstellungen generieren.

- Die Meßwerkzeuge müssen in der Lage sein, Messungen mit anderen Werkzeugen zu synchronisieren. So können statistische Abhängigkeiten von anderen Ereignissen nachgewiesen oder widerlegt werden. Ein anderes Beispiel ist die Erhöhung der generierten Last durch die Synchronisation mehrerer Lastgeneratoren. Diese Vorgehensweise ist erforderlich, wenn das zu vermessende System leistungsfähig genug ist, um alle Aufträge eines Lastgenerators zu verarbeiten.

Im Einzelfall können noch diverse weitere Anforderungen an Meßwerkzeuge gestellt werden. Ein großes Problem für die Durchführung der angestrebten Untersuchungen war, daß bereits die wenigen eben diskutierten Anforderungen nur unzureichend von den verfügbaren Werkzeugen unterstützt wurden. Die folgenden Abschnitte diskutieren daher die Modifikationen an frei verfügbaren Meßwerkzeugen[5], um den obigen Anforderungen zu genügen. Es handelt sich hierbei um den Lastgenerator *netperf* und den dazu passenden Monitor *netserver*. Eine etwas ausführlichere Darstellung ist in Anhang C zu finden.

3.3.1 Statistische Absicherung

In der Version 2.1 des *netperf*-Tools gibt es eine sehr eigenwillige Methode zur statistischen Absicherung. Anstelle über eine bestimmte Anzahl von Wiederholungen die Vertrauensintervalle zu berechnen, wird dort lediglich überprüft, ob die gemessenen Werte größeren Schwankungen unterliegen. Für eine vergleichende Auswertung ist es jedoch erforderlich, die Vertrauensintervalle auszugeben. Sie können dann in graphischen Darstellungen der Meßergebnisse automatisch eingezeichnet werden und ermöglichen dann eine einfache Kontrolle der Meßergebnisse.

Die grundsätzliche Annahme ist natürlich, daß die gemessenen Werte einer Normalverteilung genügen. Typischerweise wird dann für die Berechnungen der Vertrauensintervalle eine sehr große Anzahl von Versuchen (beispielsweise $n > 40$) durchgeführt, um die Intervallgröße mit Hilfe von Faktoren für den 95%-Vertrauensbereich (Faktor $= 1,96$) oder den 99%-Vertrauensbereich (Faktor $= 2,58$) abzuschätzen. Dieses Verfahren ist unbefriedigend, da es zu einer hohen Anzahl von Versuchswiederholungen zwingt. Abhängig davon, wie lange ein einziger Versuch dauert, kann dadurch die gesamte Meßdauer ausufern.

Aus diesem Grund wurden zwei für die oben angedeuteten Kontrollen bereits vorhandene Tabellen mit Werten für die Student-t-Verteilung erweitert. In Abhängigkeit von der genauen Anzahl der Versuchswiederholungen können nun die genauen Faktoren zur Berechnung der Vertrauensintervalle aus den Tabellen für den 95%-Vertrauensbereich und den

[5] *http://www.netperf.org/netperf/NetperfPage.html*

99%-Vertrauensbereich eingelesen werden. Dadurch erhält man bereits für wenige Versuchwiederholungen genaue (aber natürlich unschärfere) Vertrauensbereiche. Es ist nun möglich, daß in Abhängigkeit von der Dauer einer Versuchswiederholung die Anzahl der Wiederholungen gewählt werden kann, ohne daß auf genaue Vertrauensintervalle verzichtet werden müßte.

3.3.2 Trennung von Konfigurations- und Meß-Dienstzugangspunkt

Wie bereits in Abschnitt 3.3 angedeutet wurde, müssen Lastgenerator und Leistungsmonitor miteinander kooperieren, um die verteilt erhobenen Meßergebnisse zentral zu archivieren. Die Originalversionen der Meßwerkzeuge (*netperf* und *netserver*) unterstützten lediglich die Auswahl einer Adreßangabe, an die der Lastgenerator sowohl die Kontrollverbindung als auch die generierte Last adressierte.

Normalerweise ist diese Vorgehensweise für ein Benchmarking ausreichend. Allerdings ist es hiermit nicht möglich, die Auswirkung von verlustbehafteter Kommunikation zu testen, da die Kontrollverbindung zur Versuchssteuerung ebenfalls durch verlustbehaftete Netzkomponenten gestört werden würde. Die Werkzeuge wurden daher so erweitert, daß die generierten Meßdaten an eine andere Adresse gesendet werden können als an die Adresse, die für die Kontrollverbindung benutzt wird. Diese Vorgehensweise ermöglicht die ungestörte Kommunikation zwischen Lastgenerator und Leistungsmonitor über ein alternatives Rechnernetz, während die eigentliche Last über ein verlustbehaftetes Netz gesendet werden kann.

Diese Trennung der verschiedenen Kommunikationsbeziehungen zwischen den Meßwerkzeugen hat den weiteren Vorteil, daß eine Verfälschung der Meßdaten durch die Kontrollverbindung – auch ohne Kenntnis des Kontrollprotokolls – ausgeschlossen werden kann.

3.3.3 Synchronisation mehrerer Meßinstanzen

Die Durchführung von Messungen mit *netperf* und *netserver* beschränkte sich bei der Originalversion auf jeweils einen Lastgenerator und einen Monitor. Dies hat mehrere Nachteile:

- Zu vermessende Systeme können so nur einer unidirektionalen Last ausgesetzt werden. Messungen mit einer bidirektionalen oder auch multidirektionalen Last sind nicht möglich.
- Es können keine Aussagen über statistische (Un)Abhängigkeiten einer Messung von gleichzeitig erzeugter Grund- oder Störlast gemacht werden.
- Bei der Bewertung sehr leistungsfähiger Systeme können die Lastgeneratoren und Monitore selbst zum Engpaß werden. Eine Koppelung mehrerer Meßsysteme zur Erhöhung der Last und der Meßkapazität ist in solchen Fällen erforderlich.

Die Meßwerkzeuge wurden erweitert, so daß sich mehrere Lastgeneratoren vor der Durchführung von Messungen synchronisieren. Dadurch sind alle drei oben dargestellten Einschränkungen umgehbar. Details hierzu befinden sich in Anhang C.

3.4 Wahl geeigneter Parameter und Konfigurierung der Meßsysteme

In dem vorigen Abschnitt wurden Anforderungen an Meßwerkzeuge diskutiert, die durch eine Anpassung oder Erweiterung vorhandener Software umgesetzt wurden. Die resultierenden Programme werden auf Rechnern eingesetzt, die besonders leistungsfähig sein müssen, um eine ausreichende hohe Last zu generieren. Unterabschnitt 3.4.1 beschreibt die für die Messungen verwendeten Systeme. In den darauffolgenden Unterabschnitten werden die, auf Grundlage der oben formulierten Anforderungen gewählten, Last- und Konfigurationskenngrößen vorgestellt.

3.4.1 Verwendete Hardware für die Meßdurchführung

Eine für die Bewertung des SuL "ausreichend hohe Last" kann nur dann generiert werden, wenn bei allen Versuchen die Leistungsfähigkeit der Meßsysteme größer ist als die der zu bewertenden Systeme. Nur so sind beispielsweise Aussagen über den Grenzdurchsatz der zu bewertenden Systeme (SuL) möglich. Bei allen Meßreihen im Testnetz wurden daher für die Meßsysteme (für Lastgenerator und Monitor) folgende Rechner eingesetzt:

	Meßsystem	SuL
Typ	Sun Ultra 2	Sun Ultra 1
CPU	1–2 à 300 MHz	1 à 143 MHz
Schnittstelle	100 Mbit/s Fast Ethernet	

Tabelle 3.1: Meßsysteme vs. SuL

Durch die deutlich leistungsfähigere Ausstattung (Dual-CPU, doppelte Taktfrequenz) sollte sichergestellt werden, daß die Meßsysteme nicht selbst zum Engpaß bei der Leistungsbewertung werden.

Um für jede Messung vergleichbare Randbedingungen herzustellen, wurden die Rechner alle homogen konfiguriert. Die zu vermessenden Systeme führten während aller Messungen die gleichen Prozesse aus und systemnahe Einstellungen waren ebenfalls identisch über alle Meßreihen konfiguriert. Vor jeder Meßreihe wurde ein Neustart der zu vermessenden Systeme durchgeführt ("Reboot"). Dadurch konnte sichergestellt werden, daß auch die vom Betriebssystem verwendeten Ressourcen vor jeder Messung in einen Grundzustand zurückversetzt werden.

3.4.2 Größe der "Service Data Unit" auf der Transportebene

In Abschnitt 3.2.2 wurde motiviert, daß die Leistungsfähigkeit einer Packet Screen am Besten als "Forwarding Rate" angegeben wird. Es soll gemessen werden, wie viele Dateneinheiten pro Sekunde von der Packet Screen empfangen und weitergeleitet werden können. Dabei sollen Abhängigkeiten von der Paketgröße beachtet werden, so daß der Generator einen Paketstrom generieren muß, bei dem alle Pakete eine vorgegebene Länge aufweisen. Der Generator fordert das Versenden einer Dateneinheit an der UDP/TCP-Schnittstelle der Endsysteme an (IF_{pG}, vgl. Abschnitt 3.2.2.1). Hier besteht die Möglichkeit, über die Größe der Nutzdaten ("Service Data Unit", SDU) die Länge des resultierenden Ethernet-Rahmens an der sekundären Schnittstelle (IF_{sG}) zu wählen.

Es ist demnach zu erwarten, daß sowohl der Datagrammdurchsatz[6] als auch der Datendurchsatz unmittelbar von der Wahl der SDU-Größe auf der Transportebene abhängen. Je größer die SDU ist, desto besser ist das Verhältnis zwischen Protokoll-"Overhead" (im folgenden: Overhead[7]) und transportierten Nutzdaten. Die SDU-Größe kann von der Anwendung im `send()` und `sendto()` Systemaufruf festgelegt werden. Für UDP auf Basis von Ethernet gilt beispielsweise:

$$\text{Overhead } = \frac{\text{PDUsize} - \text{SDUsize}}{\text{PDUsize}}, 0 \leq \text{SDUsize} \leq 1472 \tag{3.2}$$

Analog gilt für TCP:

$$\text{Overhead } = \frac{\text{PDUsize} - \text{SDUsize}}{\text{PDUsize}}, 0 \leq \text{SDUsize} \leq 1460 \tag{3.3}$$

Werden mehr als 1472 Byte von der Anwendung an den UDP-Dienst übergeben, muß dieser die Daten auf zwei PDUs aufteilen, da die "Maximum Transfer Unit" (MTU)-Größe andernfalls überschritten würde. Während der UDP-Dienst die Aufteilung in zwei PDUs der IP-Schicht überläßt, die dann eine Fragmentierung durchführt, übernimmt TCP selbständig die Aufteilung und sendet ggf. mehrere Segmente. In beiden Fällen ist der Overhead jedoch bei Überschreiten der maximalen SDU-Größen nicht mehr monoton fallend, wie in Formel (3.2) und Formel (3.3) dargestellt.

Die Angabe des Overheads bezieht sich immer nur auf eine bestimmte Protokoll-Ebene. Innerhalb eines Protokollstapels kann es Abhängigkeiten zwischen den Protokollen auf verschiedenen Ebenen geben. Bei Ethernet wird die maximale SDU-Größe (bei UDP 1472 Byte, bei TCP 1460 Byte) auf der Transport-Ebene exakt so gewählt, daß durch den zusätzlichen Overhead gerade die konfigurierte IP-MTU erreicht wird. Die IP-MTU wird typischerweise wiederum an die maximale Rahmen-Größe auf der Datensicherungsschicht

[6] In dieser Arbeit werden die Begriffe Datagramm und Paket synonym verwendet.

[7] Im folgenden wird nur der Overhead durch zusätzliche Protokoll-Informationen betrachtet, die in der jeweiligen PDU kodiert sind. Der X.200 Name für diesen Overhead ist "Protocol Control Information" (PCI). Weiterer Overhead durch andere PDUs (bei TCP beispielsweise Bestätigungssegmente) wird hier nicht betrachtet.

angepaßt. So ist die IP-MTU beim Einsatz von Ethernet-Technologien[8] typischerweise auf 1500 Byte festgelegt.[9] Dies entspricht der maximalen Ethernet-SDU (Ethernet Daten, vgl. Abbildung 3.10).

3.4.3 Puffergröße des "Service Access Points"

Der "Service Access Points" (SAP) ist ein Dienstzugangspunkt zwischen den Protokollschichten, der adressierbar ist und somit eine Kommunikationsbeziehung in einem Endsystem identifiziert. Auf den eingesetzten Workstations werden SAPs durch Betriebssystemfunktionen implementiert. Der Zugriff auf die SAPs erfolgt über Systemaufrufe. Neben typischen Dienstprimitiven zum Senden und Empfangen von Dateneinheiten gibt es weitere, die eine Anpassung der SAPs an die Anforderungen des Dienstnehmers ermöglichen.

Durch systematische Messungen hat sich herausgestellt, daß die Größe der Sende- und Empfangspuffer in den SAPs großen Einfluß auf den Durchsatz bei der Datenkommunikation haben. Für Messungen im lokalen LAN hat sich ein Wert von 64 Kilobyte (KByte[10]) für die Sende- und Empfangspuffergröße bewährt [78]. Dieser Wert wird von den Meßwerkzeugen bei allen Ethernet-Meßreihen als Sende- und Empfangspuffergröße konfiguriert.

3.4.4 Transportprotokoll

In der TCP/IP-Protokollfamilie stehen auf der Transportebene zwei Protokolle zur Auswahl. Das "User Datagram Protocol" (UDP) wird für verbindungslose Kommunikation verwendet. Das "Transmission Control Protocol" (TCP) bietet hingegen einen verbindungsorientierten Dienst an. TCP sichert ferner die Einhaltung der Reihenfolge der übertragenen Daten und korrigiert automatisch Übertragungsfehler.

Bei der Bewertung des Datagramm-Durchsatzes ist TCP leider ungeeignet. Obwohl bei den Messungen nur unidirektional Nutzdaten gesendet werden, schickt die TCP-Instanz des Empfängers TCP-PDUs mit Bestätigungen an den Sender zurück. Die vom Empfänger gesendeten PDUs beeinflussen die Messung erheblich, können jedoch vom Meßwerkzeug nicht ausgewertet werden, da diese Protokollfunktionalität für die Anwendungen (Lastgenerator und Monitor) transparent ist. Die Last durch TCP-PDUs mit Bestätigungen fällt an der Schnittstelle IF_{sM} (vgl. Abbildung 3.9) an und wird vom SuL ebenfalls bearbeitet. An den Schnittstellen IF_{pM} und IF_{pG} ist diese Störlast jedoch nicht meßbar und auch nicht berechenbar. Das TCP-Protokoll sendet nicht in jedem Fall eine Bestätigung für ein empfangenes TCP-Segment, da es mit einer Bestätigung auch den Inhalt von mehreren TCP-Segmenten bestätigen kann. Es gibt verschiedenartige Wechselwirkungen

[8] Heute gebräuchliche Ethernet-Technologien sind Ethernet (10 Mbit/s), Fast Ethernet (100 Mbit/s) und Gigabit Ethernet (1 Gbit/s).

[9] Bei Gigabit-Ethernet gibt es zusätzlich sog. "Jumbo-Frames", die größere SDUs erlauben.

[10] 1 KByte entspricht 1024 Byte.

zwischen der Netzstruktur, der TCP-Flußkontrolle und konfigurierbaren Parametern innerhalb der TCP-Module der Endsysteme und des SuLs, die Einfluß auf die Häufigkeit der TCP-Bestätigungen haben, so daß die Störlast nicht kalkulierbar ist.

Die TCP-Flußkontrolle könnte eine weitere Ursache für nicht vorhersehbare Phänomene sein. Ist beispielsweise der Monitor nicht leistungsfähig genug, um alle Dateneinheiten zu empfangen, würde die TCP-Flußkontrolle automatisch eine Reduzierung der Senderate durch den Generator herbeiführen. Dies ist nicht wünschenswert, da so die Last über die Gesamtdauer der Messung (s.u.) variiert, was die Auswertung deutlich erschwert.

UDP hingegen ist ein reiner "best effort"-Dienst. Dateneinheiten werden nicht automatisch bestätigt, so daß bei der Bewertung des Datagramm-Durchsatzes nicht befürchtet werden muß, daß das empfangende Meßsystem den Durchsatz auf der Packet Screen beeinflußt. Für alle Leistungsbewertungen von Packet Screens wird daher ausschließlich UDP verwendet. Andere Phänomene, die auch beim Einsatz von UDP noch zu einer Störlast führen könnten, müssen durch die richtige Konfigurierung der Versuchsanordnung ausgeschlossen werden. Beispielsweise könnten "ICMP-Unreachable"-Dateneinheiten vom SuL generiert werden, falls es keinen Eintrag in der Wegwahl-Tabelle des SuL gibt, der das Weiterleiten der Dateneinheiten an den Monitor ermöglicht.

3.4.5 Meßdauer, Wiederholungen und Vertrauensintervalle

Die Meßdauer muß ebenfalls zwei Randbedingungen genügen. Sie darf nicht zu kurz sein, da in diesem Fall die Messung aufgrund der asynchronen Verarbeitung in den Meßsystemen zu ungenau ist. Moderne UNIX-Systeme[11] implementieren das Kommunikationssubsystem in der Regel mit einer Reihe von STREAM-Modulen (vgl. [229, S.547ff]). Jedes dieser Module (beispielsweise für IP, UDP, TCP, Ethernet und ATM) kann über eigene Flußkontrollmechanismen verfügen und eigenständig Speicherplatz für STREAM-Nachrichten verwalten. Dadurch ist beispielsweise eine asynchrone Nachrichtenübertragung möglich: die Anwendungen können mehrere Nachrichten an das Kommunikationssubsystem übergeben, das die Dateneinheiten lokal zwischenspeichert und anschließend die Übertragung übernimmt.

Meßwerkzeuge müssen die Effekte durch die asynchrone Übertragung beachten. Ist die Anzahl der zu übertragenen Daten sehr klein, kann das Kommunikationssubsystem diese Daten sofort entgegennehmen. Das Meßwerkzeug ermittelt u.U. aus der Zeitdifferenz zwischen "Versenden" der ersten und der letzten Dateneinheit der Meßreihe extrem hohe Übertragungsgeschwindigkeiten und Datagrammdurchsätze, da die Systemaufrufe nicht zwischen "Versenden" und "zum Versenden zwischenspeichern" unterscheiden.

Es hat sich gezeigt, daß bei einer Übertragungsgeschwindigkeit ab 100 Mbit/s auf den oben genannten Systemen mindestens 10 Sekunden lang Aufträge (Dateneinheiten) vom

[11] Marktüblich sind SystemVR4-Systeme, beispielsweise von HP mit dem Betriebssystem HP-UX, von IBM mit dem Betriebssystem AIX und von Sun mit dem Betriebssystem Solaris.

Lastgenerator erzeugt werden müssen. Auch bei sehr kleinen Dateneinheiten reicht diese Zeitspanne aus, um auf dem Lastgenerator den Meßfehler durch das Zwischenpuffern auf einen vernachlässigbar kleinen Wert zu senken. Dies konnte durch Vergleichsmessungen mit längerer Meßdauer ermittelt werden.

Der erreichbare Durchsatz ist bei aufeinanderfolgenden gleichartigen Wiederholungen der Messung Schwankungen unterworfen. Ursache für diese Schwankungen können beispielsweise zufällig auftretende Fehler bei der Übertragung sein. Es ist daher erforderlich, die einzelnen Messungen wiederholt durchzuführen. Durch Mittelung der Meßwerte und Berechnen der Vertrauensintervalle kann eine genauere Abschätzung bezüglich der Reproduzierbarkeit abgegeben werden als bei einer einmaligen Versuchsdurchführung. Zu diesem Zweck wurden Funktionen zur Berechnung der Mittelwerte und Vertrauensintervalle erstellt (vgl. Anhang C). Bei allen Meßreihen wird jeder Meßpunkt mindestens 30 mal wiederholt. Danach wird ein Mittelwert und ein Vertrauensintervall für ein Konfidenzniveau von 99% berechnet. Die Meßwerkzeuge geben diese Ergebnisse zusammen mit den übrigen Versuchsparametern in einer maschinell leicht interpretierbaren Form aus. Das Zeichnen der Meßdaten erfolgt mit einem frei verfügbaren Programm zur graphischen Darstellung von Daten. Die Aufbereitung der Meßdaten (Rohdaten) für dieses Werkzeug erfolgt mit selbst erstellten UNIX-“Shell Scripts”.

3.4.6 Anzahl der Filterregeln

Ein weiterer wichtiger Freiheitsgrad bei der Meßdatenerfassung stellt die Anzahl der auf dem SuL konfigurierten Filterregeln dar. Es wurde bereits in Abschnitt 3.2.2.2 motiviert, daß bei Netzkomponenten, die Filterregeln auf Dateneinheiten anwenden müssen, Auswirkungen auf den erreichbaren Datagrammdurchsatz zu erwarten sind.

Da sichergestellt werden muß, daß tatsächlich die gewünschte Anzahl an Filterregeln auf jedes Datagramm angewendet wird, hat sich in der Praxis folgendes Vorgehen zum Konfigurieren von m Filterregeln bewährt:

1. Konfigurieren des SuL, so daß alle Filterregeln sequentiell abgearbeitet werden.
2. Konfigurieren von $m-1$ Filterregeln, die ein Blockieren der Dateneinheiten bewirken.
3. Konfigurieren einer letzten Filterregel (Regel m), die das Weiterleiten der Dateneinheiten erlaubt.

Durch diese Vorgehensweise kann erreicht werden, daß alle eintreffenden Dateneinheiten erst nach Anwenden der gewünschten Anzahl (m) an Filterregeln von dem SuL weitergeleitet werden. Der erste Punkt der Liste ist systemspezifisch, d.h. die sequentielle Regelverarbeitung muß individuell für verschiedene Software-Produkte, ggf. durch zusätzlichen Programmieraufwand, erreicht werden.

3.5 Theoretischer Durchsatz und Referenzmessungen

Abhängig von der eingesetzten Netztechnologie ergeben sich theoretische Werte für den maximal erreichbaren UDP/TCP-Datagramm- und Datendurchsatz. Dieser theoretische Durchsatz bildet die obere Schranke bei allen Durchsatzmessungen. Die theoretischen Durchsatzschranken dienen daher als absolute Vergleichswerte. Die Meßergebnisse können diesen Vergleichswerten für eine Bewertung gegenübergestellt werden.

Neben den absoluten Vergleichswerten gibt es relative Vergleichswerte. Die Implementierung eines bestimmten Protokolls auf einer bestimmten Hardware-Architektur hat Einfluß auf den erreichbaren Durchsatz. Für die oben dargestellten Geräte wurden daher Vergleichsmessungen erstellt, die Aussagen über den (empirisch) ermittelten maximalen Durchsatz ermöglichen. Auch diese Werte können mit den Meßergebnissen über eine Packet Screen verglichen werden. Dadurch werden konkrete Aussagen zur Leistungseinbuße durch den Einsatz einer Firewall-Komponente (hier einer Packet Screen) möglich.

Maximaler Durchsatz bei TCP/IP über Fast Ethernet

Fast Ethernet ist für eine Brutto-Übertragungsgeschwindigkeit von 100 Mbit/s spezifiziert. Der Fast Ethernet Rahmen ist identisch mit dem bekannten Ethernet Rahmen (vgl. Abbildung 3.10). Für die Berechnung des maximalen Durchsatzes auf der Anwendungsebene sind zwei Aspekte zu beachten:

1. Aufeinanderfolgende Rahmen können nicht direkt hintereinander gesendet werden. Es muß eine sog. “Inter-Frame Gap” (IFG) eingehalten werden. Die IFG beträgt bei Ethernet $9,6\mu s$ und bei Fast Ethernet 960 Nanosekunden. Dies entspricht in beiden Fällen der Zeit zum Versenden von 96 Bit (12 Byte). Für die Berechnung des maximalen Durchsatzes ist es daher erforderlich, diese 12 Byte zur jeweiligen Rahmenlänge zu addieren.

2. Die minimale Rahmenlänge (ohne Präambel und IFG) darf 64 Byte nicht unterschreiten. Gegebenenfalls sind Fülldaten vor dem Versenden im Datenbereich einzutragen, um die Unterschreitung zu verhindern. Die maximale Rahmenlänge beträgt 1518 Byte.

Für IP-Protokolle ergibt sich daher eine MTU von 1500 Byte beim Einsatz von Fast Ethernet. Zum Erreichen der minimalen Rahmenlänge ist beim Einsatz von TCP (UDP) eine TCP-SDU (UDP-SDU) von mindestens 6 Byte (18 Byte) erforderlich, so daß ggf. bis zu

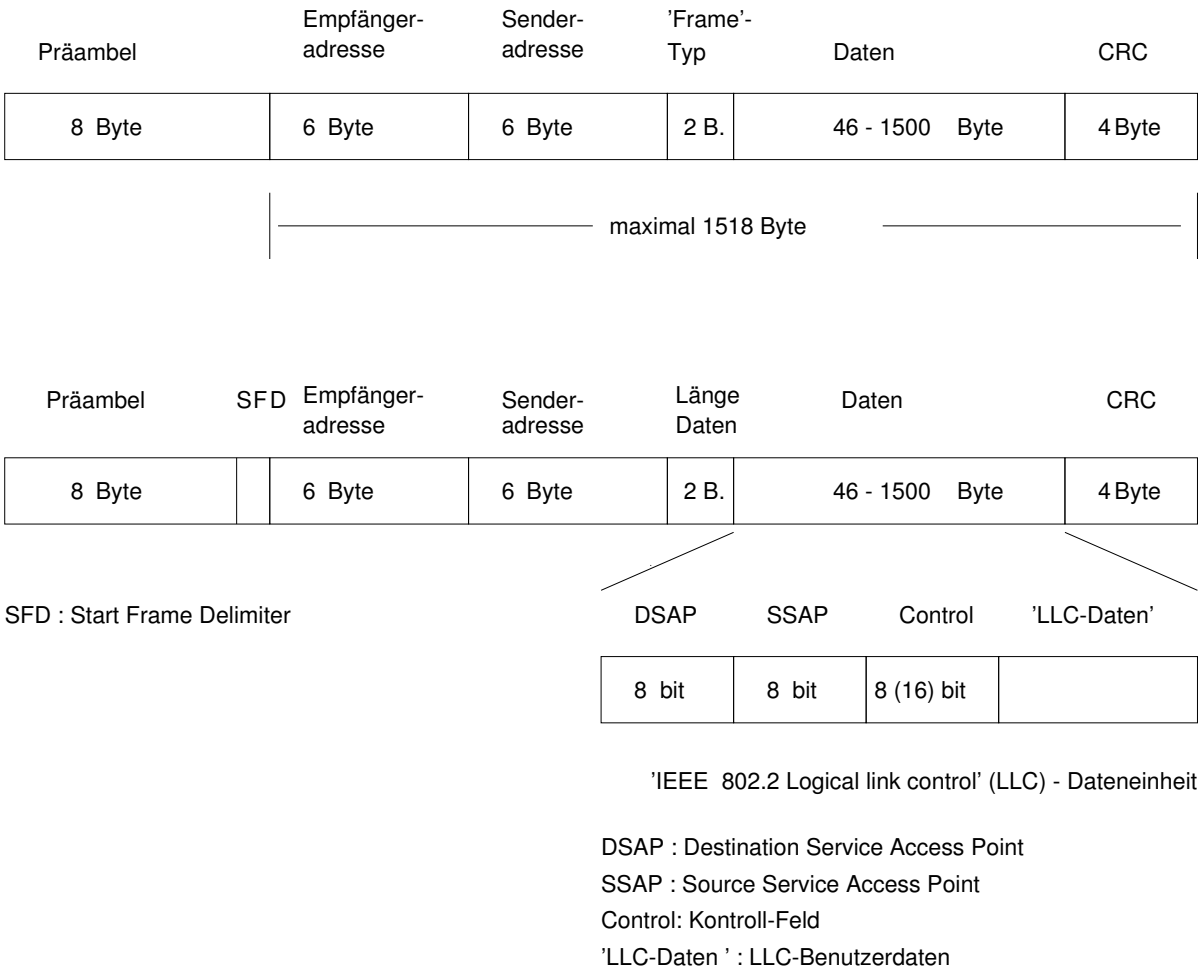


Abbildung 3.10: (Fast) Ethernet Rahmen (vgl. [29, S.124])

sechs (UDP: 18) Füllbyte[12] eingefügt werden müssen. Die Ethernet Rahmenlänge (in Byte) kann in Abhängigkeit von der Nutzdatenlänge (x) berechnet werden zu:

$$\begin{aligned} \text{TCP-Rahmenlänge}(x) &= (\text{IFG} = 12) + (\text{MAC Overhead} = 26) + (\text{IP/TCP-Kopf} = 40) \\ &\quad + max(6, x),\ 0 \leq x \leq 1460 \qquad (3.4) \\ \text{UDP-Rahmenlänge}(x) &= (\text{IFG} = 12) + (\text{MAC Overhead} = 26) + (\text{IP/UDP-Kopf} = 28) \\ &\quad + max(18, x),\ 0 \leq x \leq 1472 \qquad (3.5) \end{aligned}$$

Unabhängig vom verwendeten Transportprotokoll (TCP oder UDP) ist der minimale Fast Ethernet Rahmen (inklusive Präambel und IFG) mindestens 84 Byte lang. Pro Sekunde können theoretisch 148810 Rahmen dieser Länge versendet werden. Die maximale Länge

[12] Der Anwender braucht diese Fülldaten nicht selbst anzugeben. Sie werden automatisch vom Gerätetreiber der Betriebssysteme oder vom Ethernet-Chip bei Bedarf zur SDU hinzugefügt.

beträgt 1538 Byte und wird bei einer UDP-Datenlänge von 1472 Byte erreicht. Pro Sekunde können 8127, 44 Rahmen dieser Länge versendet werden. Der maximale Datendurchsatz bei Verwendung von UDP beträgt dann 95, 71 Mbit/s bzw. 94, 93 Mbit/s bei Verwendung von TCP. In Abbildung 3.11 sind die Kurvenverläufe für UDP und TCP über Fast Ethernet graphisch dargestellt. Der theoretisch maximal erreichbare Datagrammdurchsatz ist über die Zeit invariant, d.h. der Zeitpunkt t der Meßdatenerfassung ist beliebig. Bei $\Delta t = 1\ s$ ist der Datagrammdurchsatz laut Formel 3.1 nur noch von der Auftragsklasse a abhängig. Eine Auftragsklasse besteht aus Ethernet-Rahmen gleicher Länge, so daß der Durchsatz $D_a(\Delta t = 1s)$ unter Verwendung von Formel 3.4 gegeben ist durch:

$$D_{\text{Rahmenlänge}(x)}(\Delta t = 1sec) = 12500000\frac{\text{Byte}}{s}/\text{Rahmenlänge}(x)\frac{\text{Byte}}{\text{Rahmen}} \tag{3.6}$$

Die Einheit ist Rahmen pro Sekunde bzw. UDP/TCP-Datagramme pro Sekunde, da die Beschränkung von x in Formel 3.4 eine 1 : 1-Zuordnung von Ethernet-Rahmen zu Datagrammen ermöglicht. Die Anzahl dieser Rahmen multipliziert mit der SDU-Länge (ein Faktor 8 für die Umrechnung von Byte in Bit ist zu beachten) ergibt den Datendurchsatz in Mbit/s ($D_b(\Delta t = 1s)[Mbit/s]$), der zusätzlich auf der rechten Y-Achse von Abbildung 3.11 abgelesen werden kann.

3.6 Packet Screen als Engpaß in Hochgeschwindigkeitsnetzen

Mit den folgenden, ersten Leistungsdaten wird ein Engpaß bei der Kommunikation, verursacht durch den Einsatz einer Packet Screen, aufgezeigt. Es werden die Abhängigkeiten von der Anzahl der Filterregeln und der Datagrammlänge aufgezeigt.

Zuvor wird in Abschnitt 3.6.1 die Bearbeitung von Filterregeln in einer PS dargestellt, um die Abhängigkeit des Durchsatzes von der Anzahl der konfigurierten Filterregeln zu verdeutlichen. Hierauf wurde bereits in Abschnitt 3.2.2.2 und im vorigen Unterabschnitt hingewiesen. Hier sollen die technischen Abläufe noch einmal skizziert werden, da sie entscheidend für die späteren Argumentationen und weiteren Konfigurationen sind.

3.6.1 Verarbeitung von Filterregeln bei Packet Screens

Filterregeln werden auf Packet Screens mit dem Ziel konfiguriert, eine Zugriffskontrolle basierend auf Kommunikationsbeziehungen zu ermöglichen:

Definition (3.1): **Kommunikationsbeziehung**

> Eine Kommunikationsbeziehung ist gekennzeichnet durch zwei oder mehrere räumlich voneinander getrennte Kommunikationspartner (beispielsweise Prozesse), die über ein Netz verbindungslos oder verbindungsorientiert Nachrichten austauschen. Bei verbindungsorientierter Kommunikation ist die Dauer der

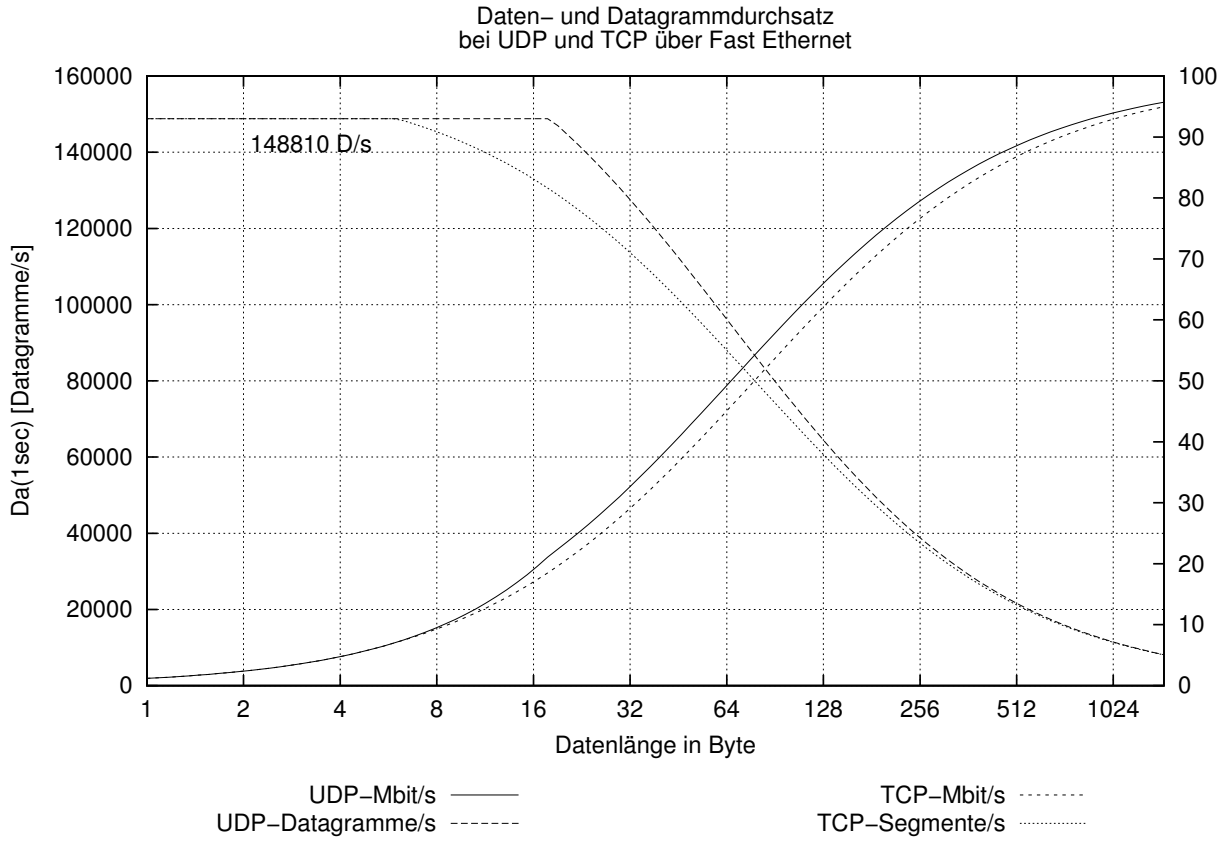


Abbildung 3.11: theoretisch möglicher Durchsatz bei TCP/IP über Fast Ethernet

> Kommunikationsbeziehung durch Verbindungsaufbau und Verbindungsabbau begrenzt. Verbindungslose Kommunikationsbeziehungen dauern an, so lange der Sender Daten an denselben Empfänger bzw. an dieselben Empfänger sendet.

Entscheidend an der Definition ist bei der verbindungslosen Kommunikation die Intention des Senders, Dateneinheiten an einen Empfänger zu senden. Es ist (extensional) nicht beobachtbar bzw. entscheidbar, wann eine verbindungslose Kommunikationsbeziehung beendet ist. Nach Empfang einer verbindungslos gesendeten Dateneinheit weiß der Empfänger nicht, ob oder wann eine weitere Dateneinheit von demselben Sender empfangen wird.

Eine PS muß alle Dateneinheiten aller Kommunikationsbeziehungen zwischen zwei Netzen untersuchen. Kritisch (im Sinne der Leistungsfähigkeit) ist bei dieser Analyse die Geschwindigkeit, mit der die Packet Screen eine *positive* Entscheidung trifft und die Dateneinheit anschließend in das jeweils andere Netz weiterleitet. Dateneinheiten, die zu

"erlaubten" Kommunikationsbeziehungen[13] gehören, sollen also möglichst wenig verzögert werden. Alle anderen Dateneinheiten müssen von der PS vernichtet werden, so daß die nicht erwünschten Kommunikationsbeziehungen verhindert werden. Im praktischen Einsatz ist daher das Weiterleiten der Dateneinheiten ("Forwarding") von erlaubten Kommunikationsbeziehungen der zu optimierende Fall ("Mainstream") bei der Protokollverarbeitung auf einer PS. Das Vernichten von Dateneinheiten ist dagegen eher die Ausnahme.[14]

Die Unterscheidung zwischen "erlaubter" und "unerwünschter" Kommunikation basiert auf den jeweiligen Richtlinien. Diese Richtlinien werden auf PSs mit Hilfe von *Filterregeln* implementiert. Offensichtlich ist der Aufwand für das Filtern auf einer PS abhängig von der Anzahl der Filterregeln, die auf jedes Datagramm angewendet werden müssen. Wieviele Filterregeln tatsächlich auf jedes Datagramm angewendet werden, hängt jedoch ganz wesentlich von der Implementation der Filterverarbeitung ab:

Lineare Liste: Bei dieser Implementierung werden die Filterregeln sequentiell in der konfigurierten Reihenfolge abgearbeitet. Eine Entscheidung wird frühestens dann getroffen, wenn eine Filterregel auftritt, die Bezug auf die Attribute des jeweiligen Datagramms nimmt, und spätestens dann, wenn die gesamte Liste abgearbeitet ist.

Streuliste: Die Filterregeln werden anhand der Attribute der Datagramme in Streulisten verwaltet, um einen schnelleren Zugriff auf passende Filterregeln zu ermöglichen.

Erste vs. letzte Entscheidung: Treffen mehrere Filterregeln auf ein Datagramm zu, ist die tatsächliche Anzahl der zu evaluierenden Filterregeln noch einmal davon abhängig, ob die erste gefundene Filterregel die Entscheidung trifft oder ob alle passenden Filterregeln nacheinander angewendet werden, so daß schließlich die letzte passende Filterregel die Entscheidung fällt.

Dynamische Filterregeln: Die Anzahl der Filterregeln ist nicht statisch festgelegt, sondern ändert sich in Abhängigkeit vom Kontext.

Einsatz von Caching: Sind wenige, breitbandige Kommunikationsbeziehungen aktiv, dann kann ein Caching von denjenigen Attributen, die zuletzt die Entscheidung für ein Vernichten oder Weiterleiten von Dateneinheiten herbeigeführt haben, die Paketverarbeitung deutlich beschleunigen. Die Attribute nachfolgender Dateneinheiten werden vor der eigentlichen Filterverarbeitung mit den zwischengespeicherten (cached) Attributen verglichen. Stimmen die Attribute überein, kann eine Entscheidung sofort getroffen werden, ohne die Filterregeln anzuwenden. "Caching" ist jedoch kein Allheilmittel. Die eigentliche Filterverarbeitung wird nicht beschleunigt, sondern nur

[13] Erlaubte Kommunikation ist konform mit den lokalen Richtlinien, die teilweise durch Filterregeln auf der Packet Screen durchgesetzt werden.

[14] Die Erfahrung zeigt, daß die Benutzer ihre Kommunikationsbeziehungen schnell an eine Policy adaptieren. Sie versuchen beispielsweise nicht jeden Tag erneut, eine unerlaubte Kommunikationsbeziehung zu etablieren, sondern konzentrieren sich auf die Verwendung der richtlinienkonformen Kommunikation.

eine schnelle Entscheidung herbeigeführt, falls darauffolgende Dateneinheiten zufällig zur selben Kommunikationsbeziehung gehören.

Diese Aufzählung verdeutlicht, wie stark die Leistungsfähigkeit von der Güte der Implementierung abhängen kann.

Bei den Messungen soll immer der "worst case" betrachtet werden. Es wird daher die Anzahl der anzuwendenden Filterregeln vor dem Weiterleiten der Dateneinheit festgelegt. Dadurch sind Aussagen über die Abhängigkeit des Datagramm-Durchsatzes von der Anzahl der Filterregeln möglich. Die zu vermessenden Packet Screens müssen daher so konfiguriert werden, daß in jedem Fall die festgelegte Anzahl von Filterregeln abgearbeitet wird. Hierbei handelt es sich somit um die wichtigste Konfigurationskenngröße (vgl. Abschnitt 3.2.2.2).

Ein weiterer wichtiger Parameter, der bei den Messungen variiert wird, ist die Länge der Dateneinheit. Dateneinheiten ("Protocol Data Units", PDUs) setzen sich zusammen aus Protokollinformationen und den zu übertragenden Nutzdaten (SDU). Die Filterregeln werden ausschließlich auf den Protokollinformationen angewendet, da hier Dienst- und Adreßinformationen[15] kodiert sind, auf die sich die Filterregeln beziehen. Ein nicht unerheblicher Aufwand entsteht auf der Packet Screen jedoch auch durch das Kopieren der Dateneinheiten von der Empfangs-Schnittstelle auf die Sende-Schnittstelle.

Offensichtlich ist der Kopieraufwand bei kleinen Dateneinheiten geringer als bei großen Dateneinheiten, so daß man abhängig von der Länge einer Dateneinheit Auswirkungen auf den erreichbaren Datagramm-Durchsatz erwarten kann. Zu erwarten ist, daß mehr Dateneinheiten pro Sekunde gefiltert werden können, je kleiner die Gesamtlänge der Dateneinheit ist. Zu beachten ist, daß ein Teil des Kopieraufwands je nach Implementation vor und nach dem Filtern stattfinden kann. Eine Implementation ist um so besser, je weniger Dateneinheiten kopiert werden müssen. Idealerweise würde eine Packet Screen somit keinerlei Kopieroperationen auf zu vernichtenden Dateneinheiten anwenden.

Implementierbar sind z.Z. allerdings nur solche Workstation-basierten Packet Screens, bei denen die Dateneinheiten mindestens einmal in den Hauptspeicher des Rechners kopiert werden. Gehört die Dateneinheit zu einer erlaubten Kommunikationsbeziehung, dann wird sie erneut kopiert: vom Hauptspeicher auf die Sende-Schnittstelle. Bei der Bearbeitung von Filterregeln ist somit der "Mainstream" auch in Hinblick auf die Kopieroperationen als der "worst case" anzusehen. Das Vernichten der Dateneinheiten nach dem Filtern ist (fast) aufwandfrei.[16] Das Weiterleiten der Dateneinheiten nach dem Filtern ist aufwendig, wobei der Aufwand abhängig von der Länge der Dateneinheit ist.

[15] Es gibt neben Dienst- und Adreßinformationen weitere Attribute, auf die sich Filterregeln beziehen können. Fast immer wird in Filterregeln jedoch auf die genannten SAP-Informationen Bezug genommen.

[16] Der mit der Vernichtung von Dateneinheiten einhergehende Aufwand kann u.U. beträchtlich sein. Beispielsweise falls das Generieren und Versenden von Audit-Informationen als Reaktion auf das Vernichten von Dateneinheiten konfiguriert wurde, ist der Aufwand direkt proportional zum Audit-Aufwand. Dies ist jedoch nicht der zu optimierende Fall.

Der wichtigste Leistungsparameter einer Packet Screen ist der Datagramm-Durchsatz. Dieser Durchsatz ist abhängig von der Länge der Dateneinheiten und der Anzahl der anzuwendenden Filterregeln. In allen Versuchsaufbauten wird daher die Anzahl der Filterregeln als Konfigurationsparameter auf dem SuL und die Länge der Dateneinheiten als Lastparameter durch den Generator variiert, um den hiervon abhängigen Datagramm-Durchsatz zu bestimmen.

In allen Versuchen sind die SuL (Packet Screens) so konfiguriert, daß der "worst case" simuliert wird. Alle aufeinanderfolgenden Dateneinheiten gehören zu verschiedenen Kommunikationsbeziehungen, so daß keine Beschleunigung durch ein "caching" der Attribute stattfinden kann. Hierfür waren die in Abschnitt 3.4.6 erläuterten Konfigurationsschritte erforderlich. Um die sequentielle Abarbeitung aller Filterregeln zu garantieren, wurde eine neue Version der Filtersoftware[17] erstellt, bei der das "Caching" bereits im Quellcode abgeschaltet wurde.

3.6.2 Leistungsbewertung einer Fast Ethernet-basierten Packet Screen

Auf Grundlage der oben diskutierten, bevorzugten Leistungskenngrößen und Szenarien zur Bewertung der Leistungsfähigkeit einer Packet Screen wurden mehrere Messungen durchgeführt. Als Grundlage für den Nachweis des Engpasses wurde der Datagrammdurchsatz in Abhängigkeit von der Nachrichtenlänge in der Versuchsanordnung aus Abbildung 3.2 gemessen. Diese Messung, bei der ein Generator direkt an einen Monitor sendet, steht stellvertretend für eine nicht kontrollierte Kommunikationsbeziehung. Mehrere anschließend in der Versuchsanordnung aus Abbildung 3.3 durchgeführte Messungen sollten den Engpaß durch die Packet Screen-basierte Zugriffskontrolle aufzeigen. Bei diesen Messungen wurde nicht nur die Nachrichtenlänge variiert, sondern auch die Anzahl der anzuwendenden Filterregeln.

In Anlehnung an Abbildung 3.9 gilt daher für alle folgenden Abbildungen mit Leistungsdaten die nachfolgende Notation. Während der Meßdauer erzeugt der Generator eine *reproduzierbare Last* (vgl. Abbildung C.3). Diese Last liegt an der Schnittstelle IF_{sG} an. Man kann sich diese Last auch als eingangsseitigen Durchsatz vorstellen. Die erreichte Leistung der Packet Screen fällt als ausgangsseitiger Durchsatz an der Schnittstelle IF_{sM} an und wird vom Monitor an der Schnittstelle IF_{pM} empfangen. Da Generator und Monitor stellvertretend für Sender (*snd*) und Empfänger (*rcv*) von Kommunikationsbeziehungen stehen, kann man auch von senderseitigem und empfangsseitigem Durchsatz sprechen. Entscheidend für die Beurteilung der Leistungsfähigkeit ist der ausgangs-/empfangsseitige Durchsatz, als direktes Maß für die Leistungsfähigkeit der Packet Screen. Vernachlässigt man Paketverluste durch Übertragungsfehler und ähnliche Störungen, so gilt folgender Zusammenhang zwischen der Last an der Schnittstelle IF_{sG} in Abhängigkeit von der Anzahl der

[17] Die Software ist unter dem Namen "ipfilter" bekannt und unter folgender URL frei verfügbar: `http://www.coombs.anu.edu.au/~avalon/ip-filter.html`

konfigurierten Filterregeln (FR) und der resultierenden Rahmenlänge in Abhängigkeit von der SDU-Länge (hier x als Kurzform für Rahmenlänge(x), vgl. Formel 3.4), der beobachtbaren Leistung der Packet Screen an der Schnittstelle IF_{sM} und den aufwandsbedingten Paketverlusten auf der Packet Screen (SuL):

$$D_{snd,FR,x}(t, \Delta t) = D^{-}_{SuL,FR,x}(t, \Delta t) + D^{+}_{rcv,FR,x}(t, \Delta t) \tag{3.7}$$

Die Differenz zwischen eingangsseitigem und ausgangsseitigem Durchsatz sind die lastbedingten Paketverluste (D^{-}_{a}) auf dem SuL bzw. der Packet Screen.

Die Ergebnisse können alle in einer Abbildung zusammengefaßt werden. Es wird der mittlere Datagrammdurchsatz in Abhängigkeit von der Anzahl der Filterregeln und der Nachrichtenlänge angegeben. Der mittlere Durchsatz (Einheit Datagramme pro Sekunde [DG/s]) wird nach einer Meßdauer von 10 Sekunden über $\Delta t = 10s$ ermittelt. Die vom Monitor gemessene Filterleistung (rcv) ist also in Abhängigkeit von der Anzahl der konfigurierten Filterregeln (FR) und der SDU-Größe (Nachrichtenlänge) aufgezeichnet ($D_{rcv,FR,x}(10s, 10s)$). Generator, Monitor und SuL waren über Switches im "full duplex"-Mode (fdx) miteinander verbunden.

Man erkennt sofort, daß die Anzahl der konfigurierten Filterregeln einen erheblichen Einfluß auf den erreichbaren Durchsatz hat. Die Nachrichtenlänge ist hingegen kaum von Bedeutung. Die Leistungsschwankungen durch die Länge der Nachricht können für kleine bis mittelgroße Dateneinheiten vernachlässigt werden (16 bis 512 Byte lange Nachrichten). Ab 10 konfigurierten Filterregeln ist die Nachrichtenlänge völlig vernachlässigbar. Der Durchsatz wird hier vollständig durch die Anzahl der Filterregeln dominiert. Bei 0 und 10 konfigurierten Filterregeln fällt der erreichte Durchsatz ab einer Länge von 896 bzw. 1024 Byte mit dem theoretisch erreichbaren Durchsatz über Fast Ethernet zusammen.

3.7 Zusammenfassung

Angesichts der Tatsache, daß eine Anzahl von 100 Filterregeln kaum ausreicht, um gängige Dienste abzusichern[18], ist die Interpretation dieser Ergebnisse eindeutig: Übliche Packet Screens sind ein Engpaß bei der Kommunikation in Hochgeschwindigkeitsnetzen.

- Der Paketdurchsatz einer schnellen UNIX-Workstation reicht nicht einmal annähernd aus, um bei typischen Nachrichtenlängen von wenigen hundert Byte den möglichen theoretischen Durchsatz in einem Fast Ethernet zu erreichen.

[18] Die Anzahl der tatsächlich benötigten Filterregeln hängt nicht nur von der Anzahl der Dienste und betroffenen Rechner ab, sondern auch davon, wie gut Gruppen von Dienstkennungen und Rechner über "Wildcards" in den Filterregeln zusammengefaßt werden können.

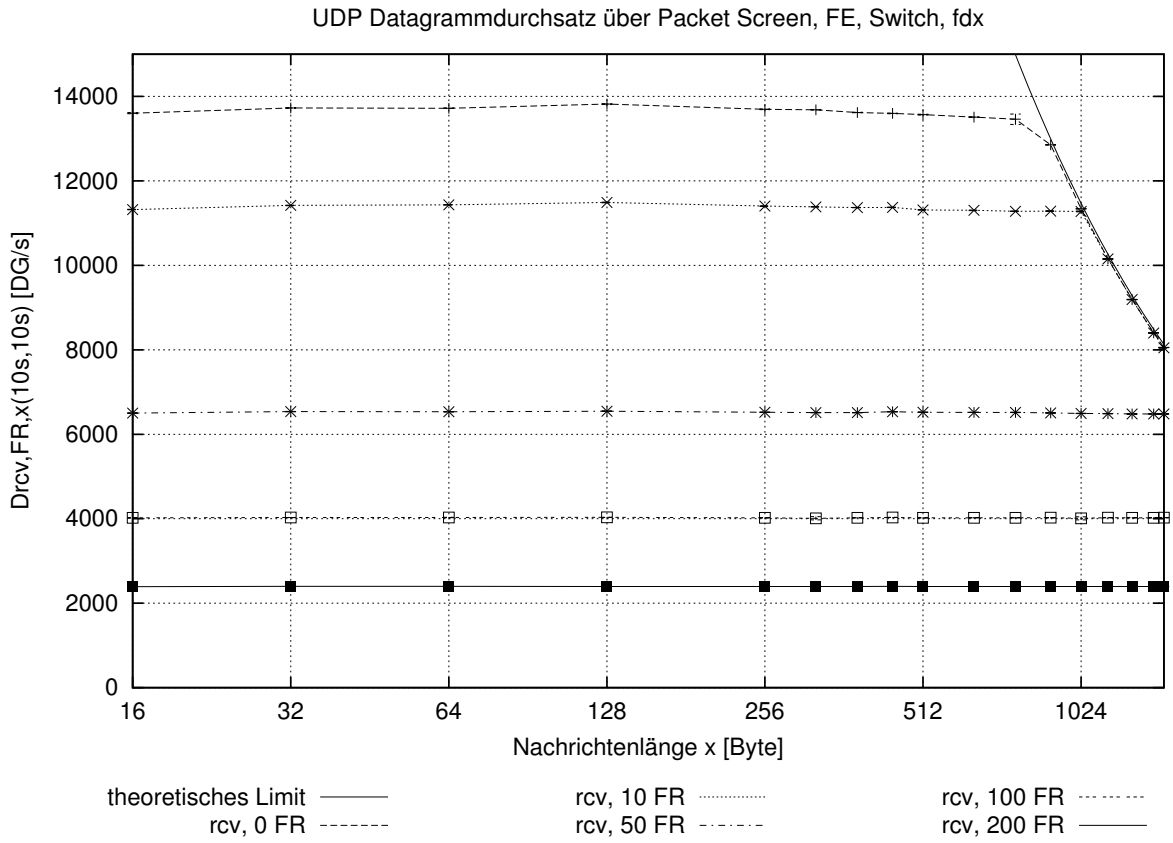


Abbildung 3.12: Engpaß Packet Screen

- Die Anzahl der konfigurierten Filterregel hat einen großen Einfluß auf den erreichbaren Paketdurchsatz. Je mehr Filterregeln konfiguriert sind, desto geringer ist der erreichbare Paketdurchsatz.[19]
- Die Auswirkungen der Nachrichtenlänge auf den Paketdurchsatz sind nur bei sehr großen Nachrichten relevant. Bei kleinen bis mittleren Nachrichtenlängen von (16 bis 512 Byte) können die Nachrichtenlängen vernachlässigt werden.

Die hohe Leistungsfähigkeit — als eine der wesentlichen Anforderungen an überlebensfähige Systeme — ist daher für konventionelle Packet Screens in Hochgeschwindigkeitsnetzen nicht mehr gegeben.

Die Migration in ein Hochgeschwindigkeitsnetz ("High Speed Network", HSN) erfordert daher eine skalierbare Lösung zur Steigerung des Paketdurchsatzes auf Packet Screens. Die aufgezeigten Engpässe müssen beseitigt werden. Darüber hinaus ist sicherzustellen,

[19] "Caching"-Mechanismen können bis zu einem gewissen Grad helfen, die Auswirkungen vieler konfigurierter Filterregeln zu mindern. Hier wird jedoch der "worst case" betrachtet.

daß die erarbeiteten Lösungen nicht bereits bei der nächsten Geschwindigkeitserhöhung konzeptionell veraltet sind, sondern mit den steigenden Übertragungsgeschwindigkeiten in HSNs gleichermaßen skalieren.

Die Anforderung lautet daher, eine Lösung zu finden, die den einfachen aber skalierbaren Aufbau von Hochgeschwindigkeits-Paketfiltern ermöglicht. Dabei ist der Paketdurchsatz die entscheidende Leistungskenngröße. Im folgenden Kapitel wird daher der Entwurf einer parallelen Packet Screen sowie die prototypische Implementierung für Fast Ethernet diskutiert.

Rückblickend ist fraglich, ob der hohe Aufwand für die Erstellung von spezialisierten Meßwerkzeugen für verteilte Systemkomponenten, wie in Kapitel 3 erfolgt, gerechtfertigt ist: Daß Firewalls einen potentieller Engpaß in verteilten Systemen darstellen, ist allgemein bekannt. Andererseits ist die systematische Belegung dieser Tatsache und das Aufzeigen der für den Engpaß ursächlich verantwortlichen Abläufe in früheren Arbeiten [21, 76, 71] noch nicht im Detail diskutiert worden. Darüber hinaus haben die Meßergebnisse auch gezeigt, daß von den beiden in Frage kommenden Parametern, die den Paketdurchsatz auf der Packet Screen beeinflussen, die Anzahl der konfigurierten Filterregeln absolut dominant gegenüber der Paketlänge ist. Diese Tatsache ließ sich nur durch die Untersuchung der Filtermechanismen am Quellcode der Filtermodule belegen, da das typischerweise verwendete "Caching" von Filterentscheidungen über den tatsächlichen Filteraufwand hinwegtäuscht. Eine oberflächliche Analyse hätte somit andere Ergebnisse geliefert.

Kapitel 4

Parallele Packet Screen für Hochgeschwindigkeitsnetze

Im vorigen Kapitel wurde gezeigt, daß die Leistungsfähigkeit typischer Packet Screens in Hochgeschwindigkeitsnetzen ("High Speed Networks", HSNs) nicht ausreicht; die Packet Screen kann zum Engpaß bei der Kommunikation zwischen verteilten Anwendungen verschiedener Sicherheitsdomänen werden. Bei der Überführung der Packet Screen in eine überlebensfähige Firewall-Komponente ist es daher das vorrangige Ziel, diesen Engpaß zu beseitigen.

Es kann angesichts des rasanten Fortschritts in der Netztechnik, mit ständig steigenden Übertragungsgeschwindigkeiten, nicht darum gehen, eine vorübergehende Lösung für heutige HSNs zu finden. Vielmehr muß versucht werden, skalierbare Lösungen und Konzepte zu finden, die nicht schon kurzfristig wieder veralten. Die zu erarbeitenden Lösungsansätze sollten im Idealfall ohne Änderungen auf kommende HSN-Technologien übertragbar sein.

In diesem Kapitel werden auf Grundlage der Meßergebnisse von Kapitel 3 in Abschnitt 4.1 Anforderungen an Packet Screens für HSNs formuliert. Es wird die Universalrechnerbasierte Parallelverarbeitung als vielversprechender Mechanismus zur Leistungssteigerung vorgestellt und als Ausgangsbasis für die folgenden Ausarbeitungen festgelegt. Die Idee, Parallelverarbeitung als Mechanismus für die Beschleunigung der Auftragsverarbeitung einzusetzen, ist nicht neu. In Abschnitt 4.2 wird daher kurz auf die gängigen Klassifikationen im Bereich der Parallelverarbeitung eingegangen. Dabei sollen vor allem die für die Packet Screens wichtigen Klassifikationen aus dem Bereich der parallelen Protokollverarbeitung diskutiert werden.

In Abschnitt 4.3 wird der Entwurf einer parallel arbeitenden Packet Screen vorgestellt. Die Entwurfsziele werden aus den in Abschnitt 4.1 formulierten Anforderungen abgeleitet. Im Vordergrund des Entwurfs steht die Auswahl geeigneter Strategien und Algorithmen für die parallele Zugriffskontrolle auf einer Packet Screen. Die Realisierung eines Prototypen einer parallelen Packet Screen wird in Abschnitt 4.4 für Fast Ethernet-Netze vorgestellt. Es wird auf die Unterschiede in der Implementierung für "shared"- und "switched" Fast

Ethernet eingegangen und erklärt, wie die im Zuge des Entwurfs diskutierten Mechanismen implementiert werden können.

In Abschnitt 4.5 wird eine Leistungsbewertung des Prototypen durchgeführt. Dies geschieht mit den in Kapitel 3 entwickelten Szenarien und Meßwerkzeugen. Die Ergebnisse können mit den Meßergebnissen des vorigen Kapitels verglichen werden, um Aussagen über die Leistungsfähigkeit der parallelen Packet Screen abzuleiten. Darüber hinaus sind die Meßergebnisse die Grundlage für ein analytisches Modell der parallelen Packet Screen, das in Abschnitt 4.6 vorgestellt wird. Mit diesem Modell können die Meßergebnisse validiert werden, und es können über die empirisch analysierten Konfigurationen hinausreichende Leistungsprognosen erstellt werden.

4.1 Leistungsanforderungen an eine Packet Screen für Hochgeschwindigkeitsnetze

Es gibt mehrere Möglichkeiten, Packet Screens für Hochgeschwindigkeitsnetze zu realisieren. Grundsätzlich kann man zwischen hoch spezialisierten Hardware-Lösungen (ASICs in dedizierten Netzkomponenten [241] wie beispielsweise Routern) oder Universalrechner-basierten Ansätzen unterscheiden. Letztere sind in jedem Fall flexibler:

- Die Anzahl der unterstützten Protokollfamilien ist leicht erweiterbar. Dies hat insbesondere Vorteile beim Entwurf neuer Hochgeschwindigkeitsprotokolle, da neben den Protokollprototypen auch Filterkomponenten auf der universellen Hardware realisiert werden können. Dies ist auf dedizierten Komponenten nicht wirtschaftlich realisierbar, da für proprietäre Protokolle aus Kostengründen keine Chipentwicklung stattfinden kann.

- Das Aufzeichnen und Auswerten von sicherheitsrelevanten Informationen (Audit) ist auf Universalrechnern wesentlich flexibler möglich. Insbesondere der Einsatz von zusätzlichen Analysewerkzeugen und die Konfigurierung von verschiedenen Detailstufen mit unterschiedlicher Informationsmenge sowie die Möglichkeit zur einfachen Komprimierung/Vorverarbeitung und zur lokalen Speicherung sprechen für den Einsatz von Universalrechnern.

- Alte Workstations, die im Produktionsbetrieb nicht mehr eingesetzt werden, können bei Universalrechner-basierten Lösungen verwendet werden, sofern diese noch zuverlässig arbeiten. Dedizierte Hardware wäre in jedem Fall neu zu beschaffen.

- Die Administration von Universalrechnern ist in der Regel besser verstanden, als die Administration proprietärer Netzkomponenten. Sie ist in der Regel auch sicherer durchzuführen, da hier alle bekannten Techniken zur sicheren Fern-Administration auf Software-Basis nachgerüstet werden können. Spezialisierte Hardware verfügt

i.d.R. auch über eine spezielle "Mensch-Maschine-Schnittstelle". Die Administration dieser Komponenten erfordert daher die Aneignung von zusätzlichem Werkzeugwissen.

- Die Filter-Software auf einem Universalrechner unterstützt eher die neuesten Funktionen. Sie ist auf Zugriffskontrolle und Audit spezialisiert. Sie ist schnell an die aktuellen Erkenntnisse aus dem Sicherheitsbereich anpaßbar, da das Einspielen von Software-Updates auf Universalrechnern ein alltäglicher Vorgang ist. Spezialisierte Hardware erlaubt nur dann eine Anpassung an neueste Erkenntnisse aus dem Sicherheitsbereich, wenn die Implementierung eines erforderlichen Algorithmus auch von der spezialisierten Hardware unterstützt wird bzw. auf ihr möglich ist. Ist die spezialisierte Hardware auf andere Aufgaben optimiert, dann sind die Funktionen zur Zugriffskontrolle und Audit häufig suboptimal implementiert. Beispielsweise sind Einbußen bei der Paketvermittlung auf Routern möglich, wenn diese zusätzlich Filterfunktionen ausführen müssen.
- Filter-Software für Universalrechner ist in der Regel um Größenordnungen billiger als die Anschaffung von spezialisierter Hardware.

Andererseits ist mit dedizierter Hardware eine deutlich höhere Verarbeitungsgeschwindigkeit möglich. Dieser vermeintliche Vorteil ist jedoch kaum als skalierbar zu bezeichnen. Ist der Durchsatz der Hardware nicht mehr hoch genug, um die Dateneinheiten der nächsten, höheren Geschwindingkeitsstufe zu verarbeiten, muß in jedem Fall neue spezialisierte und somit teure Hardware beschafft werden (s.o.).

Geht man davon aus, daß universelle Hardware eingesetzt wird, so ist unmittelbar einsichtig, daß eine geschwindigkeitsoptimierte Reimplementierung der Filterfunktionen alleine keine dauerhafte Lösung darstellt. Im folgenden Abschnitt wird daher die Parallelverarbeitung als eine skalierbare Lösung für die Protokollverarbeitung und Analyse in Hochgeschwindigkeitsnetzen (HSN) vorgestellt.

Die Anforderungen an eine skalierbare, leistungsfähige Packet Screen lassen sich daher in zwei wesentlichen Punkten zusammenfassen:

1. Die Paket Screen muß flexibel konfigurierbar sein: Neue Protokollfamilien, die Selektion der Audit-Daten sowie die bei Packet Screens bekannten Filterregeln müssen jederzeit konfigurierbar bzw. anpaßbar sein.
2. Die Packet Screen muß einfach an höhere Übertragungsgeschwindigkeiten und eine zunehmende Anzahl an Filterregeln anpaßbar sein.

Die beiden oben genannten Anforderungen stellen die Minimalanforderung dar. Die erste Anforderung ließe sich beliebig verfeinern bzw. detaillierter ausführen. Aber auch in diesem Fall muß auf die eingangs gemachte Annahme verwiesen werden, daß die Packet Screen den funktionalen Anforderungen genügt. Dieser Funktionsumfang sollte durch die Überführung

in ein überlebensfähiges System nicht eingeschränkt werden, sondern muß im wesentlichen erhalten bleiben. Die wesentlichen funktionalen Anforderungen an alle Packet Screens sind daher im ersten Punkt aufgeführt.

Der zweite Punkt berücksichtigt die in der Einleitung zu diesem Kapitel diskutierte Anforderung, daß das zu erarbeitende Konzept skalierbar sein muß. Skalierbarkeit bedeutet bei Packet Screens die Möglichkeit zur einfachen Anpassung an höhere Übertragungsgeschwindigkeiten und an einen höheren Bearbeitungsaufwand durch eine steigende Anzahl an pro Paket abzuarbeitenden Filterregeln.

Zu diesem Zweck sollte Universalrechner-basierte Parallelverarbeitung eingesetzt werden, da dedizierte Hardware nicht flexibel genug ist (s.o.) und die absehbaren Zuwächse in den Übertragungsgeschwindigkeiten der Rechnernetze die Rechenleistung auch neuer von Neumann Architekturen weit übersteigen. Bereits heute bringen Verdoppelungen der CPU-Geschwindigkeiten kaum noch Gewinne bei der Datenverarbeitung, da es meist nicht zu einer entsprechenden Anpassung der Speicherzugriffszeiten und des Bustaktes kommt; folglich sind Universalrechner-basierte Ansätze ohne gleichzeitige Parallelverarbeitung nicht geeignet. Beispielsweise kann ein 32-Bit breiter PCI-Bus bei der standardgemäßen Taktgeschwindigkeit von 33 MHz maximal 132 MByte/s transportieren. Im praktischen Einsatz sind ca. 100 MByte/s nachweisbar [10]. Das entspricht nicht einmal der Geschwindigkeit eines Gigabit-Ethernets.

4.2 Leistungssteigerung durch parallele Protokollverarbeitung

In der vorangegangenen Diskussion wurde argumentiert, daß Universalrechner-basierte Parallelverarbeitung zur Leistungssteigerung bei der Protokollverarbeitung auf der Packet Screen eingesetzt werden soll, um die in Kapitel 3 aufgezeigten Engpässe zu beseitigen. Da Parallelverarbeitung seit langem als Mechanismus zur Beschleunigung der Auftragsverarbeitung bekannt ist, gibt es gängige Klassifikationen und Begriffe, die in Abschnitt 4.2.1 kurz zusammengefaßt werden sollen. Neben den dort vorgestellten Klassifikationen bezüglich der Rechnerarchitektur und der Prozeßzuordnung sind die Erkenntnisse aus dem Bereich der parallelen Protokollverarbeitung für die Beschleunigung von Packet Screens von besonderer Bedeutung und werden daher in Abschnitt 4.2.2 genauer diskutiert.

4.2.1 Gängige Klassifikationen im Bereich der Parallelverarbeitung

Parallelverarbeitung gibt es in verschiedenartigen Ausprägungen, die zu einer Reihe von Klassifikationen geführt haben. Diese Klassifikationen beziehen sich:

- auf die Architektur der Rechner, die die parallele Auftragsverarbeitung durchführen (vgl. Abschnitt 4.2.1.1), und
- auf die Art und Weise, wie Aufträge auf die zur Verfügung stehenden "Bedienstationen" (Prozessoren) abgebildet werden (vgl. Abschnitt 4.2.1.2).

4.2.1.1 Klassifikation bezüglich Rechnerarchitektur

Die bekannteste Klassifikation bezüglich der Rechnerarchitektur (nach Flynn) wird beispielsweise in [114, S.637] dargestellt. Alle Architekturen werden in eine von vier Klassen eingeteilt, die nach zwei Freiheitsgraden unterteilt sind, Kontrollpfade und Datenpfade.

Eine etwas detailliertere Klassifikation wird in [214] gegeben. Sie erweitert die oben genannte Klassifikation um zusätzliche Angaben zur Anzahl der "Instruction Processors" (IPs) und "Data Processors" (DPs), der Art der Verknüpfung zwischen IPs und DPs und der Art der Verknüpfung der Prozessortypen mit ihrem jeweiligen Speicher ("Instruction Memory", IM und "Data Memory", DM).

Durch diese feinere Einteilung können beispielsweise verschiedene "Multiple Instruction Streams, Multiple Data Streams" (MIMD)-Architekturen weiter unterteilt werden in "lose-" oder "eng-" gekoppelte Systeme sowie in von Neumann Rechner und Datenflußrechner.

4.2.1.2 Klassifikation bezüglich Zuordnung

Eine weitere Klassifikation (vgl. [112]) ist für die Beschreibung der Zuordnung von Prozessen zu Prozessoren erforderlich, um die dynamischen Aspekte bei der Parallelverarbeitung zu beschreiben. Die folgende Darstellung orientiert sich an [112, 22].

Ein Programm realisiert einen oder mehrere Algorithmen. Das Programm muß zur Berechnung mindestens auf einem Prozessor ausgeführt werden. Zu den Problemstellungen der sequentiellen Programmierung, wie beispielsweise die prozedurale Dekomposition der Algorithmen in überschaubare kleinere Teilalgorithmen, kommen bei der parallelen Programmierung die beiden folgenden Problemkomplexe hinzu:

- Aufspüren der *inhärenten Parallelität* eines Algorithmus
- Abbilden der parallelen Programme auf die Prozessoren

Die inhärente Parallelität eines Algorithmus besteht aus denjenigen parallel ausführbaren Teilalgorithmen, die nur aufgrund der bisher dominierenden sequentiellen Programmierung nacheinander ausgeführt wurden. Diese inhärente Parallelität muß für eine feingranulare Zuordnung "wiederentdeckt" werden.

Realisiert man die inhärente Parallelität eines Algorithmus als nebenläufige Prozesse, erhält man parallele Programme, die aus einer Menge kooperierender[1] Prozesse bestehen. Der zweite Problemkomplex ist dann das Abbilden (Zuordnen, engl. "mapping") der parallelen Programme bzw. der Prozesse auf die Prozessoren des Parallelrechners.

Formale Präzisierung der Zuordnung Einige Begriffe der Zuordnungsproblematik lassen sich formal darstellen:

$$\begin{aligned} P &\quad \text{Menge der Prozessoren } (|P| = n); \\ A := \{A_1, A_2, \cdots A_{|A|}\} &\quad \text{Last, bestehend aus einer} \\ &\quad \text{Menge paralleler Programme;} \\ T_i &\quad \text{Menge der Prozesse des Programms } A_i \\ &\quad (i = 1, \cdots, |A|). \end{aligned} \tag{4.1}$$

Als *Programmzuordnung* bezeichnet man die Abbildung von der Menge der Programme (A) auf die Potenzmenge der Prozessoren ($\wp(P)$), da durchaus zwei oder mehr Programme einem Prozessor zugeordnet werden können.

$$\text{Programmzuordnung:} \quad \varphi : A \to \wp(P) \tag{4.2}$$

Will man eine disjunkte Abbildung der Programme auf die Prozessoren, so daß ein Prozessor einem Programm exklusiv zur Verfügung steht, kann man eine disjunkte Programmzuordnung durch folgende Abbildung erreichen:

$$\text{disjunkte Programmzuordnung:} \quad \varphi : A \to \wp(P), \quad \forall i \neq k : \quad \wp(A_i) \cap \wp(A_k) = \emptyset \tag{4.3}$$

Die Programmzuordnung ist demnach entweder *disjunkt* oder *überlappend*.

Die Prozessoren sind durch ihre Verbindungen untereinander zu einer bestimmten Topologie angeordnet. Die Topologie läßt sich graphentheoretisch als Menge von Knoten (Prozessoren) und Menge von gerichteten oder ungerichteten Kanten (Verbindungen) darstellen. Eine Zuordnung eines Programms A_i heißt zusammenhängend, wenn der durch die Zuordnung entstehende Untergraph (Menge der belegten Knoten und deren Kanten) zusammenhängend ist. Eine nicht zusammenhängende Zuordnung bezeichnet man als *gestreut*.

Bei einer disjunkten Partitionierung kann man darüber hinaus von der Zuordnung zu bestimmten Prozessoren abstrahieren und ausschließlich festlegen, wieviele Prozessoren ein Programm belegen soll. Man bezeichnet diese Abbildung von der Menge der Programme auf die Prozessorzahlen ($\{1, \cdots, n\}$) als *quantitative Partitionierung*:

$$\text{quantitative Partitionierung:} \quad \chi : A \to \{1, \cdots, n\} \text{ mit } \sum_{i=1}^{|A|} \chi(A_i) \leq n \tag{4.4}$$

[1] Die Kooperation der Prozesse besteht darin, daß sich zwei oder mehr Prozesse synchronisieren müssen, um beispielsweise Daten auszutauschen.

Dabei können an alle Programme (A) nur so viele Prozessoren vergeben werden, wie tatsächlich vorhanden sind (hier n). Dies wird durch den Term mit der Summe über alle Programme in Formel 4.4 sichergestellt.

Bei der Prozeßzuordnung werden nicht mehr die Programme den Prozessoren zugeordnet, sondern die Menge aller Prozesse aller Programme (T) wird auf die Prozessoren abgebildet:

$$\text{Prozeßzuordnung:} \quad \pi : T \rightarrow P \tag{4.5}$$

Sind mindestens so viele Prozessoren vorhanden wie Prozesse ($|T| \leq n$), kann man jedem Prozeß einen Prozessor zuteilen. Eine solche Abbildung heiß *injektiv*:

$$\forall i, k \in \{1, \cdots, |T|\}, i \neq k : \qquad \pi(i) \neq \pi(k) \tag{4.6}$$

Werden zwei oder mehr Prozesse auf einen Prozessor abgebildet, heißt die Zuordnung *kontraktiv*. Will man stets eine injektive Zuordnung von Prozessen zu Prozessoren erreichen, muß man bei $|T| \geq n$ Prozessen eine *unvollständige Zuordnung* vornehmen, bei der eine Teilmenge der Prozesse nicht zugeordnet wird. Bei Kontraktion oder wenn gilt $|T| \leq n$, kann man eine *vollständige Zuordnung* erreichen.

Aus der Sicht der Prozeßverwaltung eines Parallelrechners kann sich eine Zuordnung mit der Zeit ändern. Das hat hauptsächlich zwei Gründe:

1. bereits aktive Prozesse terminieren und geben einen Prozessor frei
2. neue Programme sollen abgearbeitet werden bzw. neue Prozesse werden dynamisch generiert und müssen auf (freie) Prozessoren abgebildet werden.

Die Gesamtmenge der zu einem Zeitpunkt t zu verwaltenden Prozesse (T_t) setzt sich zusammen aus den aktiven Prozessen, die bereits zum Zeitpunkt t auf Prozessoren plaziert sind und den wartenden Prozessen (T_t^w), die bis zum Zeitpunkt t neu hinzugekommen sind. Bezeichnet man die Menge der zum Zeitpunkt t nicht belegten Prozessoren mit P_t^f, so kann man den dynamischen Verlauf der Belegung als *konservativ* oder als *aggressiv* bezeichnen:

$$\begin{aligned} \pi_{konservativ} : &\quad T_t^w \rightarrow P_t^f \\ \pi_{aggressiv} : &\quad T_t \rightarrow P \end{aligned} \tag{4.7}$$

Der Unterschied besteht darin, daß bei der aggressiven Zuordnung Prozesse von den Prozessoren verdrängt werden können. Diese können entweder einem anderen Prozessor zugeordnet werden (Migration) oder eine bestimmte Zeit lang warten, bis sie wieder zugeordnet werden. Bei der konservativen Zuordnung werden nur solche Prozessoren belegt, die ihre vorherigen Aufträge ausgeführt haben.

4.2.2 Klassifikation bezüglich Protokollverarbeitung

Die oben dargestellten Klassifikationen sind sehr allgemein gehalten und decken somit jede Art von (Parallel-)Rechnerarchitektur und Zuordnungsproblem ab. Eine Analyse der

einem Algorithmus inhärenten Parallelität legt zusätzlich eine bestimmte Strukturierung von Aufträgen in kleinere, parallel-ausführbare Teilaufträge nahe. Für die weiteren Untersuchungen am Beispiel einer Netzkomponente sind insbesondere die für die Protokollverarbeitung relevanten Klassifikationen interessant.

Legt man sich auf bestimmte zu verarbeitende Auftragsklassen bzw. Algorithmen fest, sind weitere Klassifikationen sinnvoll, die die Art der Parallelverarbeitung problembezogener darstellen. Eine Packet Screen ist eine hochspezialisierte Netzwerk-Komponente, die Protokolldateneinheiten (PDUs) verarbeitet. Es folgt daher in diesem Abschnitt eine Klassifikation der Parallelverarbeitung, die sich auf die Protokollverarbeitung bezieht. Sie beruht auf [97, 240, 245] und wurde in [22] vorgestellt.

Eine mögliche Klassifikation der parallelen Protokollverarbeitung geht von der hierarchischen Schichtenstruktur des 'Open Systems Interconnection Reference Model' (OSIRM) aus. Dieses Referenzmodell beschreibt eine für Protokolle typische hierarchische Anordnung, die Grundlage für die verschiedenen Ansätze zur parallelen Protokollverarbeitung ist.

Es lassen sich Methoden zur parallelen Protokollverarbeitung aus der

- statischen Struktur und der
- dynamischen Struktur

im Referenzmodell ableiten (vgl. Abbildung 4.1). Die statische Struktur des Referenz-

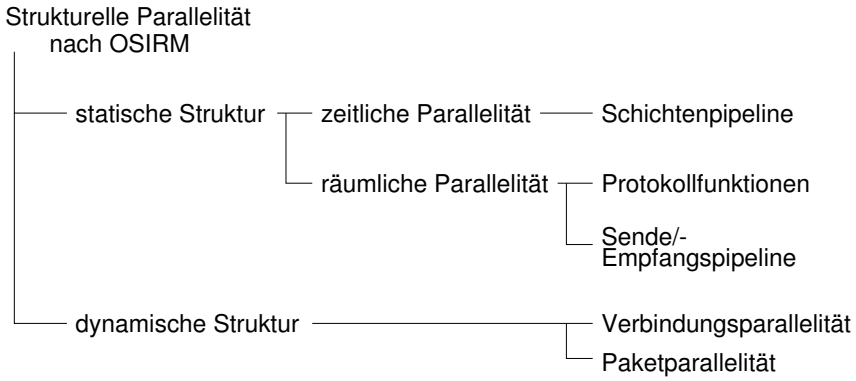


Abbildung 4.1: Strukturelle Parallelität nach OSIRM

modells (vgl. [134, S. 36ff]) beschreibt die Hierarchisierung der Protokolle, die Dienste, die die Protokolle erbringen, sowie die Struktur und Adressierung der Instanzen, die das Potential der Protokollverarbeitung in einem System bilden. Die sich hieraus ableitbaren Methoden der Parallelverarbeitung unterscheiden sich in der funktionalen Dekomposition der Aufgaben bei der Protokollverarbeitung.

Die dynamische Struktur des Referenzmodells (vgl. [134, S.50ff]) beschreibt das Verhalten der Kommunikationspartner bei der Datenübertragung, z.B. das Aufbauen von Verbindungen oder das verbindungslose Übertragen von Daten. Die hierunter subklassifizierten Formen der Parallelverarbeitung gehen von den versendeten Dateneinheiten (Paketen) aus. Sie unterscheiden sich in der Dekomposition der Daten(einheiten), die in einem Endsystem eintreffen bzw. von dort versendet werden.

Zeitliche Parallelität (Schichtenpipeline) Die *zeitliche Parallelität* ergibt sich aus der hierarchischen Struktur des Protokollstapels. Die Instanzen der Protokolle bilden eine Protokollpipeline. Dateneinheiten durchlaufen nacheinander (zeitliche versetzt) die einzelnen Stufen dieser Pipeline. Auf den verschiedenen Schichten[2] arbeiten die Protokollinstanzen parallel zueinander. Wenn eine Protokollinstanz eine Dateneinheit verarbeitet hat, wird diese der folgenden Schicht übergeben. Dort beginnt die Protokollinstanz mit der weiteren Verarbeitung, während die Protokollinstanz der vorigen Schicht eine weitere Dateneinheit bearbeiten kann (vgl. Abbildung 4.2).

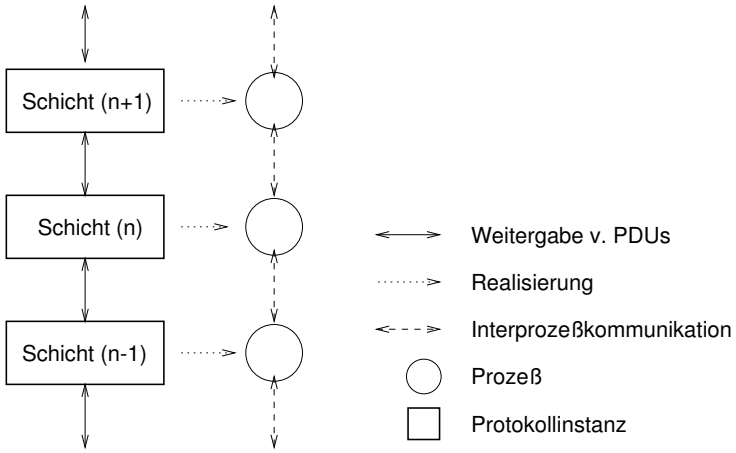


Abbildung 4.2: Zeitliche Parallelität (Schichtenpipeline)

Räumliche Parallelität (Sende/-Empfangspipeline) Die räumliche Parallelität entsteht durch die konzeptionelle Unterteilung jeder Protokollinstanz in ihre Sende- und Empfangsfunktionen. Diese (Gruppen von) Funktionen können weitgehend unabhängig voneinander — und somit parallel — ausgeführt werden (vgl. Abbildung 4.3). Während eine Dateneinheit empfangen wird, kann häufig unabhängig von diesem Empfang eine Dateneinheit versendet werden. Diese Unterteilung in weitgehend unabhängige Sende- und Empfangsprozesse verfeinert die oben diskutierte Schichtenpipeline. Die Granularität läßt

[2] Bildlich gesprochen, sind die Schichten *vertikal* übereinander angeordnet.

sich gegenüber einer einfachen Schichtenpipeline verdoppeln, da bei einer Realisierung pro Pipelinestufe nun zwei Prozesse parallel arbeiten können.

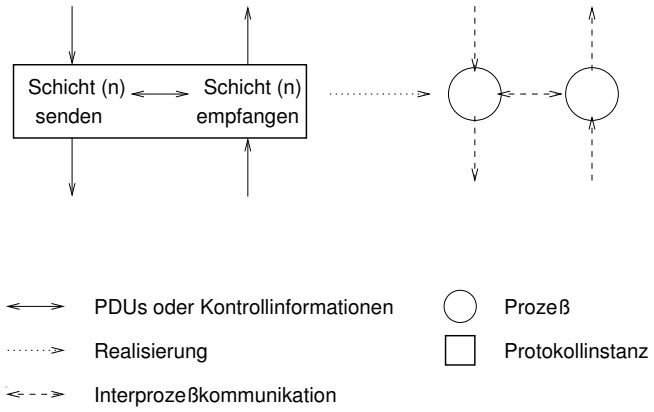


Abbildung 4.3: Räumliche Parallelität (Sende/-Empfangspipeline)

Räumliche Parallelität (Protokollfunktionen) Will man eine feinere funktionale Dekomposition[3] erreichen als bei der Kombination aus horizontaler und vertikaler Parallelität[4], muß man die einzelnen Protokollinstanzen weiter unterteilen. Diesem Konzept liegt die Annahme zugrunde, daß in Protokollen verschiedene Funktionen unabhängig voneinander ausgeführt werden können. Diese Funktionen sind protokollspezifisch und müssen durch eine genaue Analyse des Protokolls gefunden werden. Dazu muß die inhärente Parallelität eines Protokolls aufgedeckt werden, um die Funktionen so umzuordnen oder auch zu verschmelzen, daß eine möglichst asynchrone Bearbeitung dieser Funktionen möglich wird (vgl. [246, S. 56f], Abbildung 4.4).

[3] Je feiner die funktionale Dekomposition ist, desto höher ist der erreichbare Grad an möglicher Parallelverarbeitung.

[4] Angenommen in einem Endsystem wird eine Protokollhierarchie aus n Protokollen eingesetzt, dann gibt es nach der funktionalen Dekomposition in Schichten- und Sende/Empfangspipeline genau $2n$ parallel ausführbare Funktionen, die als eigenständige Prozesse realisiert werden können.

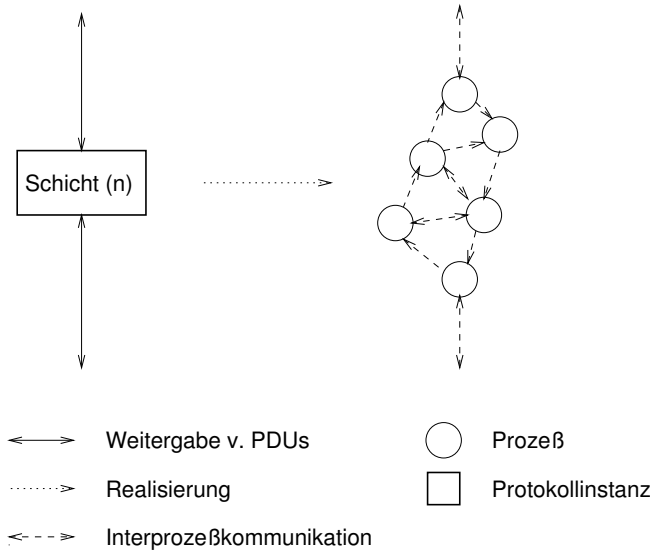


Abbildung 4.4: Räumliche Parallelität (Protokollfunktionen) (vgl. [246, S. 52ff])

Verbindungsparallelität Die Verbindungsparallelität entsteht durch die verschiedenen Verbindungen, die bei verbindungsorientierter Kommunikation in einem Endsystem aufgebaut werden können. Die Protokollverarbeitung kann i.d.R. für verschiedene Verbindungen völlig parallel (ohne Synchronisation zwischen Protokollinstanzen im selben Endsystem) erfolgen. Die Verbindungen werden den Prozessen zugeordnet. Offensichtlich läßt sich der Grad der erreichbaren Parallelität nicht im voraus angeben. Vielmehr entspricht dieser exakt der Anzahl der zu einem Zeitpunkt aktiven Verbindungen in einem Endsystem. Die Granularität der Parallelität kann sich also dynamisch verändern (vgl. Abbildung 4.5 für eine schematische Darstellung).

Paketparallelität Bei der Paketparallelität versucht man die Granularität gegenüber der Verbindungsparallelität zu erhöhen, indem man nicht mehr Verbindungen als kleinste, unabhängige Einheiten ansieht, sondern die Dateneinheiten (Pakete) aller Verbindungen als voneinander unabhängig betrachtet. Jedes Paket kann konzeptionell unabhängig von allen anderen verarbeitet werden (vgl. [97, S. 221ff], und Abbildung 4.6). Eine Dateneinheit kann auf einem System einem Prozeß zugeordnet werden, der dann die gesamte Protokollverarbeitung für diese Dateneinheit im System durchführt. Bildlich gesprochen "trägt" der Prozeß die Dateneinheit durch den Protokollstapel. Auch in diesem Fall ist die Granularität der Parallelverarbeitung nicht statisch vorgegeben, sondern sie kann sich dynamisch mit der Anzahl der gleichzeitig vom System verarbeiteten Pakete ändern.

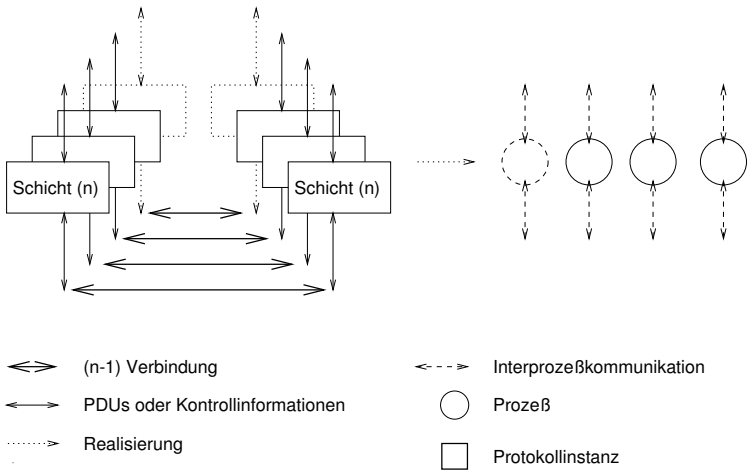


Abbildung 4.5: Verbindungsparallelität

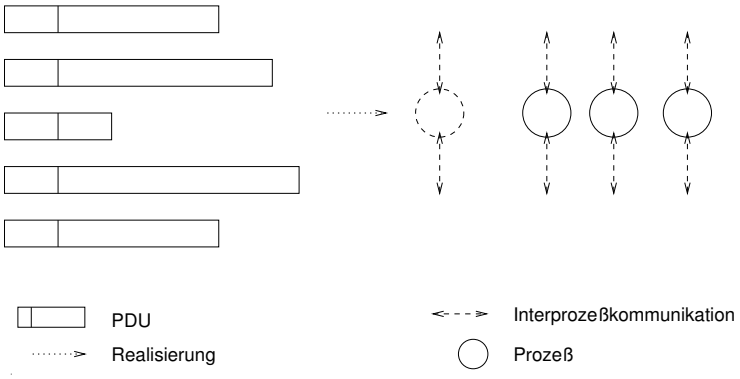


Abbildung 4.6: Paketparallelität

Bei dem folgenden Entwurf einer parallelen Packet Screen kann prinzipiell aus den eben diskutierten Alternativen gewählt werden. Hierzu sind die Vor- und Nachteile der jeweiligen Parallelisierung abzuwägen.

4.3 Entwurf einer parallelen Packet Screen

Die parallele Protokollverarbeitung ist seit langem als wichtiger Mechanismus bekannt, um Benutzern die stetig steigenden Datenübertragungsgeschwindigkeiten auf der Anwendungsebene zur Verfügung zu stellen. Der Einsatz der Parallelverarbeitung bei Packet Screens wurde bereits in [77] vorgestellt und soll hier ausführlich dargestellt werden.

Das Hauptproblem besteht in der Auswahl eines geeigneten Algorithmus für die Parallelverarbeitung. Neben der Frage, wie die anfallende Rechenlast auf die parallel arbeitenden Prozessoren aufgeteilt werden soll, müssen folgende Kriterien berücksichtigt werden:

- Wie gut skaliert der Algorithmus?
  Grundsätzlich möchte man eine möglichst feine Granularität bei der Parallelverarbeitung erreichen. Es sollte also möglich sein, mit vielen Prozessoren parallel zu arbeiten. Diese Forderung läßt sich jedoch nicht "ad hoc" erfüllen, da die Granularität sowohl vom Problem (hier die Anwendung der Paketfilter auf die Dateneinheiten) als auch vom Parallelisierungskonzept abhängen kann.

- Wieviel Overhead entsteht durch die Parallelisierung?
  Die Aufteilung der Arbeitslast auf verschiedene Prozessoren kann selbst sehr aufwendig sein. Da die dafür notwendigen Berechnungen und die resultierende Rechenzeit nicht der eigentlichen Aufgabe dienen, ist dieser zusätzliche Mehraufwand (Overhead) unerwünscht und sollte möglichst gering gehalten werden. Overhead kann auch durch aufwendige Verfahren bei der Programmzuordnung (vgl. Abschnitt 4.2.1.2) entstehen. Insbesondere wenn versucht wird, immer eine möglichst optimale Zuordnung zu finden, muß möglicherweise zusätzlicher Overhead für die Prozeßmigration beachtet werden.

- Wie aufwendig ist die Implementierung des Algorithmus?
  Der Aufwand für die Implementierung des Algorithmus muß vertretbar sein. Möglichst geringer Aufwand sollte große Durchsatzverbesserungen ergeben.

- Wie aufwendig ist die Konfigurierung des Algorithmus?
  Die Konfigurierung von Paket-Filtern ist bereits anspruchsvoll und somit aufwendig. Die Konfigurierung sollte daher durch die Parallelisierung möglichst nicht noch aufwendiger werden. Andererseits werden Packet Screens nicht ständig neu konfiguriert, so daß hier ein selten anfallender, moderater Mehraufwand durchaus vertretbar ist.

- Wie zuverlässig ist die resultierende Architektur?
  Wenn durch die Parallelverarbeitung die Ausfallwahrscheinlichkeit der Packet Screen steigt, ist sowohl die Zuverlässigkeit als auch die Sicherheit negativ beeinträchtigt. Ideal wäre demnach ein Algorithmus, der die Ausfallsicherheit zusätzlich zur Leistung erhöht. Dies könnte beispielsweise durch einen Algorithmus geschehen, der sich dynamisch an die verfügbaren[5] Prozessoren anpaßt.

- Welche einschränkenden Annahmen werden über die Netztopologie gemacht?
  Der Algorithmus sollte unabhängig von einer bestimmten Netztopologie sein. Insbesondere sollten keine Annahmen über ein bestimmtes Zugriffsverfahren oder über

[5] Verfügbar bedeutet hier: "noch nicht ausgefallen". Voraussetzung für die dynamische Anpassung ist selbstverständlich das schnelle (maschinelle) Erkennen eines Ausfalls.

ein zu verwendendes Netzprotokoll (Ethernet, Token Ring, ATM,...,) gemacht werden. Idealerweise ist ein Algorithmus unabhängig von dem jeweiligen Zugriffsverfahren und der Netztopologie, kann aber die besonderen Eigenschaften von bestimmten Protokollen und Zugriffsverfahren nutzbringend einsetzen.

4.3.1 Statische und dynamische Parallelität

Die Wahl des Algorithmus sollte in Hinblick auf die weiter oben vorgestellten, allgemeinen Konzepte zur parallelen Protokollverarbeitung durchgeführt werden. Da die Wahl eines Parallelisierungskonzeptes den weiteren Entwurf maßgeblich bestimmt, muß sorgfältig abgewogen werden, welches Parallelisierungskonzept für die Parallelverarbeitung auf Packet Screens am besten geeignet ist.

Geht man von der Grobeinteilung in die Kategorien (vgl. Abschnitt 4.2.2):

- statische Parallelität und
- dynamische Parallelität

aus, so fällt schnell der Hauptnachteil aller statischer Methoden auf. Die statischen Methoden verfügen konzeptionell über eine beschränkte Granularität und eine Unausgewogenheit der ableitbaren Pipelinestufen. Nach dem ISO/OSI-Schichtenmodell ergibt sich durch die sieben Protokollschichten eine siebenstufige vertikale Pipeline, die zusammen mit der horizontalen Parallelität (trennen der Sende- und Empfangsfunktionen) maximal 14 unabhängige Pipelinestufen zuläßt. Außerdem ist der Rechenaufwand innerhalb der Pipelinestufen sehr unausgewogen, da auf höheren Protokollschichten deutlich komplexere Aufgaben ausgeführt werden als auf niedrigeren Protokollebenen. Dadurch ergibt sich eine Beschränkung des erreichbaren Gewinns durch die komplexeste Pipelinestufe. Der Hauptnachteil der Protokollfunktionen ist neben der dargestellten Unausgewogenheit der Aufwand für die Analyse aller eingesetzten Protokolle auf inhärente Parallelität. Es müßte für jedes Protokoll, das von der Packet Screen im Verlauf der Zugriffskontrolle verarbeitet werden muß, eine aufwendige Analyse der inhärenten Parallelität durchgeführt werden. Darüber hinaus müßte vor der Kenntnis des tatsächlichen Kommunikationsaufkommens ein "Mapping" (vgl. Abschnitt 4.2.1.2) von Protokollfunktionen auf Prozessoren durchgeführt werden. Will man dies vermeiden, müßte zusätzlicher Overhead durch Prozeßmigrationen einkalkuliert werden. Ein Prozeß, der eine Protokollfunktion bearbeitet, könnte dann bei Bedarf auf einen weniger belasteten Prozessor wechseln. Dies würde jedoch weitere Synchronisation, beispielsweise für den Austausch von Auslastungsinformationen und für das Migrations-Management, erfordern.

Eine ausführliche Darstellung der Konzepte mit den jeweiligen Vor- und Nachteilen ist in [22, 245, 240] enthalten. Zusammenfassend läßt sich jedoch sagen, daß die statischen Parallelisierungskonzepte zu grob und unausgewogen oder sehr aufwendig zu implementieren sind. Bei allen statischen Methoden stellt sich darüber hinaus die Frage, wie die

erforderliche Prozeßkommunikation (beispielsweise für die Übergabe einer Dateneinheit an die nachfolgende Pipelinestufe) basierend auf Universalrechnern erfolgen soll. Beim Einsatz von Rechnern, die über mehrere Prozessoren verfügen, könnte die Übergabe im Speicher erfolgen. Bei Rechnern, die über jeweils nur einen Prozessor verfügen oder bei Mischformen aus Ein- und Mehrprozessormaschinen, müßte diese Übergabe durch zusätzliche Kommunikation erfolgen. Angesichts des Ziels, gerade die Protokollverarbeitung zu beschleunigen, sollte möglichst auf zusätzliche Kommunikation verzichtet werden.

Bei den dynamischen Methoden wird versucht, die Parallelität der Kommunikationsbeziehungen auszunutzen. Die dynamischen Methoden sind:

- Verbindungsparallelität
  Bei der Verbindungsparallelität werden alle gleichzeitig existierenden Verbindungen als voneinander unabhängig und somit als parallel betrachtet. Die Dynamik ergibt sich aus der zeitlichen Schwankung der Granularität, die direkt proportional zur Anzahl der gleichzeitig aktiven Verbindungen ist.

- Paketparallelität
  Bei der Paketparallelität wird die Eigenschaft der Kommunikationsprotokolle, Daten in voneinander unabhängigen Einheiten (im folgenden Pakete) zu transportieren, ausgenutzt, um die gleichzeitig in einem Kommunikationssystem vorhandenen Pakete parallel zu verarbeiten. Die Granularität ergibt sich ebenfalls dynamisch, da sowohl alle Pakete verschiedener zeitgleicher Verbindungen voneinander unabhängig verarbeitet werden als auch alle Pakete derselben Verbindung.

Der Vorteil der dynamischen Methoden ist, daß sie grundsätzlich eine beliebig feine Granularität erlauben. Ein weiterer Vorteil gegenüber den statischen Methoden ist, daß keinerlei zusätzliche Kommunikation zwischen Prozessen auf verschiedenen Prozessoren erforderlich ist. Bei einer Zuordnung einer Verbindung zu einem Prozessor kann die für die Übergabe zwischen verschiedenen Protokollschichten notwendige Kommunikation über den Hauptspeicher des Universalrechners erfolgen. Es ist keine zusätzliche Kommunikation über das Netz erforderlich.

Bei der Verbindungsparallelität entsteht allerdings dadurch ein Nachteil, daß bei einer Zuordnung von wenigen parallelen Verbindungen zu Prozessoren einige der Prozessoren nur gering ausgelastet sind (vgl. Abschnitt 5.2). Dies ist ein inhärenter Nachteil, der durch die Paketparallelität vermieden wird. Hier werden wenige oder auch viele Verbindungen gleichermaßen gut unterstützt. Durch die Zuordnung von Paketen zu Prozessoren ergibt sich schon bei einer Verbindung ein Beschleunigungspotential. Dynamisch hinzukommende Verbindungen erhöhen die Last, brauchen aber nicht auf freie Prozessoren zu warten, da

auch die Pakete der neuen Verbindungen dynamisch[6] von den vorhandenen Prozessoren verarbeitet werden.

Das Konzept der Paketparallelität wurde zuerst in [97] vorgestellt und bereits erfolgreich für die Parallelisierung von Kommunikationssubsystemen eingesetzt (vgl. [97, 22]).

4.3.2 Parallele Packet Screen auf Basis der Paketparallelität

Gerade beim Einsatz der Paketparallelität zum Beschleunigen von Packet Screens ergeben sich die folgenden Vorteile gegenüber anderen Parallelisierungsmöglichkeiten:

Konzeptionelle Übereinstimmung: Packet Screens führen ihre Zugriffskontrolle auf Basis der Paketinformationen (Paket-Attribute) und der darauf definierten Filterregeln durch. Das Paket wird bei den Packet Screens daher in Übereinstimmung mit dem Konzept der Paketparallelität als unabhängige Informationseinheit angesehen, die somit parallel zu anderen Paketen verarbeitet werden kann.

Keine Notwendigkeit für Kontextinformationen: Die bei der Verbindungsparallelität notwendigen Kontextinformationen über den Zustand der jeweiligen Verbindungen werden bei der Paketparallelität nicht benötigt. Das vereinfacht die Verteilung der Dateneinheiten entscheidend, da somit eine beliebig einfache Verteilungsstrategie eingesetzt werden kann (beispielsweise eine zufällige Verteilung). Falls dennoch Kontextinformationen bezüglich der zu überwachenden Kommunikationsbeziehungen für anspruchsvolle Filteraufgaben ("statefull packet inspection") benötigt werden, können die in Abschnitt 5 diskutierten Mechanismen eingesetzt werden.

Keine Notwendigkeit für aufwendige Zuordnungen: Alle parallelen Prozessoren (Packet Screens) können den gleichen Filteralgorithmus ausführen. Bei der statischen Parallelität wäre eine Zuordnung der Filterfunktionen zu dedizierten Prozessoren erforderlich. Diese Zuordnung entfällt bei der Paketparallelität. Der Aufwand für die Parallelisierung ist bei der Paketparallelität daher sehr gering.

Sehr gute Skalierbarkeit: Da alle Prozessoren denselben Algorithmus ausführen, können theoretisch beliebig viele Packet Screens parallel angeordnet werden. Damit skaliert dieses Konzept besser als alle statischen Methoden und auch besser als die Verbindungsparallelität.

[6] Eine ganz einfache und effiziente Strategie ist 'first come first serve'. Auf einem Netzsegment treffen Dateneinheiten ohnehin sequentiell ein, so daß der nächste freie Prozessor die am längsten in der Warteschlange stehende Dateneinheit verarbeitet. Eine Unterscheidung der Verbindungen findet nicht mehr statt. Die Dateneinheiten werden konzeptionell als gleichwertig (keine Priorisierung) angesehen, so daß diese einfache Bedienstrategie bei der dynamischen Zuordnung von Paketen zu Prozessoren ausreicht.

Leichte Konfigurierbarkeit: Da derselbe Algorithmus auf allen Packet Screens eingesetzt wird, müssen auch alle Filterregeln auf den Packet Screens identisch konfiguriert werden, so daß kaum Mehraufwand bei der Filterkonfigurierung gegenüber einer herkömmlichen Packet Screen entsteht. Bei den statischen Konzepten müßten alle Filter analog zu den Funktionen zu einzelnen Packet Screen-Instanzen zugeordnet werden. Dies wäre ein erheblicher Mehraufwand. Insbesondere könnte sich durch Hinzufügen eines weiteren Prozessors, einer weiteren Filterregel oder durch die Notwendigkeit, ein zusätzliches Protokoll zu analysieren, die bei statischen Methoden gewählte Zuordnung als nicht mehr optimal erweisen. Dieses Problem tritt bei der Paketparallelität nicht auf.

Potentiell hohe Ausfallsicherheit: Die Ausfallsicherheit kann leicht erhöht werden, da Packet Screens auf Basis der Paketparallelität inhärent redundant ausgelegt sind. Bei Ausfall von einer der n parallelen Instanzen können die verbleibenden $n-1$ Instanzen ohne Rekonfiguration der Filterregeln weiterarbeiten, da die Filterregeln auf allen Instanzen identisch sind. Ebenso einfach ist eine Erhöhung der Anzahl der Packet Screen-Instanzen. Durch das Hinzufügen weiterer Prozessoren läßt sich der Packetdurchsatz und die Ausfallsicherheit erhöhen. Mehraufwand entsteht allerdings für die Erkennung eines Ausfalls einer Packet Screen. Ein ähnlicher Mehraufwand würde jedoch bei allen anderen Parallelisierungskonzepten ebenfalls anfallen, falls eine erhöhte Ausfallsicherheit erreicht werden soll.

Aufgrund der aufgeführten Vorteile soll eine parallele Packet Screen auf Basis der Paketparallelität realisiert werden.

4.3.3 Lastverteilung bei der Paketparallelität

Eine wesentliche Entwurfsentscheidung ist getroffen; die Parallelverarbeitung soll auf Grundlage des Konzeptes der Paketparallelität erfolgen. Es bleibt jedoch das *Problem der Verteilung der Pakete* auf die parallel angeordneten Packet Screen-Instanzen zu lösen.

In Abschnitt 3.2.2.1 wurde dargestellt, daß die Last (hier die zu analysierenden Pakete) an der Netzwerkschnittstelle IF_{sG} anfällt. Diese Last kann man sich unter Berücksichtigung der Paketparallelität am besten als sequentiellen Paketstrom bzw. Aneinanderreihung von Paketen vorstellen. Damit eine nach dem Konzept der Paketparallelität arbeitende, parallelisierte Version der Packet Screen die Dateneinheiten parallel verarbeiten kann, muß ein Mechanismus gefunden werden, die Pakete aus diesem Paketstrom an die parallelen Instanzen zu verteilen.

Es gibt grundsätzlich zwei Möglichkeiten. Die Verteilung kann zentral oder dezentral vorgenommen werden. Dieser Aspekt wurde bereits in [78, 76, 71] ausführlich diskutiert und kann daher an dieser Stelle auf die wesentlichen Erkenntnisse reduziert werden. Für eine zentrale Lastverteilung durch einen dedizierten Verteiler an den Schnittstellen IF_{sG} (vgl. Abbildung 3.9) spricht:

- Der zentrale Verteiler kann durch Rückkoppelung mit den Instanzen für eine gleichmäßige Auslastung der Instanzen sorgen. Dies ist besonders wichtig, wenn die Verarbeitung von verschiedenen Pakettypen zu deutlichen Unterschieden in der Verarbeitungszeit führt. Der zentrale Verteiler würde dann stark belasteten Instanzen weniger Pakete zuteilen als den weniger belasteten Instanzen. Der zentrale Verteiler kann leichter lastabhängige Verteilungen durchführen. Dies ist bei Packet Screens allerdings nicht zu erwarten, da die Notwendigkeit für die Parallelverarbeitung aus der potentiell hohen Anzahl an Aufträgen (Paketen) entsteht und weniger aus der Komplexität der Filteraufgabe.
- Der zentrale Verteiler könnte zusätzlich die Überwachung der Instanzen übernehmen. Fällt eine Instanz aus, beschränkt der Verteiler die Aufteilung der Dateneinheiten auf die verbleibenden Instanzen.

Gegen einen zentralen Verteiler und für die dezentrale Verteilung spricht:

- Der Verteiler kann selbst zum Engpaß bei der Kommunikation werden. Die Vorteile des zentralen Verteilers liegen darin, eine besonders ausgewogene Verteilung bei gleichzeitiger Überwachung der Packet Screen-Instanzen zu ermöglichen. Die Verteilung selbst, sowie die periodische Berechnung der Unterschiede in der Auslastung der Instanzen und die Überwachung der Verfügbarkeit der Instanzen kann dazu führen, daß der zentrale Verteiler selbst zum Engpaß bei der Kommunikation wird. Ein dezentrales Verfahren, bei der die zusätzliche Last durch die Verteilung und Überwachung von den Instanzen selbst übernommen wird, skaliert daher besser als ein zentraler Verteiler.
- Der Verteiler ist ein neuer "single point of failure". Der zentrale Verteiler kann zwar relativ leicht die Instanzen überwachen. Er selbst kann allerdings auch ausfallen. Dies würde zu einem Totalausfall der parallelen Packet Screen führen und ist somit ein ernsthaftes Problem. Durch diese Schlüsseleigenschaft des zentralen Verteilers für die parallele Packet Screen wäre er auch ein bevorzugtes Angriffsziel und stellt somit ein zusätzliches Risiko dar. In jedem Fall müßte der Verteiler besonders gut (ähnlich einer Bastion, vgl. Definition 2.8) gesichert werden.

Zusammenfassend läßt sich feststellen, daß die Nachteile durch einen zentralen Verteiler im Fall der parallelen Packet Screen zu groß sind. Es soll daher eine dezentrale Zuordnung der Pakete zu Prozessoren vorgenommen werden.

4.3.4 Dezentrale Zuordnung der Pakete zu Prozessoren

Die Paketparallelität erfordert die Zuordnung der einzelnen Pakete zu den parallel angeordneten Packet Screen-Instanzen (Prozessoren). Diese Zuordnung führt letztendlich zu einer Lastaufteilung.

Bei dem Konzept der Paketparallelität bedeutet eine dezentrale Zuordnung der Pakete zu den Prozessoren, daß die einzelnen Instanzen der Packet Screen selbst und unabhängig voneinander entscheiden müssen, ob sie eine Dateneinheit bearbeiten oder eine der parallelen Instanzen das jeweilige Datagramm bearbeitet. Dieser dezentrale Entscheidungsprozeß erfordert wiederum, daß alle parallel angeordneten Packet Screen-Instanzen Zugriff auf alle Dateneinheiten haben müssen, um die Entscheidung, welche Instanz eine bestimmte Dateneinheit verarbeiten soll, unabhängig voneinander treffen zu können.

Somit ist das Problem der *dezentralen Zuordnung von Paketen zu Prozessoren* auf zwei Teilprobleme reduzierbar:

1. Weiterleiten der Pakete an alle Prozessoren und
2. dezentrales Selektieren der zu verarbeitenden Pakete.

Das Weiterleiten der Pakete an alle Prozessoren ist zeitkritisch, muß schnell erfolgen und sollte daher in Abhängigkeit vom jeweiligen Netzzugriffsverfahren effizient gelöst werden. Idealerweise übernimmt die ohnehin hochspezialisierte Hardware der aktiven Netzkomponenten (Hubs, Switches, Router) das Kopieren und Weiterleiten der Dateneinheiten an die einzelnen Prozessoren bzw. Instanzen. Besonders vorteilhaft bei dieser Vorgehensweise ist, daß nur solche Komponenten verwendet werden, die ohnehin für den Anschluß der verschiedenen Sicherheitsdomänen an die Packet Screen benötigt werden. Der Hauptnachteil des zentralen Verteilers, einen zusätzlichen "single point of failure" in die Firewall-Installation einzufügen, kann somit einfach vermieden werden.

Nachdem alle Instanzen der parallelen Packet Screen das weitergeleitete Paket empfangen haben, ist dezentral eine Entscheidung darüber zu treffen, welche(r) Prozessor/Instanz die Dateneinheit verarbeitet. Dies ist ein algorithmisches Problem und wird im folgenden Abschnitt diskutiert.

4.3.5 Dezentrale Selektion von Paketen

Es gibt mehrere wichtige Anforderungen an die dezentrale Selektion:

- Die Selektion muß möglichst schnell erfolgen, da sie die Paketverarbeitung andernfalls verzögert und den potentiellen Gewinn durch die Parallelverarbeitung verringert. Eine schnelle Selektion ist nur möglich, wenn der Selektionsalgorithmus aufwandsarm ist. Die Berechnungen für die Selektion müssen daher sehr einfach und von den Prozessoren schnell zu verarbeiten sein. Um einen möglichst hohen Gewinn durch die Parallelverarbeitung zu erreichen, sollte daher für den Zeitaufwand für die Selektion eines Paketes (B_{sel}) im Vergleich zum Zeitaufwand für das Filtern (B_{fil}) der Pakete (P) gelten:

$$\forall p \in P : B_{sel}(p) << B_{fil}(p) \tag{4.8}$$

- Die Zuordnung muß eindeutig sein. Jedes einzelne Paket muß genau einer Packet Screen zugeordnet werden. Der Algorithmus darf demnach weder zu Paketverlusten noch zu Paketvervielfältigungen führen.

- Die Zuordnung sollte zu einer möglichst gleichmäßigen Belastung aller Instanzen[7] führen.

- Der Algorithmus sollte unabhängig von der Anzahl der Prozessoren/Instanzen sein. Dies ist eine Grundvoraussetzung für die Skalierbarkeit. Es ist beispielsweise nicht ausreichend, eine Lösung zu finden, die nur für zwei, drei oder fünf Instanzen geeignet ist.

- Vorteilhaft bei vorübergehenden Paketverlusten auf einer Instanz (beispielsweise aufgrund einer Überlastsituation) wäre eine Zuordnung, die bei einer Paketwiederholung (typisch bei verbindungsorientierten Protokollen wie beispielsweise TCP) mit großer Wahrscheinlichkeit einen anderen Prozessor auswählt. Hierdurch könnte die Robustheit der Protokollverarbeitung durch die parallele Packet Screen unterstützt werden.

Um diesen Anforderungen zu genügen, wird für die verteilte Selektion eine Hash-Funktion (Streu-Funktion) verwendet, die von allen Instanzen unter Verwendung von Informationen aus den eintreffenden Dateneinheiten berechnet werden muß. Eine schnelle Selektion kann durch Berechnen eines Hash-Wertes, also durch Anwenden einer Hash-Funktion erreicht werden. Darüber hinaus liefert eine Hash-Funktion einen eindeutigen Wert, so daß eine Abbildung der Dateneinheiten auf die Instanzen erreicht werden kann. Durch Modifikation der Hash-Berechnung kann der Wertebereich an die Anzahl der Instanzen angepaßt werden, so daß diese Lösung gut skaliert bzw. an eine sich verändernde Anzahl an Instanzen angepaßt werden kann.

Als Eingabe für die Hash-Funktion stehen alle Informationen des Paketes, das selektiert werden soll, zur Verfügung. Sowohl durch die Wahl der Hash-Funktion selbst, als auch durch die Wahl der Attribute der Pakete als Parameter für die Hash-Berechnung, kann die Zuordnung von Paketen zu Prozessoren beeinflußt werden:

IP Adressen: Die Sende- oder Empfänger-Adresse des Datagramms kann als Eingabe für die Hash-Funktion dienen. Aus Sicht einer möglichst hohen Effizienz, ist der Hauptnachteil dieser Wahl, daß alle Pakete mit derselben Adresse (Sende- oder Empfänger-Adresse) von derselben Instanz selektiert werden. Falls es nur eine einzige breitbandige Verbindung gibt, würden alle Pakete dieser Verbindung von derselben Instanz verarbeitet werden. Es gäbe keine Beschleunigung durch die Parallelverarbeitung.

Eine bessere Wahl wäre die Kombination aus Sende- und Empfänger-Adresse. Hierbei könnten zumindest verschiedene Kommunikationsbeziehungen desselben Endgerätes

[7] Bei identischen, d.h. gleich leistungsfähigen Instanzen, ist der größte Gewinn bei einer gleichmäßigen Aufteilung der Aufträge auf die Instanzen zu erwarten.

zu unterschiedlichen Endsystemen auch von verschiedenen Packet Screens verarbeitet werden.

IP Identifikation: Dieser Wert im IP-Datagramm muß für alle Datagramme, die ein Rechner versendet, eindeutig sein. Dieser Wert ist eine gute Wahl für die Berechnung eines Hash-Wertes, da so auch Datagramme derselben Kommunikationsbeziehung von verschiedenen Packet Screens verarbeitet werden können. Außerdem erfüllt eine auf der "IP Identifikation" basierende Selektion die zuletzt genannte Anforderung: Eine TCP-Dateneinheit, die aufgrund von ausbleibenden Bestätigungen erneut gesendet wird, würde auf der IP-Schicht eine andere Identifikation erhalten und somit mit hoher Wahrscheinlichkeit von einer anderen Instanz selektiert werden, als die ursprünglich gesendete Dateneinheit.

IP Kopfprüfsumme: Alle sendenden Systeme berechnen für jedes Datagramm eine Prüfsumme über die Werte im Paketkopf. Dieser Wert eignet sich aufgrund seiner großen Streuung ebenfalls als Eingabe für eine Hash-Funktion. Da die IP Identifikation in die Berechnung der Kopfprüfsumme einbezogen wird, ist die Kopfprüfsumme bei Wiederholungen ebenfalls mit hoher Wahrscheinlichkeit unterscheidbar von der Prüfsumme im ursprünglichen Paket.

Der Hauptnachteil aller bisher diskutierten Eingabewerte ist, daß sie nur in IP-Datagrammen vorkommen. Die TCP/IP Protokollfamilie kennt darüber hinaus aber auch noch das ARP-Protokoll, das neben den IP-Datagrammen ebenfalls unmittelbar in Rahmen des jeweiligen Netzprotokolls (Schicht-2 Protokoll) eingebettet wird. Sollen auch ARP-Dateneinheiten[8] von Packet Screens parallel gefiltert werden, muß daher ein anderer, allgemeinerer Wert für die Verteilung gefunden werden.

Rahmen-Prüfsumme: Als allgemeineren Eingabewert für die Berechnung kann beispielsweise die Rahmen-Prüfsumme der Dateneinheiten des Netzprotokolls verwendet werden. Ethernet, Fast Ethernet, FDDI und auch ATM sichern die Informationen mit Rahmen-Prüfsummen zum Erkennen von Verfälschungen bei der Übertragung ab. Dieser Wert kann ebenfalls zur Selektion benutzt werden und erlaubt somit die Verteilung aller eingebetteten Dateneinheiten.[9]

Mischformen: Eine weitere Alternative stellt die Kombination mehrerer Werte dar. Beispielsweise kann es für eine gleichmäßige Verteilung erforderlich sein, die Hash-Funktion auf mehrere der oben genannten Attribute zu berechnen. Für einige kontextsensitive Packet Screens ('statefull Packet Screens') kann es sogar erforderlich sein,

[8] Falls die Packet Screen transparent in ein Netzsegment integriert werden soll, dann muß das ARP-Protokoll unterstützt werden.

[9] Firewalls werden in der Regel in TCP/IP Netzen eingesetzt. Die Behandlung weiterer Protokollfamilien, wie beispielsweise AppleTalk oder SPX/IPX ist ggf. auch erforderlich. Da die Rahmen-Prüfsumme des Netzprotokolls unabhängig vom eingesetzten Schicht-3 Protokoll vorhanden ist, können alle Dateneinheiten der verschiedenen Protokolle verteilt werden.

daß alle Pakete einer Kommunikationsbeziehung[10] von derselben Packet Screen verarbeitet werden. In diesem Fall wäre es beispielsweise erforderlich, über das Tupel `{IP.source, IP.dest, Port.source, Port.dest}` einen Hash-Wert zu berechnen.

Unabhängig von der Wahl, welche Informationen als Eingabe der Hash-Funktion gewählt werden, kann der Algorithmus zur Selektion einer Packet Screen auf Basis der Paketinformationen folgendermaßen zusammengefaßt werden:

1. Numeriere alle n parallelen Packet Screen-Instanzen von $0, \cdots, n-1$. Jede Packet Screen-Instanz kennt somit ihren Index $0 \leq PS_i < n$, im folgenden auch ID genannt.
2. Wiederhole für alle eintreffenden Pakete auf allen n Instanzen:
   (a) Berechne den Hash-Wert $hw(p)$ über den festgelegten Eingabewerten des Pakets p
   (b) Wenn $hw(p) \neq PS_i$, dann ignoriere (vernichte) das Paket
   (c) Wenn $hw(p) = PS_i$, dann filtere das Paket entsprechend den konfigurierten Regeln

Eine sehr einfache Form, einen Hash-Wert in Abhängigkeit von der Anzahl der Prozessoren und dem Paketinhalt zu berechnen, ist die Modulo-Division der IP-Datagramm-Prüfsumme durch die Anzahl der Prozessoren. Im folgenden Listing ist dieser Algorithmus in Pseudocode dargestellt.

```
/* Folgende globale Definitionen sind auf allen Prozessoren zu konfigurieren */

/* Anzahl der parallelen Prozessoren = n, IDs von 0 bis (n-1) */
#define ANZAHL n

/* Die ID des jeweiligen Prozessors */
#define MYID x

/*
 * MYID : Nummer des Prozessors
 * Datagramm : Die zu verarbeitende Dateneinheit
 *
 * Rueckgabewert: 0 == nicht selektiert
 *                1 == selektiert (dieser Prozessor filtert die Dateneinheit)
 */

int select (int MYID, struct datagram* ip)
```

[10] Eine Kommunikationsbeziehung bezeichnet einen logisch zusammenhängenden Datenfluß zwischen Kommunikationspartnern. Dies kann verbindungsorientiert (z.B. TCP) oder auch verbindungslos (z.B. UDP) erfolgen (vgl. Definition 3.1).

```
{
  int hashval;

  hashval = ntohs(ip->header_check_sum) % ANZAHL;
  if ( hashval == MYID ){
    return 1;
  }
  else{
    return 0;
  }

}
```

Diese Selektion kann zu verschiedenen Zeitpunkten nach dem Empfang der Dateneinheit im Kernel geschehen. Grundsätzlich werden alle Dateneinheiten von allen parallel angeordneten Packet Screens über die jeweiligen Hardware-Schnittstellen (hier Fast Ethernet) empfangen. Unmittelbar danach kann aufgrund der Informationen in den Paketen die Selektion beginnen.

Es gibt bei Universalrechner-basierten Implementationen grundsätzlich zwei verschiedene Verfahren, die theoretisch zu unterschiedlichen (Paket-)Durchsätzen führen. Die Filterverarbeitung kann im Kernel oder auf der Anwendungsebene erfolgen. Aus Gründen der Effizienz werden nur die Kernel-basierten Möglichkeiten diskutiert:

- Selektion innerhalb des IP-Filter Moduls
- Selektion innerhalb des IP-Kernel Moduls
- Selektion innerhalb der Geräte-Treiber

Die theoretischen Vor- und Nachteile der verschiedenen Methoden werden in den folgenden Unterabschnitten diskutiert.

4.3.6 Selektion innerhalb des IP-Filter Moduls

Die Selektion innerhalb des IP-Filter Moduls erscheint sehr naheliegend. Bei einer einzigen Packet Screen müßten alle Dateneinheiten innerhalb dieses Moduls entsprechend den konfigurierten Regeln ausgewertet werden. Bei einer parallelen Lösung kann eine zusätzliche "Regel" die erforderliche Selektion übernehmen, die vor dem eigentlichen Filtern die Dateneinheit ggf. aussortiert. Das Filtern findet also nur noch für die Dateneinheiten statt, die der verteilte Selektionsalgorithmus der jeweiligen Packet Screen zuteilt, so daß die Last beim Filtern reduziert wird.

Neben der Reduktion der Last beim Filtern tritt eine weitere Lastreduktion auf. Die Dateneinheiten, die vom Filteralgorithmus nicht vernichtet werden, müssen in das jeweils

andere Netz übertragen werden. Hier entsteht also Aufwand für das Versenden der empfangenen und gefilterten Dateneinheiten. Ein Teil dieser Dateneinheiten wird jedoch durch die Selektion vernichtet, also von einer anderen Instanz weitergeleitet, so daß der Aufwand für das Weiterleiten ("Forwarding") von Dateneinheiten ebenfalls abnimmt und es zu einer weiteren Entlastung der einzelnen Instanzen kommt.

4.3.7 Selektion innerhalb des IP-Kernel Moduls

Eine weitere Möglichkeit, die Selektion durchzuführen, besteht darin, im IP-Kernel Modul die Dateneinheiten zu selektieren und ggf. zu vernichten. Da das IP-Kernel Modul als STREAM-Modul "über" dem IP-Filter Modul liegt, sollte hierdurch allerdings keine Durchsatzsteigerung gegenüber dem Selektieren im Filter-Modul möglich sein.

Diese Alternative bietet sich immer dann an, wenn die IP-Implementierung des Kernels bereits über Filtermöglichkeiten verfügt. In diesem Fall kann auf den Einsatz eines zusätzlichen IP-Filtermoduls verzichtet werden. Beispielsweise verfügen viele LINUX-Kernels bereits über Filterfunktionalität innerhalb ihrer Standard-Netzmodule ("IP-Chains" [201]).

4.3.8 Selektion innerhalb der Geräte-Treiber

Der Hauptnachteil der beiden vorgestellten Methoden ist, daß alle empfangenen Dateneinheiten als STREAM-Nachrichten [229, 224] durch den Kernel an die entsprechenden Module (IP oder IP-Filter) geleitet werden müssen. Dabei entsteht ein nicht zu vernachlässigender Overhead durch das Allozieren entsprechenden Kernelspeichers und Kopieren der Dateneinheit von der Netzschnittstelle in diesen Speicher (vgl. auch die Modellierung dieser Vorgänge in Anhang A.2.3).

Wesentlich effizienter könnte eine Selektion sein, die Dateneinheiten bereits vernichtet, bevor aufwendige Kopieroperationen stattfinden. Dies ist nur auf der untersten Ebene im Betriebssystem möglich, also im Geräte-Treiber (Device-Driver) der Netzschnittstelle. Hier können beispielsweise die Informationen einer Dateneinheit in den Kernel kopiert werden, die für eine Selektion unbedingt erforderlich sind. Falls die Dateneinheit aufgrund der Selektion vernichtet werden kann, ist kein weiteres Kopieren mehr erforderlich, und es braucht auch kein Kernelspeicher alloziert zu werden. Theoretisch ist mit dieser Methode der größte Leistungsgewinn erzielbar.

Ein Nachteil dieser Methode ist, daß die Änderungen für die Selektion bei verschiedenen Netz-Schnittstellen in allen dazugehörenden Geräte-Treibern durchgeführt werden müssen. Die anderen Methoden arbeiten auf der vom jeweiligen Medium unabhängigen IP-Ebene und müßten daher nur an einer Stelle im Kernel implementiert werden.

Allerdings spricht für diese Methode, daß alle Firewall-Produkte, die auf Informationen höherer Protokollschichten zurückgreifen, von dieser Parallelisierungsmethode auf niedrigster Ebene profitieren können. Diese Produkte müssen selbst nicht verändert werden, und

es kann dennoch eine Beschleunigung herbeigeführt werden. Auch im Sinne der Überlebensfähigkeit ist dies eine sinnvolle Vorgehensweise, da durch die Selektion innerhalb der Gerätetreiber ein Einsatz von kommerziellen[11] Firewall-Produkten (Software) auf höheren Protokollschichten möglich wird, die andernfalls mangels fehlender Quellcodes[12] nicht zum Einsatz kommen könnten. Es wäre durch die Selektion im IP-Modul oder im Gerätetreiber sogar möglich, verschiedene Packet Screen Versionen auf darüberliegenden Ebenen/Modulen einzusetzen.

Es wird daher für eine Prototyp-Implementation eine Selektion im Netz-Gerätetreiber angestrebt. Der folgende Abschnitt beschreibt die Realisierungen der parallelen Packet Screen für Fast Ethernet. Im Abschnitt 4.5 werden die mit dieser Vorgehensweise erreichten Durchsatzgewinne anhand von Messungen dokumentiert und diskutiert.

4.4 Realisierung von Hochleistungs-Packet Screens

Im vorigen Abschnitt wurde bereits angedeutet, daß bei der Realisierung das Weiterleiten einer Dateneinheit — zum Zweck der verteilten Selektion — an alle parallelen Instanzen der Packet Screen sehr zeitkritisch ist. In Abhängigkeit vom jeweiligen Netztyp ("shared vs. switched") muß daher eine Realisierung speziell für Fast- und Gigabit Ethernet entwickelt werden.

4.4.1 Parallele Packet Screen für "shared" Ethernet

In diesem Abschnitt wird die Realisierung eines Prototypen für alle 'Carrier Sense Multiple Access with Collision Detection' (CSMA/CD) Netze vorgestellt, also insbesondere Ethernet und Fast Ethernet. Eine Möglichkeit für den Aufbau einer solchen Packet Screen basiert auf der in Abbildung 4.7 dargestellten Topologie.

Die dargestellte Konfiguration zeigt zwei Netze, die durch die Packet Screen voneinander getrennt sind. Die Topologie der Netze (die innere Struktur der Netze) ist hierbei beliebig. Die Packet Screen ist zwischen zwei Hubs (Multiport Repeatern) eingefügt, die sich als preisgünstige Netzbausteine für UTP-basiertes Ethernet und Fast Ethernet bewährt haben. In Abbildung 4.8 ist dargestellt, wie einfach ein paralleles Setup auf Basis dieser Architektur zu realisieren ist.

Die Hubs übernehmen hierbei das Weiterleiten der Dateneinheiten aus dem angeschlossenen Netz an alle Instanzen. Das bei "Multiport Repeatern" übliche Verhalten ist, daß alle Dateneinheiten, die auf einem Port (physikalische Schnittstelle) eintreffen, auf alle anderen

[11] Die Verwendung von sog. "Commercial Off-The-Shelf, COTS"-Komponenten für überlebensfähige Systeme ist strittig.

[12] Durch die fehlenden Quellcodes könnten die Modifikationen für die Parallelisierung nicht in der Firewall-Software vorgenommen werden.

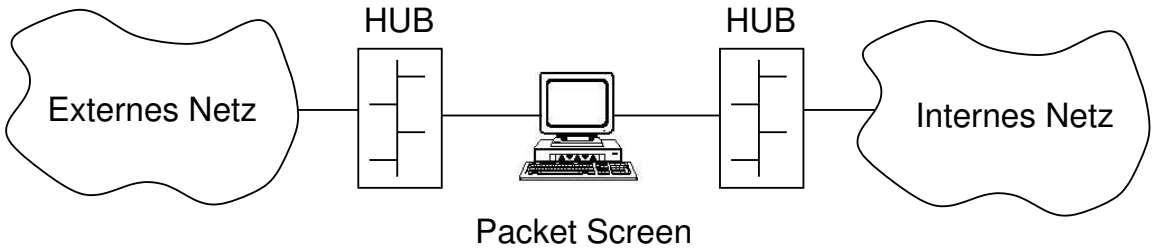


Abbildung 4.7: Packet Screen in CSMA/CD-Netzen

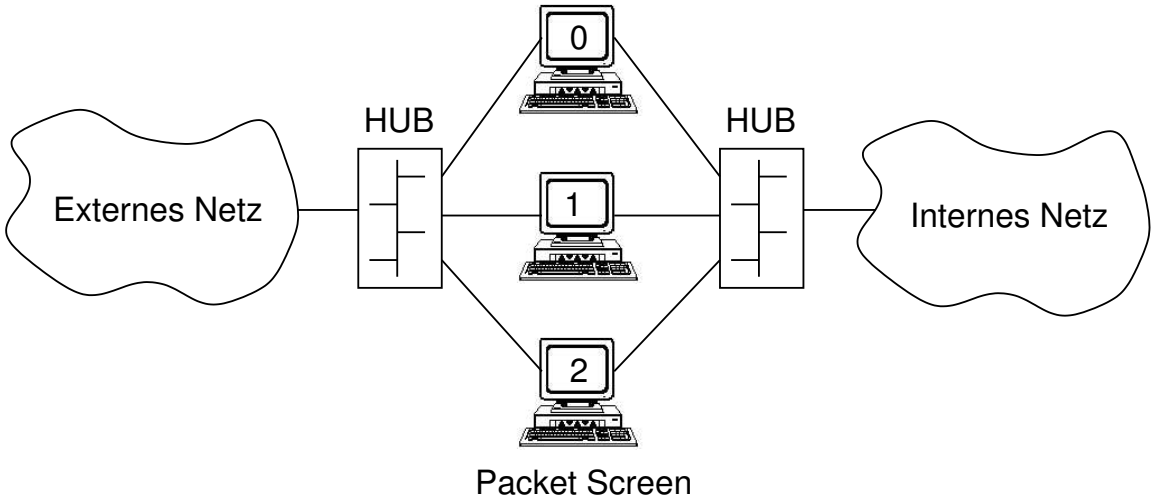


Abbildung 4.8: Parallele Packet Screen in "shared" CSMA/CD-Netzen

Ports ausgegeben werden. Damit ist sichergestellt, daß alle Dateneinheiten vom angeschlossenen Netz an alle Packet Screens geleitet werden. Das Kopieren und Weiterleiten der Dateneinheiten an alle Instanzen ist somit in "shared" CSMA/CD-Netzen trivial.

Damit die Dateneinheiten jedoch von allen Instanzen empfangen werden, ist es erforderlich, den sog. "promiscous mode" auf den Ethernet-Schnittstellen zu aktivieren. Dadurch werden alle Dateneinheiten empfangen und nicht nur diejenigen mit der Empfangsadresse der jeweiligen Instanz. Auf den Packet Screens muß bei dieser Topologie jedoch sichergestellt werden, daß Dateneinheiten, die eine PS in ein Netz sendet, von den parallel angeordneten PS nicht erneut verarbeitet werden. Dies ist leicht durch Konfigurieren der physikalischen Netzadressen aller Ethernet-Schnittstellen auf den PS in einer "Ausschlußliste" zu erreichen. Dateneinheiten von einer PS werden somit von den anderen PS ignoriert.

Bei der eben diskutierten Vorgehensweise kann die parallele Packet Screen transparent in ein Netzsegment eingesetzt werden. Die Rechner in den angeschlossenen Netzen nehmen die Packet Screen nicht wahr.

Alternativ können die Packet Screen Instanzen aber auch als Router konfiguriert werden. Dann müssen die Rechner der angeschlossenen Netzsegmente die Packet Screen direkt adressieren, um Dateneinheiten durch sie weiterleiten zu lassen. Die Packet Screen wäre somit nicht mehr transparent, jedoch könnte dann auf die "Ausschlußliste" verzichtet werden, da aufgrund der direkten Adressierung kein "promiscous mode" mehr erforderlich ist. Dieses Verfahren wird im nächsten Abschnitt beschrieben und wurde bei allen Messungen benutzt.

4.4.2 Verbesserte parallele Packet Screen für "switched" Ethernet

Leider hat die im vorigen Abschnitt vorgeschlagene Art der Verteilung über Hubs einen großen Nachteil: Durch den Einsatz der Hubs ist es nicht mehr möglich, die Packet Screens im "full duplex" Modus zu betreiben. Der "full duplex" Betrieb (fdx) ist bei Fast Ethernet (FE) nur über Switches möglich, die jedem angeschlossenen Rechner ein eigenes Segment mit getrennten Sende- und Empfangsleitungen zuordnen. Erst so kann ein Rechner gleichzeitig senden und empfangen. Dadurch ist theoretisch eine Gesamtübertragungsgeschwindigkeit von 200 Mbit/s möglich, also eine Verdoppelung der Übertragungsgeschwindigkeit im Vergleich zum "half duplex" Betrieb (hdx). Durch die drastisch sinkenden Kosten für Fast Ethernet Switches ist es sehr wahrscheinlich, daß Switches in Zukunft auch zum Endgeräteanschluß eingesetzt werden. Es ist daher erforderlich, die Verteilung auch über einen Switch zu realisieren.

Da Switches die Ethernet Adressen der an ihren Ports angeschlossenen Geräte lernen, können sie eintreffende Dateneinheiten zielgerichtet weiterleiten. Kollisionen treten im Switch nur noch auf, falls gleichzeitig von mehr als einem Port Dateneinheiten auf denselben Ausgangsport geleitet werden. Auch in diesem seltenen Fall können moderne Switches durch Puffern (sog. "Store and Forward"-Techniken) von Dateneinheiten viele Kollisionen vermeiden.

Durch das gezielte Weiterleiten der Ethernet-Rahmen wird im Gegensatz zum Hub keine automatische Verteilung auf mehrere parallele Packet Screen-Instanzen erreicht. Man benötigt einen zusätzlichen Mechanismus zum Replizieren der Dateneinheiten im Switch.

Für die verbesserte parallele Packet Screen wurde die Verteilung auf dem Switch durch die Verwendung von Ethernet-"Multicast"-Adressen erreicht. Alle parallel angeordneten Packet Screen-Instanzen wurden so konfiguriert, daß sie Dateneinheiten empfangen, die an eine bestimmte Ethernet-"Multicast"-Adresse gerichtet sind. Die Switches müssen entsprechend konfiguriert werden. Alle Ports (Schnittstellen), an denen eine der parallelen Packet Screen-Instanzen angeschlossen ist, müssen durch Konfigurieren des Switches einer "Multicast"-Gruppe zugeordnet werden. Dies geschieht durch die statische Zuordnung der Ethernet-"Multicast"-Adresse der parallelen Packet Screen zu den Schnittstellen des Switches, an die eine der Packet Screen-Instanzen angeschlossen ist. Alle eintreffenden Dateneinheiten mit der gewählten "Multicast"-Adresse als Empfangsadresse werden dann vom Switch auf

den konfigurierten Schnittstellen ausgegeben. Die extrem schnelle Verteilung auf mehrere parallele Instanzen ist damit erreicht.

An den Geräte-Treibern für die FE-Schnittstellen der Packet Screen-Instanzen mußten zu diesem Zweck einige Änderungen vorgenommen werden:

1. Die Rechner, die über die Packet Screen kommunizieren sollen, erfragen die IP-Adresse der Packet Screen mit dem "Address Resolution Protocol" (ARP, vgl. [189]). Es ist daher erforderlich, daß eine der parallelen Packet Screen-Instanzen auf die ARP-Anfrage mit der konfigurierten "Multicast"-Adresse antwortet.

   Die Geräte-Treiber der FE-Schnittstellen wurden daher so erweitert, daß bei Anfragen nach bestimmten IP-Adressen immer mit der konfigurierten "Multicast"-Adresse als "Hardware"-Adresse im ARP-Protokoll geantwortet wird. Die Kommunikationspartner benutzen dann zur Kommunikation über die Packet Screen automatisch die "Multicast"-Adresse, so daß die Verteilung durch den Switch erfolgen kann (vgl. Abschnitt C.3).

2. Der Empfang von Dateneinheiten mit "Multicast"-Empfängeradresse erfolgt nicht automatisch. Vielmehr muß für jeden "Service Access Point" (SAP) festgelegt werden, welche "Multicast"-Adressen gültig sind. Da das IP-STREAM-Modul für den IP-SAP den "Multicast"-Empfang nicht ohne eine entsprechende Anforderung eines Anwendungsprogramms einschaltet, wurde der "Multicast"-Empfang durch einen zusätzlich implementierten "StreamIO"-Aufruf zur Verfügung gestellt. Auf Anforderung kann der Geräte-Treiber dazu veranlaßt werden, "Multicast"-Pakete mit einer bestimmten Adresse zu empfangen.

Der erstellte Prototyp konnte mit den in Abschnitt 3 erstellten Werkzeugen analysiert werden. Die Ergebnisse dieser Leistungsbewertung werden im folgenden Abschnitt vorgestellt.

4.5 Leistungsbewertung der parallelen Packet Screen

Die Bewertungen der Filteroptionen in den Abschnitten 4.3.6–4.3.8 haben ergeben, daß durch das frühzeitige Vernichten von Dateneinheiten durch die Geräte-Treiber theoretisch der größte Durchsatzgewinn bei parallelen Szenarien erreichbar ist.

Andererseits will man die eigentliche Filteraufgabe, aus Gründen der Modularität und zur Vermeidung von Redundanz im Kernel, in einem eigenständigen Modul unterbringen. Dadurch kann der Selektionsalgorithmus bzw. die Parallelisierung auch unabhängig von der Filteraufgabe für andere Zwecke verwendet werden (vgl. Kapitel 10).

Der Prototyp einer parallelen Packet Screen ist als Mischform dieser beiden Konzepte implementiert. Dadurch kann man die Vorteile der beiden Ansätze kombinieren:

- Schnelle Selektion und Lastreduktion im Geräte-Treiber

- Komfortable Filterkonfigurierung und -Anwendung im IP-Filter Modul

Der erstellte Prototyp wurde mit Hilfe von Messungen bewertet. Es sollte festgestellt werden, ob die vorgenommene Parallelisierung tatsächlich zu einer Beschleunigung der Packetverarbeitung führt und die gewählten Mechanismen für die Lastverteilung unabhängig vom Szenario verwendet werden können. Dazu wurden vier Meßreihen geplant:

1. Bewerten der Leistungsfähigkeit in Abhängigkeit von der Anzahl der eingesetzten Instanzen und der Nachrichtenlänge. Bei dieser Meßreihe wird auf das Konfigurieren von Filterregeln verzichtet, um die "Maximum Forwarding Rate" (MFR, vgl. [158]) zu ermitteln. Diese Meßreihe zeigt also, ob überhaupt eine Leistungssteigerung durch die Parallelverarbeitung möglich ist (vgl. Abschnitt 4.5.1.1).

2. Bewerten der Leistungsfähigkeit in Abhängigkeit von der Anzahl der eingesetzten Instanzen, der Nachrichtenlänge und der Anzahl der konfigurierten Filterregeln. Auch bei dieser Meßreihe wird die "Forwarding Rate" (FR, vgl. [158]) ermittelt. Sie müßte durch den zusätzlichen Aufwand für die Anwendung von Filterregeln geringer sein als die MFR. Falls diese Erwartung zutrifft, kann mit dieser Meßreihe (vgl. Abschnitt 4.5.1.2) zusätzlich analysiert werden, ob die Leistungseinbußen durch die Anwendung der Filterregeln genauso auffällig sind, wie bei einer sequentiellen Packet Screen.

3. Bewerten der Leistungsfähigkeit in Abhängigkeit von der Kommunikationsrichtung. Bei dieser Meßreihe sollen Abhängigkeiten der Leistungsfähigkeit von der Kommunikationsrichtung aufgezeigt werden. Da Hubs für die Verteilung der Last eingesetzt werden und die an der Schnittstelle IF_{sM} anfallenden Paketströme aller parallelen Instanzen wieder bündeln müssen, ist es nicht auszuschließen, daß bei einer bidirektionalen Kommunikation, bei der die Last sozusagen aus "beiden Richtungen" auf das SuL einströmt, Pakete durch Kollisionen im Übertragungsmedium verloren werden. Die Bewertung dieses Szenarios erfolgt in Abschnitt 4.5.1.3 nach dem Schema aus Abbildung 3.5.

4. Bewerten der Leistungsfähigkeit in Abhängigkeit vom Übertragungsmedium. Die oben aufgestellte Vermutung, der Einsatz von Hubs als physikalisches Übertragungsmedium könnte zu Kollisionen und Paketverlusten führen, motivierte u.a. den Entwurf der verbesserten parallelen Packet Screen aus Abschnitt 4.4.2, die Switches anstelle der Hubs verwendet. Die Meßreihe in Abschnitt 4.5.2 soll daher die Leistung in Abhängigkeit von der Anzahl der Instanzen und der Nachrichtenlänge unter Verwendung von Switches als Übertragungsmedium aufzeigen.

Für diese Untersuchungen mußten die Messungen aus Abschnitt 3 präzise wiederholt werden. Anstelle jedoch eine einzige Packet Screen als SuL nach Abbildung 3.3 zu betrachten, wurde der Prototyp bestehend aus zwei, drei und vier parallel arbeitenden Instanzen als SuL nach Abbildung 3.4 untersucht.

Der Generator reproduzierte dazu die in Abschnitt 3.2.2.1 diskutierte Last. Wie in den vorigen Abschnitten diskutiert, wurde die Verteilung dieser Last von Hubs bzw. Switches vorgenommen. Man kann sich diese Netzkomponenten als Implementation der Schnittstelle IF_{sG} vorstellen (vgl. Abbildung 3.9), so daß jedem Subsystem (SuL_i) die gesamte Last zugeführt wird.

Auch in diesem Fall wird, wie bereits in Abschnitt 3.6.2, über einen Zeitraum von 10 Sekunden gemessen und die mittlere Leistung in Abhängigkeit von der Nachrichtenlänge und der Anzahl der konfigurierten Filterregeln berechnet. Die vom Lastgenerator erzeugte Last ist mit `snd` angegeben. Diese Last entspricht der in Anhang C.2.4 diskutierten, reproduzierbaren Last, die an der Schnittstelle IF_{pG} vom Generator beauftragt wird. Der ausgangsseitige Durchsatz ($rcv(i)$) ist um die Anzahl der parallelen Instanzen (i) in den folgenden Abbildungen ergänzt. Es handelt sich hierbei um eine weitere Konfigurationskenngröße, die zusätzlich zu der Anzahl der Filterregeln vor jeder Messung eingestellt wurde.

4.5.1 Parallele Packet Screen in "shared" Fast Ethernet Netzen

Die folgenden Ergebnisse wurden bereits im Detail in [21] diskutiert, so daß hier eine etwas knappere Darstellung ausreicht. Als Übertragungsmedium wurden Hubs verwendet, so daß es sich um ein "shared" Fast Ethernet-Netz handelt. Es werden drei Abbildungen mit Darstellungen von durchgeführten Meßreihen diskutiert. Das erste Szenario zeigt, daß die Verteilung über einen Hub (Multiport Repeater) tatsächlich funktioniert. Die zweite Abbildung belegt die Erwartung, daß die Gewinne durch Parallelverarbeitung im Falle von aufwendigen Filteroperationen besonders gut sind. Die dritte Abbildung dient als Ausgangspunkt für die Weiterentwicklung der "shared medium" zur "switched" parallelen Packet Screen, da deutlich die Nachteile einer Hub-basierten Konfiguration erkennbar sind.

4.5.1.1 Messung ohne konfigurierte Filterregeln

Abbildung 4.9 zeigt, wie sich der Einsatz von zwei bis vier parallelen Packet Screen Instanzen auf den erreichbaren Durchsatz in Abhängigkeit von der Paketlänge auswirkt. Zu beachten ist, daß hier keine Filterregeln abgearbeitet wurden. Dies ist somit der maximal erreichbare Durchsatz (MFR) durch Einsatz der vorgeschlagenen Parallelisierungsmethode. Es wird sozusagen ein optimales "Caching" der Filterentscheidung simuliert, die den Filteraufwand vernachlässigbar klein werden läßt. Die generierte Last entspricht der Last bei der Leistungsbewertung aus Abschnitt 3.6.2. Der Durchsatz ist wiederum als mittlerer Durchsatz nach $t = 10s, \Delta t = 10s$ in Abhängigkeit von der Nachrichtenlänge und der Anzahl der Filterregeln angegeben. Als zusätzliche Konfigurationskenngröße ist, wie oben angedeutet, die Anzahl der parallelen Instanzen beim ausgangsseitigen Durchsatz (rcv) vermerkt. Der Versuchsaufbau entspricht dem Modell aus Abbildung 3.4.

In der Abbildung ist der theoretisch erreichbare Datagrammdurchsatz in Abhängigkeit von der Paketlänge eingezeichnet. Diese Kennlinie ergibt sich unmittelbar aus der Argumenta-

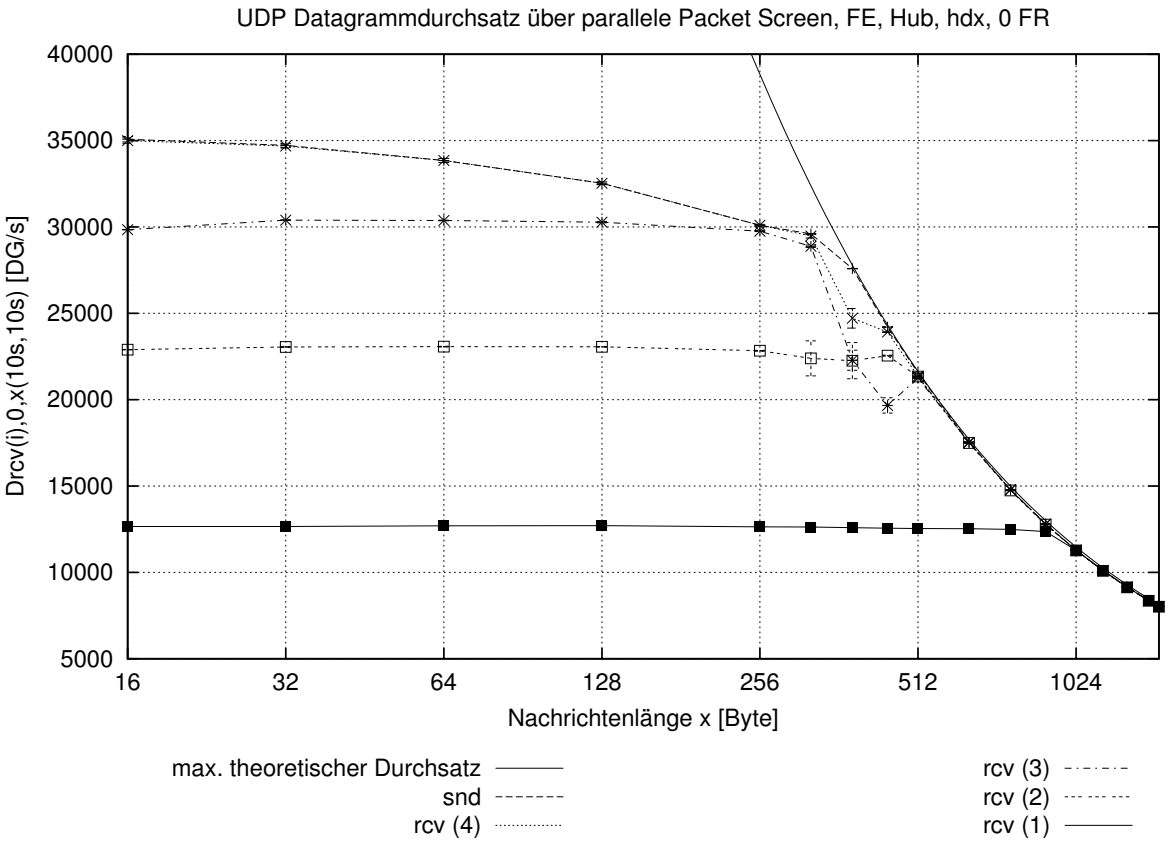


Abbildung 4.9: Erzielter Durchsatz in Datagrammen pro Sekunde, "shared" FE-Netz, keine Filterregeln

tion in Abschnitt 3.5. Die Kurve verläßt den dargestellten Ausschnitt (0 bis 40.000 Datagramme pro Sekunde), da bei kleineren Dateneinheiten der theoretische Durchsatz weiter ansteigen würde (bis maximal 148.810 Datagramme pro Sekunde, vgl. Abbildung 3.11).

Qualitativ lassen sich die Meßergebnisse folgendermaßen zusammenfassen:

- Die Parallelverarbeitung ergibt deutliche Leistungsverbesserungen. Bei zwei Instanzen wird fast eine Verdoppelung des Durchsatzes einer einzelnen Instanz erreicht. Die Filterleistung von vier parallelen Instanzen ist ausreichend, um die gesamte Last durch den Lastgenerator zu verarbeiten. Die entsprechende Empfangskurve (`rcv (4)`) überdeckt bis auf zwei Meßpunkte die Lastkurve (`snd`). Das läßt vermuten, daß vier parallele Instanzen noch über Leistungsreserven verfügen.

- Es gibt bei bestimmten Paketlängen (320 und 384 Byte) bei zwei Instanzen größere Leistungsschwankungen, erkennbar an den etwas größeren Vertrauensintervallen. Bei drei Instanzen findet bei 384 und 448 Byte langen Nachrichten ein Leistungseinbruch

statt. Dieses Verhalten ist reproduzierbar; es kann – als eine mögliche Erklärung für dieses Phänomen – nur vermutet werden, daß diese Nachrichtenlängen ungünstig für die Pufferverwaltung im Kernel ist (auch bei vier Instanzen ist bei 384 Byte langen Nachrichten ein kleiner Einbruch erkennbar).

- Bereits zwei Instanzen sind in der Lage, den theoretischen Durchsatz ab einer Nachrichtenlänge von 512 Byte zu erreichen.

4.5.1.2 Messung mit 50 konfigurierten Filterregeln

In Abbildung 4.10 ist der erreichbare Durchsatz als weiteres Beispiel bei 50 konfigurierten Filterregeln angegeben ($D_{rcv(i),50,x}(10s, 10s)$). Auch hier erkennt man die dramatischen Auswirkungen der Anwendungen von Filterregeln.[13] Es fallen folgende Aspekte auf:

- Durch alleinige, visuelle Auswertung der Meßreihen anhand von Abbildung 4.10 könnte man den Eindruck erhalten, es würde sich ein linearer Gewinn durch die Parallelverarbeitung einstellen. Der Aufwand für das Filtern dominiert durch die 50 konfigurierten Filterregeln den Aufwand für die Selektion zumindest deutlich ($rcv(1) * 4 \approx rcv(4)$ für kleine bis mittlere Paketlängen).

- Die Leistungsgewinne durch Parallelverarbeitung sind besonders gut, je aufwendiger die Filterverarbeitung ist. Dies war auch zu erwarten, da der Overhead durch die zusätzliche Selektion prozentual geringer wird, je aufwendiger die anschließende Filterverarbeitung ist.

4.5.1.3 Bidirektionale Messung mit zwei Lastgeneratoren

Die in Abschnitt 4.5.1.1 angedeuteten Leistungsreserven der parallelen Packet Screen aus vier Instanzen wurden weiter untersucht. Offensichtlich war der Lastgenerator nicht in der Lage, eine entsprechend hohe Last zu generieren, um alle vier Instanzen auszulasten. Es wurde daher eine weitere Meßreihe durchgeführt, bei der zwei Lastgeneratoren und zwei Monitore zum Einsatz kamen. Außerdem muß die Auswirkung einer bidirektionalen Last untersucht werden.

Für die Durchführung dieser Meßreihe waren u.a. die Erweiterungen zur Synchronisation der Meßwerkzeuge erforderlich. Das Modell für die Messung entspricht Abbildung 3.5. Der dafür erforderliche Testnetzaufbau wurde bereits in Abbildung 3.8 gezeigt.

[13] Beim direkten Vergleich mit Abbildung 3.12 fällt auf, daß bei einer Instanz der Durchsatz mit 50 konfigurierten Filterregeln nicht dem bei der Engpaßanalyse erreichten Durchsatz einer sequentiellen Packet Screen entspricht. Dafür gibt es zwei Gründe: erstens braucht die sequentielle Packet Screen keine Selektion durchzuführen und zweitens wurde der Engpaß unter Verwendung von Switches (vgl. Abschnitt 4.5.2) bestimmt.

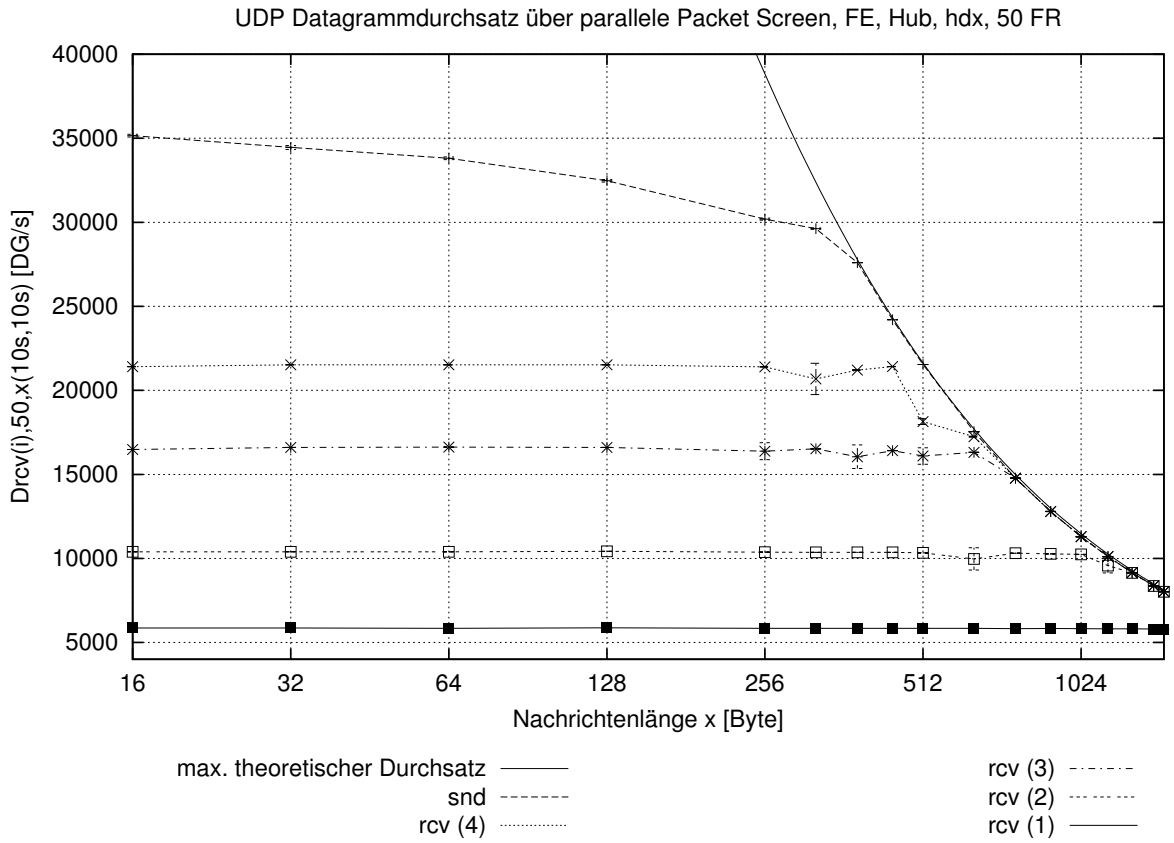


Abbildung 4.10: Durchsatzgewinn in Datagrammen pro Sekunde, "shared" FE-Netz, 50 Filterregeln

In Abbildung 4.11 ist der Leistungseinbruch der parallelen Packet Screen bei Verwendung von zwei Lastgeneratoren zu erkennen:

- Die generierte Last ist erwartungsgemäß verdoppelt. Zusammen können die beiden Lastgeneratoren maximal etwa 70.000 Dateneinheiten pro Sekunde bei kleinen Nachrichtenlängen generieren.

- Der gemessene Durchsatz in Paketen pro Sekunde verläuft qualitativ völlig anders als bei den beiden vorherigen (unidirektionalen) Meßreihen. Es ist bei mittleren Paketlängen (ab 128 Byte bis ca. 576 Byte) ein sehr starker Durchsatzeinbruch zu erkennen.

- Der Paketdurchsatz ist auch bei großen Dateneinheiten stets deutlich unterhalb des theoretischen Limits.

- Die erwarteten Leistungsreserven sind bei kleinen Paketlängen (bis ca. 128 Byte) erkennbar. Der Durchsatz ist hier bei vier Instanzen um mehrere tausend Dateneinheiten pro Sekunde höher als bei der unidirektionalen Lastmessung.

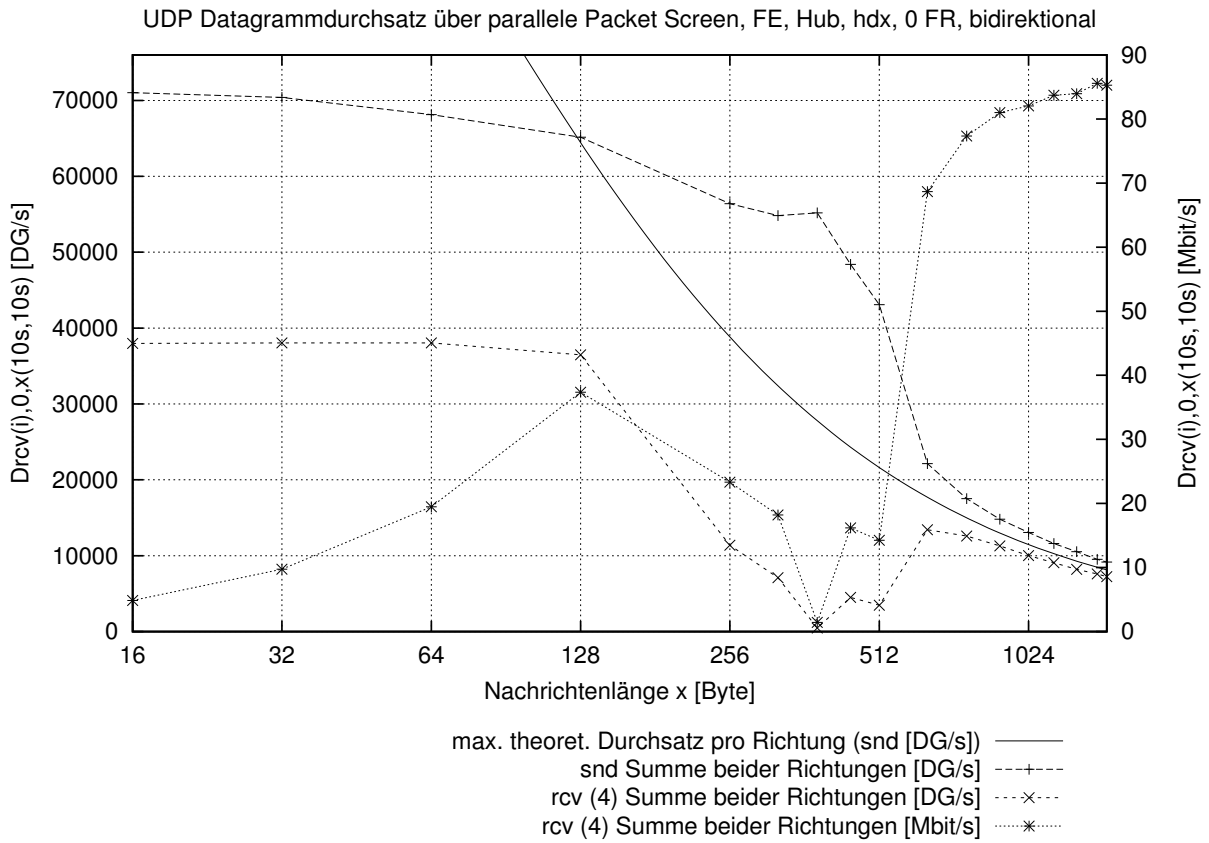


Abbildung 4.11: Durchsatzeinbruch bei mittleren Datagrammlängen und beidseitiger Last, "shared" FE-Netz, 0 Filterregeln

Das beobachtete Phänomen ist zumindest partiell erklärbar. Durch die zunehmende Last auf dem Übertragungsmedium kommt es offenbar immer häufiger zu Kollisionen und somit zu Paketverlusten. Die Kollisionen entstehen dadurch, daß beide Lastgeneratoren Pakete generieren, die mit den bereits gefilterten Paketen kollidieren können. Es gibt auf beiden Seiten der parallelen Packet Screen durch die Hubs jeweils eine Kollisionsdomäne, bei der alle Instanzen und der Lastgenerator um das Übertragungsmedium konkurrieren (daher auch "shared" FE-Netz). An der Schnittstelle IF_{sG} liegt somit nicht nur die vom Generator produzierte Last ($genload(x)$) in Abhängigkeit von der Nachrichtenlänge (x) an, sondern zusätzlich die aus der "anderen Richtung" eintreffende Last durch die Filterleistung[14] der

[14] Dies ist bei unidirektionalen Messungen der an der Schnittstelle IF_{sM} anfallende, ausgangsseitige Durchsatz der Packet Screen. Dieser Durchsatz ist für das "shared medium" bei bidirektionalen Messungen jedoch eine zusätzliche Last.

Packet Screen ($filterload(x)$). Die Summe dieser beiden Teillasten ergibt die Gesamtlast ($FEload(x)$) pro Kollisionsdomäne (Ethernet-Segment).

Nimmt man an, daß sich die gefilterten Pakete auf beide Übertragungsrichtungen gleichmäßig verteilen, so ist die Netzauslastung bei 16 Byte langen Nachrichten (UDP-SDU = 16) in beiden Kollisionsdomänen bei nur:

$$\begin{aligned} FEload(16) &= genload(16)/2 + filterload(16)/2 \\ 54504.7\ DG/s &= 35531.43\ DG/s + 18973.27\ DG/s \\ &\approx 36,63\% \text{ Netzauslastung} \end{aligned} \tag{4.9}$$

Betrachtet man hingegen die Netzauslastung in den beiden Kollisionsdomänen bei 128 Byte großen Nachrichten (UDP-SDU = 128 Byte), so ergibt sich:

$$\begin{aligned} FEload(128) &= genload(128)/2 + filterload(128)/2 \\ 50819.73\ DG/s &= 32580.31\ DG/s + 18239.42\ DG/s \\ &\approx 78,87\% \text{ Netzauslastung} \end{aligned} \tag{4.10}$$

Hierdurch steigt die Wahrscheinlichkeit für Kollisionen natürlich dramatisch an. An der Einbruchsstelle bei Nachrichtenlängen von 384 Byte (UDP-SDU = 384 Byte) ist bereits anhand der Graphik aus Abbildung 4.11 erkennbar, daß die beiden Lastgeneratoren in der Lage sind, die zwei Kollisionsdomänen zu 100% auszulasten: Die akkumulierte Last ist doppelt so hoch, wie das theoretische Limit. Die genauen Zahlen bestätigen diese visuelle Abschätzung:

$$\begin{aligned} FEload(384) &= genload(384)/2 + filterload(384)/2 \\ 27817.95\ DG/s &= 27584.55\ DG/s + 233.4\ DG/s \\ &\approx 100\% \text{ Netzauslastung} \end{aligned} \tag{4.11}$$

Die empirischen Werte ergeben in Formel 4.11 eine Netzauslastung von 100% (auf 2% genau).

Vollständig erklären läßt sich die der Kurvenverlauf aus zwei Gründen nicht:

1. Die Netzauslastung von 100% bei der Nachrichtenlänge von 384 Byte erklärt zwar grundsätzlich Paketverluste durch Kollisionen aufgrund der hohen Kollisionswahrscheinlichkeit, jedoch nicht die unausgewogene Verteilung der "erfolgreich" übertragenen Pakete. Es ist nicht klar, warum die Lastgeneratoren deutlich häufiger beim Zugriff auf das Übertragungsmedium gewinnen, als die Instanzen. Alle fünf Stationen pro Kollisionsdomäne (Lastgenerator, vier Instanzen) sind nach einer kurzen Anlaufphase ständig sendebereit. Der Lastgenerator setzt sich aber gegenüber den vier Instanzen häufiger durch (bei gleicher Fast Ethernet Hardware). Ein möglicher Erklärungsansatz wäre der sog. "Ethernet Capture"-Effekt (vgl. [193,210]): Aufgrund des "exponential backoff"-Algorithmus ist bei häufigen Kollisionen eine Instanz, die

das Medium erfolgreich belegen konnte, mit höherer Wahrscheinlichkeit wieder erfolgreich das Medium zu belegen, als eine Instanz, die zuvor schon einmal beim gemeinsamen Zugriff auf das Medium verloren hat.

2. Auch bei größeren Nachrichtenlängen (UDP-SDU > 384 Byte) sind die Lastgeneratoren in der Lage, die Kollisionsdomänen alleine vollständig auszulasten. Dennoch gelingt es den Packet Screen Instanzen, sich bei größeren Dateneinheiten gegenüber den Lastgeneratoren häufiger durchzusetzen (erkennbar an dem Einbruch der Lastkurve (`snd`) und dem Ansteigen der Leistungskurve (`rcv`)). Das Medium ist weiterhin vollständig ausgelastet, aber der Zugriff darauf ist deutlich ausgewogener.

Zusammenfassend ergeben sich somit deutliche Leistungssteigerungen durch die Parallelverarbeitung, insbesondere bei vielen konfigurierten Filterregeln, solange die Kommunikation unidirektional ist. Bei bidirektionaler Kommunikation ergeben sich teilweise extreme Durchsatzeinbrüche bei mittleren Datenlängen. Dies ist besonders mißlich, da die mittleren Datenlängen im normalen LAN/WAN-Verkehr (typischer Zugriffskontrollpunkt) besonders häufig auftreten.

4.5.2 Parallele Packet Screen in "switched" Fast Ethernet Netzen

Bereits in Abschnitt 4.4.2 wurde argumentiert, daß es wünschenswert ist, die parallele Packet Screen auch in "switched" Fast Ethernet Netzen zu betreiben. Durch den "full duplex"-Betrieb (Durchsatz von jeweils 100 Mbit/s in beide Kommunikationsrichtungen) ist der theoretische Durchsatz verdoppelbar.

Darüber hinaus zeigen die Meßergebnisse des vorigen Abschnitts, daß es auch zur Vermeidung der Kollisionen (die ursächlich für den Durchsatzeinbruch bei mittleren Nachrichtenlängen sind) erforderlich ist, den konkurrierenden Zugriff auf das Medium zu vermeiden. Die verbesserte parallele Packet Screen für "swiched" Fast Ethernet benutzt die in Abschnitt 4.4.2 diskutierten zusätzlichen Mechanismen, um eine Verteilung der Dateneinheiten über Switches zu ermöglichen. Durch den Einsatz von Switches bekommt jede Ethernet-Schnittstelle eine eigene Kollisionsdomäne. Kollisionen können nur noch als "output blocking" im Switch auftreten und werden dort durch Zwischenpuffer abgemildert. Der Erfolg ist in Abbildung 4.12 erkennbar. Auch hier besteht die Packet Screen aus vier Instanzen. Die beiden Lastgeneratoren generieren wiederum die akkumulierte Last (`snd Summe beider Richtungen`) und die Leistung ist wiederum als `rcv`-Kurve erkennbar. Das theoretische Durchsatzlimit ist bei dieser Meßreihe natürlich ebenfalls doppelt so hoch (100 Mbit/s pro Richtung).

Man erkennt, daß es bei der verbesserten Packet Screen zu keinen Leistungseinbrüchen kommt. Die Nachrichtenlänge hat kaum noch einen Einfluß auf die Filterleistung. Es werden bis zum Zusammenlaufen der Last- und der Leistungskurve beim Erreichen des theoretischen Limits durchgängig ca. 38.000 Dateneinheiten pro Sekunde gefiltert. Der Switch-basierte Ansatz ist somit ein voller Erfolg. Für große Nachrichtenlängen ist die

parallele Packet Screen sogar in der Lage, das theoretische Durchsatzlimit einzustellen (ab Nachrichtenlängen von 384 Byte) bis hin zu über 191 Mbit/s (bei 1472 Byte SDUs).

Darüber hinaus bestätigen die Ergebnisse, daß die oben gezeigten Leistungseinbrüche durch die Kollisionen verursacht werden. Durch den Switch-basierten Ansatz kann dieser Effekt vollständig ausgeschlossen werden.

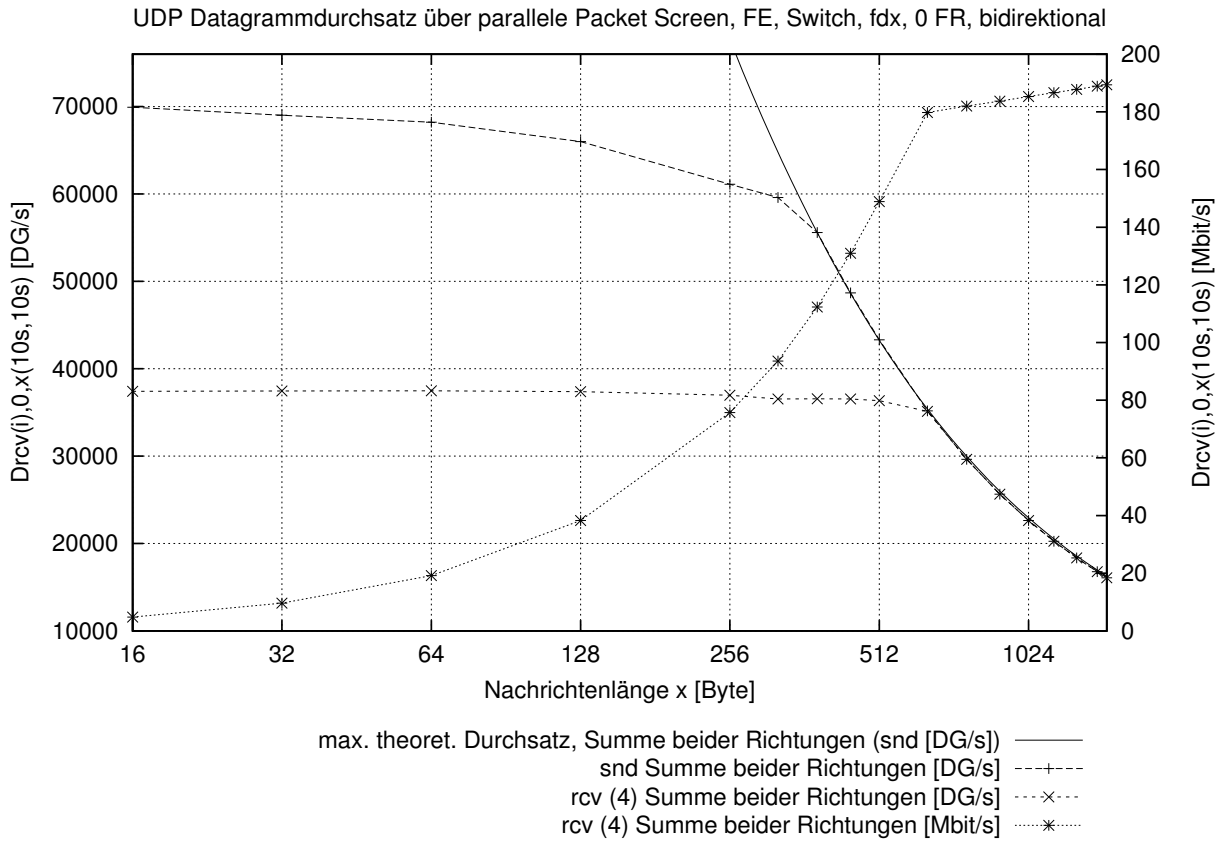


Abbildung 4.12: Durchsatzgewinn in Datagrammen pro Sekunde, "switched" FE-Netz, keine Filterregeln

4.5.3 Zusammenfassung: Meßergebnisse Parallele Packet Screen

Zusammenfassend zeigen die Messungen zur parallelen Packet Screen, daß das vorgestellte Konzept zur Lastverteilung durch die aktiven Netzkomponenten mit anschließender, verteilter Selektion der Aufträge auf den Instanzen sehr gut geeignet ist, um eine Beschleunigung der Filterverarbeitung zu erreichen. Dabei wird sowohl der absolut erreichbare Durchsatz deutlich verbessert, als auch der Durchsatz in Abhängigkeit von der Anzahl der konfigurierten Filterregeln: Je höher der Filteraufwand pro Dateneinheit ist, desto bes-

ser ist der Algorithmus geeignet, da der zusätzliche Selektionsaufwand immer weniger ins Gewicht fällt.

Die Verteilung ist sowohl über Hubs für "shared" Ethernet als auch über "Switches" für "switched" Ethernet möglich. Während die Hub-basierte Lösung leichter zu implementieren ist, ist die Switch-basierte Lösung deutlich leistungsfähiger und immer dann zu bevorzugen, wenn bidirektionale Kommunikation erforderlich ist bzw. beschleunigt werden soll. In Kapitel 10 werden verschiedene Einsatzszenarien für Komponenten diskutiert, die nach dem vorgestellten Prinzip eine Lastverteilung durchführen.

4.6 Analytische Bewertung der Parallelen Packet Screen

In diesem Abschnitt wird eine analytische Bewertung der Ergebnisse durchgeführt. Dazu ist es erforderlich, ein Modell der parallelen Packet Screen zu entwerfen, das die wesentlichen Aspekte der Paketverarbeitung beschreibt (vgl. Abschnitt 4.6.1). Diese gängige Vorgehensweise bei der Systemanalyse hat einerseits zum Ziel, die Meßergebnisse zu validieren ("Sind die Meßergebnisse erklärbar?"). Andererseits möchte man mit Modellen von der bei den Messungen verwendeten Hardware und Software abstrahieren, um davon losgelöste Aussagen (Prognosen) für andere Konfigurationen abzuleiten.

Es wird eine Formel zur Abschätzung des maximal erreichbaren Paketdurchsatzes einer aus n Instanzen bestehenden parallelen Packet Screen erstellt. Mit dieser Formel sollen die Meßergebnisse validiert werden.

Zwar zeigen die Messungen deutliche Leistungsgewinne, eine Aussage über mögliche Gewinne durch die Verwendung anderer Hardware oder durch die Verwendung von noch mehr Instanzen ist mit den Meßergebnissen alleine jedoch nicht möglich. Mit Hilfe des folgenden Modells soll daher nach der Validation der Meßergebnisse versucht werden, von der verwendeten Hardware zu abstrahieren, um weiterführende Erkenntnisse zu gewinnen.

4.6.1 Modell einer parallelen Packet Screen

Für die Modellbildung ist es erforderlich, einige wesentliche Aspekte/Attribute der Packet Screen mit eindeutigen Bezeichnungen zu beschreiben, die eine formale Darstellung erlauben. Im Zuge der weiteren Diskussion werden die folgenden Bezeichnungen benutzt:

- Die Anzahl der Instanzen einer parallelen Packet Screen wird mit $n \in \mathbb{N}$ bezeichnet. Eine bestimmte Instanz der parallelen Packet Screen mit PS_i, wobei für den Index ($i \in \mathbb{N}$) gilt: $0 \leq i < n$.

- Jede Instanz ist aus mehreren Funktionseinheiten aufgebaut. Es gibt eine Funktionseinheit (sel), die den Selektionsalgorithmus aus Abschnitt 4.3.5 ausführt. Der damit verbundene Aufwand wird mit B_{sel} $[s/DG]$ bezeichnet.[15]

- Jede Instanz verfügt über eine Funktionseinheit (fil), die die Zugriffskontrolle durchführt (die Filterregeln anwendet) und die Dateneinheiten ggf. weiterleitet oder vernichtet. Der in dieser Funktionseinheit anfallende Aufwand wird mit B_{fil} $[s/DG]$ bezeichnet.

- Jede Instanz verfügt über eine Datensenke (del), die Dateneinheiten vernichtet, die nicht weitergeleitet werden sollen. Der Aufwand für das Vernichten einer Dateneinheit im Zuge der Selektion wird mit B_{del} $[s/DG]$ bezeichnet.

- Die Packet Screen und jede ihrer Instanzen erreichen einen Durchsatz, der in Paketen pro Sekunde gemessen wird. Für die Leistungsbewertung wird ausschließlich der maximal erreichbare Durchsatz betrachtet, der mit $Pmax_i$ $[DG/s]$ und $Pmax_n$ $[DG/s]$ bezeichnet wird, je nachdem, ob der Durchsatz einer Instanz oder der Gesamtdurchsatz aller n-Instanzen gemeint ist.

- Der Gewinn durch den Einsatz der Parallelverarbeitung wird in Abhängigkeit von der Anzahl der eingesetzten Instanzen (n) im Vergleich zu einer einzelnen Instanz gemessen. Der Gewinn wird mit $G(n)$ (ohne Einheit) bezeichnet.

Eine parallele Packet Screen besteht aus n parallelen Instanzen ($PS_i, 0 \leq i < n$). Jede einzelne Instanz muß die Selektion durchführen, bei der die Pakete bestimmt werden, die von der jeweiligen Instanz bearbeitet werden müssen.

Bereits in Formel 4.8 wurde darauf hingewiesen, daß der zusätzliche Aufwand[16] für die Selektion (B_{sel}) deutlich kleiner als der Aufwand für die anschließende Paketverarbeitung (B_{fil}) sein sollte. Nur so ist sichergestellt, daß durch die Parallelverarbeitung auch ein Gewinn erzielt werden kann, der nicht gleich wieder durch den zusätzlichen Selektionsaufwand ausgeglichen wird.

Jede einzelne Instanz einer parallelen Packet Screen besteht somit aus drei wichtigen Komponenten: Eine Bedieneinheit für die Selektion von Paketen und eine dahintergeschaltete

[15] Im folgenden wird nur noch von Paketen bzw. "Protocol Data Units" (PDUs) als Sammelbegriff für beliebige Dateneinheiten gesprochen. Dies ist sinnvoll, da sich Filterregeln einer Packet Screen auf alle Protokollschichten beziehen können. Somit ist es von der jeweiligen Konfiguration abhängig, ob die Packet Screen die Filter auf Ethernet-Rahmen, IP/UDP-Datagramme, TCP-Segmente oder Dateneinheiten höherer Protokollschichten anwendet. Darüber hinaus wird zur Erhöhung der Übersichtlichkeit in den Tabellenüberschriften und den folgenden Formeln immer die Abkürzung "DG" für "Datagramm" benutzt, obwohl Pakete als Sammelbegriff für beliebige Dateneinheiten gemeint sind.

[16] Mit Aufwand ist im folgenden immer die Bedienzeitanforderung für die Bearbeitung eines Pakets gemeint. Die Einheit ist somit Sekunde pro Paket $[s/DG]$.

Bedieneinheit, die Filteroperationen, Routing und Weiterleitung der Dateneinheiten vornimmt. Zusätzlich gibt es eine Komponente für die unmittelbare Vernichtung von nichtselektierten Dateneinheiten. Der Fall, daß eine Dateneinheit erst im Zuge der Filteranwendung vernichtet wird, braucht hier nicht betrachtet zu werden.

Das Ziel der folgenden Modellierung ist es, Aussagen über den erreichbaren Gewinn durch die Parallelisierung zu errechnen. Es ist daher wichtig zu klären, wie daß Verhältnis zwischen B_{sel} und B_{fil} ist und welche Abhängigkeiten es von der Anzahl der parallelen Instanzen (n) und der Anzahl der konfigurierten Filterregeln (FR) gibt.

Bei dem in Abschnitt 42 diskutierten Algorithmus wird eine zufällige Verteilung der eintreffenden Dateneinheiten auf die parallelen Instanzen vorgenommen. Für die Modellierung wird der Idealfall angenommen, bei dem im Mittel jede Instanz etwa gleich viele Dateneinheiten verarbeiten muß. In der Abbildung 4.13 ist ein einfaches Modell einer parallelen Packet Screen aus n Instanzen aufgeführt, in das bereits die Wahrscheinlichkeiten (W) für die Selektion eines Paketes für die weitere Bearbeitung auf der Instanz eingetragen sind. Aufgrund der Annahme, daß durch den Algorithmus eine gleichmäßige Verteilung der Pakete auf alle parallelen Instanzen erreicht wird, muß bei einer n-parallelen PS jede Instanz eine von n Dateneinheiten verarbeiten.

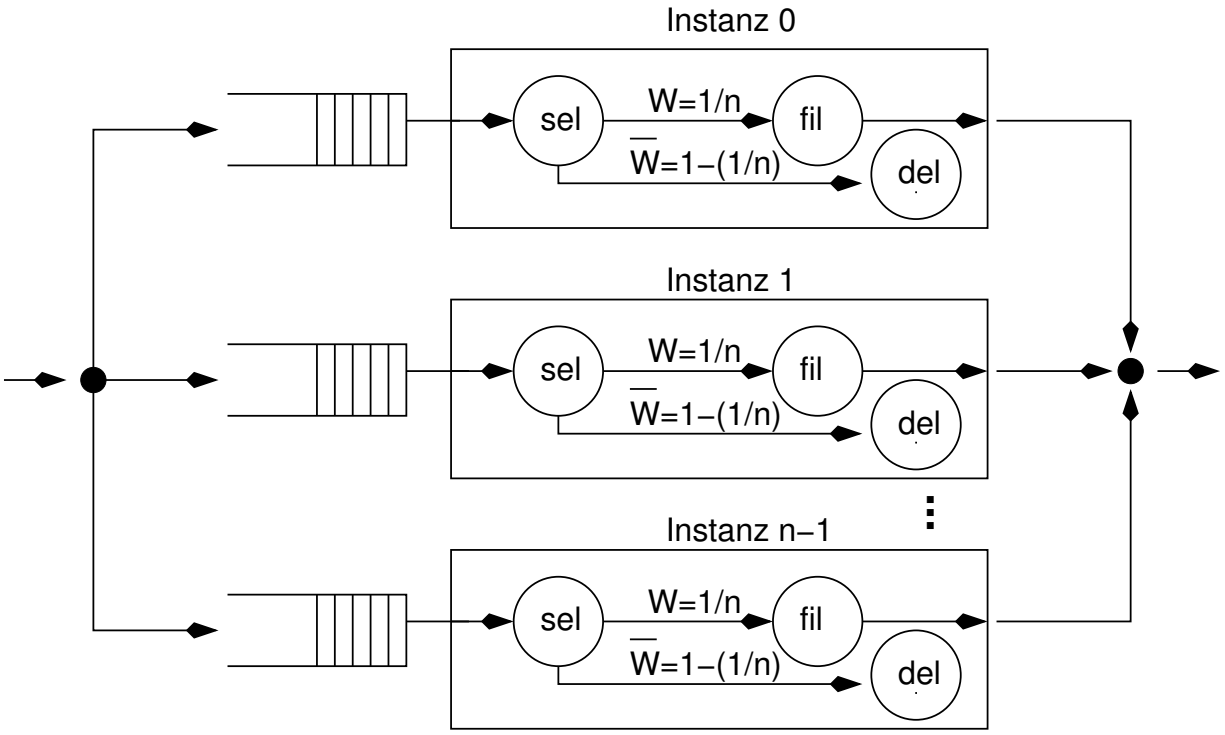


Abbildung 4.13: Modell einer parallelen Packet Screen

Die Selektionsoperation ("sel") ist der weiteren Verarbeitung (hier als "fil" markiert) vorgeschaltet. Durch Instrumentierung der Netz-Module der benutzten Workstations konnte

festgestellt werden, daß Paketverluste nach der Selektionsoperation auch unter hoher Last vernachlässigbar gering sind. Paketverluste treten daher nur während der Selektion durch Überlaufen von Empfangspuffern auf. Dies ist durch die angedeuteten Warteschlangen (mit begrenzter aber unbekannter Kapazität) angedeutet. Zwischen "sel" und "fil" ist keine weitere Warteschlange eingezeichnet. Dies liegt an der Eigenschaft des Kernels, durch eine Art Flußkontrolle[17] die "sel"-Komponente davon abzuhalten, Dateneinheiten zu filtern, solange die "fil"-Komponente noch mit einem Auftrag beschäftigt ist. Andererseits sichert diese Flußkontrolle auch zu, daß unter Last ein neuer Auftrag für die "fil"-Bedieneinheit sofort verfügbar ist, sobald der vorherige Auftrag verarbeitet wurde. Eine eintreffende Dateneinheit wird mit der Wahrscheinlichkeit von $W_i = 1/n$ für die weitere Verarbeitung auf der Instanz PS_i selektiert. Mit der Wahrscheinlichkeit $\overline{W_i} = 1 - (1/n)$ kann die Dateneinheit auf der Instanz PS_i vernichtet werden. Für dieses Modell wird angenommen, daß die Zeit für die Vernichtung der Dateneinheit vernachlässigbar (frei von Overhead) ist, und somit für den damit verbundenen Aufwand gilt: $B_{del} = 0s/DG$.

Die Netzkomponenten (Hubs und Switches), die jede Dateneinheit in alle n-Warteschlangen kopieren, sind hier nicht modelliert. Bei der Verwendung von modernen Switches stellen diese Netzkomponenten erfahrungsgemäß auch keinen Engpaß dar. In Anlehnung an Abbildung 3.9 stellen die schwarzen Punkte in Abbildung 4.13 die Schnittstellen IF_{sG} (links) und IF_{sM} (rechts) dar.

Außerdem wird bei der analytischen Modellierung die Abhängigkeit von der Datagrammlänge vernachlässigt. Dies scheint auch gerechtfertigt, da die empirischen Ergebnisse in weiten Bereichen (für kleine Dateneinheiten) kaum Abhängigkeiten von der Paketlänge aufzeigen.

4.6.2 Analytische Berechnung des Paketdurchsatzes

Der maximale Paketdurchsatz einer parallelen Packet Screen mit n gleichartigen Instanzen $(PS_0, \cdots, PS_{n-1})$ kann aufgrund des obigen Modells annähernd bestimmt werden.

Der maximale Paketdurchsatz ($Pmax_n[DG/s]$) der parallelen Packet Screen ist offensichtlich nach oben begrenzt (OS_n = obere Schranke für den Paketdurchsatz) durch

$$Pmax_n \leq \frac{1}{B_{sel}} = OS_n \tag{4.12}$$

Dieser Grenzwert drückt aus, daß nicht mehr Pakete verarbeitet als selektiert werden können. Wenn also alle selektierten Dateneinheiten anschließend vernichtet werden[18], dann kann die ganze Leistung aller Instanzen (Auslastung = 1 in Formel 4.12) für die Selektion

[17] Es handelt sich hierbei um die bei System VR4 Systemen übliche STREAM-basierte Flußkontrolle [229, S.558ff].

[18] Unter der Annahme, es gäbe eine entsprechende Filterregel mit vernachlässigbarem Zeitaufwand für die Paketvernichtung.

verbraucht werden. Da weiterhin Formel 4.8 gilt und nach Modell 4.13 mindestens die Selektion erfolgen muß, ergibt sich der in Formel 4.12 dargestellte Grenzwert.

Der Grenzwert ist jedoch nicht sehr aussagekräftig:

- Es sind keine Abhängigkeiten von der Anzahl n der parallelen Instanzen vorhanden. Vielmehr ist dies die obere Schranke für eine beliebige Anzahl von gleichartigen parallelen Instanzen, da der Selektionsaufwand auf allen Instanzen gleichermaßen anfällt.

- Der Grenzwert ist auch sehr unrealistisch. Normalerweise werden Dateneinheiten von einer PS nicht vernichtet, sondern im eingespielten Betrieb[19] durchgelassen. Das Vernichten aller Dateneinheiten ist eher die Ausnahmesituation, beispielsweise wenn für kurze Zeit die gesamte Netzkapazität von Dateneinheiten eines unerlaubten Dienstes beansprucht wird.

Es muß demnach das Verhältnis von selektierten zu weiterverarbeiteten Dateneinheiten betrachtet werden. Offensichtlich gilt für jede einzelne Instanz, daß für jede Dateneinheit, die in der "fil"-Komponente bearbeitet wird, zuvor im Mittel n Dateneinheiten von der "sel"-Komponente verarbeitet werden müssen. Damit ist der maximale Paketdurchsatz pro Sekunde auch wie erwartet abhängig von n, der Anzahl der Instanzen. Eine beliebige Instanz verrichtet unter maximaler Auslastung[20] ($U = 1$) somit die folgenden Tätigkeiten:

$$\begin{aligned} Pmax_i(n) * n * B_{sel} + Pmax_i(n) * B_{fil} &= U = 1 \\ Pmax_i(n) &= \frac{1}{nB_{sel} + B_{fil}} \end{aligned} \tag{4.13}$$

Der Gesamtdurchsatz aller n Instanzen ist somit:

$$Pmax_n(n) = Pmax_i(n) * n = \frac{n}{nB_{sel} + B_{fil}} \tag{4.14}$$

und für den erreichbaren Gewinn von n Instanzen gegenüber einer sequentiell arbeitenden Packet Screen ergibt sich unmittelbar:

$$G(n) = \frac{Pmax_n(n)}{Pmax_i(1)} = \frac{n(B_{sel} + B_{fil})}{nB_{sel} + B_{fil}} = \frac{B_{sel} + B_{fil}}{B_{sel} + (1/n)B_{fil}} \tag{4.15}$$

Formel 4.13 ist deutlich aussagekräftiger als Formel 4.12, da hier der zunehmende Selektionsaufwand durch weitere Instanzen als Teilaufwand der gesamten Paketverarbeitung

[19] Die Benutzer wissen in der Regel nach sehr kurzer Zeit, welche Dienste verwendet werden können und welche nicht. Der normale Betriebsmodus der PS ist somit das Weiterleiten von zulässigen Dateneinheiten und nicht das Filtern von unzulässigen Dateneinheiten.

[20] Die Auslastung (Utilization) ist hier mit U angedeutet.

einfließt und somit als Overhead durch die Parallelverarbeitung erkennbar ist. Außerdem besteht zwischen den beiden Formeln der folgende Zusammenhang:

$$\lim_{n\to\infty} Pmax_n(n) = \frac{1}{B_{sel}} = OS_n \tag{4.16}$$

Offensichtlich gilt auch bei Formel 4.13 die bereits in Formel 4.12 erkennbare obere Schranke für den Paketdurchsatz als Kehrwert des Selektionsaufwands. Die maximal erreichbare Leistungssteigerung ergibt sich direkt aus Gleichung 4.15 durch:

$$\lim_{n\to\infty} G(n) = \frac{B_{sel} + B_{fil}}{B_{sel}} \tag{4.17}$$

Durch Formel 4.15 ist es nun allerdings auch möglich zu entscheiden, ab wann es nicht mehr lohnt, weitere Instanzen zur parallelen Packet Screen hinzuzufügen. Wählt man beispielsweise als Abbruchbedingung, daß weniger als 10% Leistungssteigerung durch Hinzufügen einer weiteren Instanz erreicht werden, dann kann die folgende Ungleichung Aufschluß über ein geeignetes n geben:

$$\frac{Pmax_n(n+1)}{Pmax_n(n)} < 1,1 \tag{4.18}$$

Darüber hinaus ermöglicht Formel 4.15 eine Prognose über die erreichbare Durchsatzsteigerung in Abhängigkeit von der Anzahl der Instanzen.

4.6.3 Bestimmung der Werte für eine realistische Modellparametrisierung

Die beiden Größen B_{sel} und B_{fil} sind systemabhängig und müssen empirisch bestimmt werden. Da diese beiden Werte nur indirekt über die gemessene Leistung an der Schnittstelle IF_{pM} beobachtet werden können, muß vor der Kalibrierung des Modells geprüft werden, ob auf den Instanzen das Verhältnis zwischen der Anzahl der empfangenen Dateneinheiten und den anschließend gefilterten Dateneinheiten tatsächlich im Verhältnis $n : 1$ steht. Dies ist zwar die Modellannahme in den Formeln 4.13 und 4.14, aber eine genaue Überprüfung könnte auch zu einem anderen Ergebnis führen. Beispielsweise könnte es durch eine ungeschickte Flußkontrolle im Kernel der Packet Screen-Instanzen zu Paketverlusten nach der Selektion kommen. In diesem Fall wäre der Aufwand für die Selektion noch höher als im Modell angenommen, da der Aufwand für die anschließend verlorenen Dateneinheiten auf die tatsächlich gefilterten Dateneinheiten angerechnet werden müßte. Außerdem könnte sich herausstellen, daß der gewählte Hash-Algorithmus (vgl. Abschnitt 42) eine sehr unausgewogene Selektion der Dateneinheiten auf den Instanzen herbeiführt. Die parallele Packet Screen würde dann suboptimal arbeiten und nicht den in Formel 4.14 angegebenen Durchsatz erreichen können.

Für die direkte Messung des Verhältnisses von Selektions- zu Filteroperationen ist es im wesentlichen erforderlich, drei zusätzliche Zähler in den Geräte-Treiber für die Netzschnittstelle einzufügen:

- Ein Zähler C_{rx} für die Anzahl der empfangenen Dateneinheiten. Dieser Zähler notiert vor der Selektion, daß eine Dateneinheit vom Geräte-Treiber von der Netzschnittstelle in den Hauptspeicher geladen wurde.

- Ein Zähler C_{sel} für die Anzahl der selektierten Dateneinheiten. Dieser Zähler wird immer dann erhöht, wenn die Instanz eine Dateneinheit selektiert und an das Filtermodul weitergeleitet hat. Aus dem Verhältnis dieses Zählers zur Anzahl der empfangenen Dateneinheiten kann validiert werden, ob die Annahme bezüglich der gleichmäßigen Verteilung der Dateneinheiten gerechtfertigt ist. Es sollte nämlich für alle n auf einer beliebigen Instanz gelten: $n * C_{sel} \approx C_{rx}$

- Ein Zähler C_{tx} für die Anzahl der versendeten Dateneinheiten. Aus dem Verhältnis zur Anzahl der selektierten Dateneinheiten kann so überprüft werden, wieviele Dateneinheiten in der Instanz selbst verloren werden.

Die Geräte-Treiber wurden entsprechend erweitert. Durch zusätzliche Nachrichten an die Geräte-Treiber können diese Zähler wieder auf Null zurückgesetzt oder abgefragt werden.

Für die Berechnung von B_{sel} und B_{fil} wurde eine Meßreihe durchgeführt, bei der einer Instanz einer PS viele kurze Dateneinheiten (32 Byte Nutzdatenlänge zuzüglich UDP-, IP- und Ethernet-"Overhead") gesendet wurden. Die Instanz wurde nacheinander derart konfiguriert, daß die Zähler (C_{rx}, C_{sel} und C_{tx}) für $n = 1, 2, 3$ gemessen werden konnten. Bei den hiermit empirisch ermittelten Werten bezeichnet C_{tx} die Anzahl der Dateneinheiten, die im angegebenen Zeitintervall die Instanz wieder verlassen haben, also vollständig bearbeitet und somit auch gefiltert wurden. C_{rx} bezeichnet die Anzahl der Dateneinheiten, die in diesem Zeitintervall von der Instanz empfangen wurden. Dies ist somit die Gesamtanzahl der Dateneinheiten, die der "sel"-Komponente übergeben wurden, während C_{sel} die tatsächlich selektierten Dateneinheiten angibt. Bei den Messungen konnte festgestellt werden, daß die selektierten Dateneinheiten auch tatsächlich alle gefiltert wurden (minimale Abweichungen $< 1‰$). Dateneinheiten, die von der FE-Schnittstelle in den Kernel der Workstations kopiert wurden, werden demnach zu 100% abgearbeitet. Es kann für die weitere Diskussion daher angenommen werden, daß $C_{sel} = C_{tx}$ ist. Die folgenden Meßwerte C_{rx} und C_{tx} beziehen sich immer auf den Durchsatz einer Instanz und nicht auf den Gesamtdurchsatz der Packet Screen.

n	snd $[DG]$	C_{rx} $[DG]$	C_{tx} $[DG]$	C_{tx}/C_{rx}	$rcv(n)/n$ $[DG]$	Δt $[s]$
1	3987802	1670727	1670727	1	1669918	120
2	3884979	2950000	1474439	0,4998	1474204	120
3	3906147	3906150	1301850	0,33328	1300671	120

Tabelle 4.1: Empirische Werte für sel- und fil-Operationen, 32 Byte Benutzerdaten über UDP

Es fällt auf, daß das Verhältnis zwischen n, C_{rx} und C_{tx} sehr nah an das im Modell aus Abschnitt 4.6.2 angenommene Verhältnis von $n * C_{tx} = C_{rx}$ herankommt. Dies bedeutet, daß die Selektion im Mittel tatsächlich, wie zu erwarten, n Dateneinheiten bearbeitet, um eine dieser Dateneinheiten schließlich an die "fil"-Komponente weiterzugeben.

Darüber hinaus erkennt man Schwankungen in der Sendeleistung des Lastgenerators (*snd*) trotz der langen Meßdauer von $\Delta t = 2$ Minuten in Höhe von ca. 2%. Hierauf wird in Anhang C noch genauer eingegangen. Auf dem Weg zum Monitor kommt es nach dem Filtern und trotz der Verwendung von Switches auch noch zu Paketverlusten, denn es gilt:

$$\forall n : C_{tx} > \frac{rcv(n)}{n}, \text{ jedoch } \frac{C_{tx} - \frac{rcv(n)}{n}}{\Delta t} < 10 \text{ DG/s} \tag{4.19}$$

Formel 4.19 drückt zweierlei aus:

1. Es gibt die bei verbindungsloser Kommunikation üblichen, unentdeckten Paketverluste. Diese Paketverluste werden vom Monitor nicht von Verlusten durch die Überlastung des SuL unterschieden. Insofern ist die tatsächliche Filterleistung der Packet Screens sogar höher als in den Meßdaten und Graphiken (beispielsweise Abbildungen 4.9 und 4.12) zu erkennen ist.

2. Die Anzahl der Nachrichtenverluste ist LAN-typisch, d.h. sehr gering. Bei den obigen Messungen sind das weniger als 10 Pakete pro Sekunde, die zwischen SuL und Monitor verloren werden. Das entspricht aufgerundet vernachlässigbaren 1‰ des Gesamtverkehrs zwischen SuL und Monitor.

Aus der Gleichung 4.13 kann ein lineares Gleichungssystem erstellt werden, um beispielsweise für $n = 1$ und $n = 3$ die empirisch ermittelten Werte aus den Meßreihen für eine und drei parallele Instanz(en) (Anzahl der "sel"-Operationen) als $Pmax_i * n$ und $Pmax_i$ einzusetzen.

Da die empirischen Ergebnisse aus Tabelle 4.1 eine zufällige, gleichmäßige Verteilung der Dateneinheiten auf die angeschlossenen Instanzen bestätigen, können für die Lösung des linearen Gleichungssystems (s.u.) die idealisierten Werte für die Anzahl der Selektions- und Filteroperationen mit $\#sel = n * C_{tx}$ und $\#fil = C_{tx}$ angenommen werden. Darüber hinaus kann für die Bestimmung von C_{tx} der in Formel 4.19 angedeutete, geringe Fehler durch die indirekte Beobachtung an der Schnittstelle IF_{pM} vernachlässigt werden, so daß für die weitere Berechnung angenommen wird: $C_{tx} = \frac{rcv(n)}{n} = Pmax_i(n)$.

In Tabelle 4.2 sind die an der Schnittstelle IF_{pM} empirisch erhobenen Meßwerte für eine und drei Instanzen bei einer Nachrichtenlänge von 32 Byte angegeben. Die Nachkommastellen der Paketangaben ergeben sich durch die Normierung auf Pakete pro Sekunde.

Aus Formel 4.13 und den Werten aus Tabelle 4.2 ergeben sich die Gleichungen:

$$\begin{aligned} 13727,73\ DG * B_{sel} + 13727,73\ DG * B_{fil} &= 1s \\ 31553,13\ DG * B_{sel} + 10517,70\ DG * B_{fil} &= 1s \end{aligned} \tag{4.20}$$

n	$\#sel$ [DG]	$\#fil$ [DG]	Meßdauer [s]
1	13727,73	13727,73	1
3	31553,13	10517,70	1

Tabelle 4.2: Idealisierte Werte für sel- und fil-Operationen, 32 Byte Benutzerdaten über UDP

Löst man dieses lineare Gleichungssystem, so ergibt sich:

$$\begin{aligned} B_{sel} &\approx 1,11163E^{-5}\frac{s}{DG} \\ B_{fil} &\approx 6,1729E^{-5}\frac{s}{DG} \end{aligned} \tag{4.21}$$

B_{sel} ist tatsächlich deutlich kleiner als B_{fil}.

4.6.4 Modellbasierte Analyse des Gewinns durch Parallelverarbeitung

Durch die Bestimmung von B_{sel} und B_{fil} ist es nun auch möglich, Aussagen zum erreichbaren Gewinn durch die Parallelverarbeitung zu treffen.

Aus Formel 4.15 ergibt sich der mögliche Gewinn in Abhängigkeit von n. Dies ist in Tabelle 4.3 für $1 \leq n \leq 6$ dargestellt, wobei die mit einem Stern (*) markierten Werte für die Kalibrierung des analytischen Modells in den Formeln 4.20 und 4.21 verwendet wurden. Mit Formel 4.18 ergibt sich bei $n = 4$ durch Hinzunahme einer weiteren Instanz

#Instanzen (n)	$G(n)$
*1	1
2	1,735
*3	2,298
4	2,744
5	3,105
6	3,403

Tabelle 4.3: Gewinn durch Parallelverarbeitung

eine Steigerung von etwas mehr als 13%. Fünf parallele Instanzen sind somit noch "sinnvoll" bei der verwendeten Hardware einsetzbar. Eine weitere (sechste) Instanz bringt lediglich 10,9% Leistungssteigerung. Der maximal erreichbare Gewinn liegt nach Formel 4.17 bei ca. $6,55$. Der mögliche Gewinn ist in Abbildung 4.14 in Abhängigkeit von der Anzahl der parallelen Instanzen graphisch dargestellt.

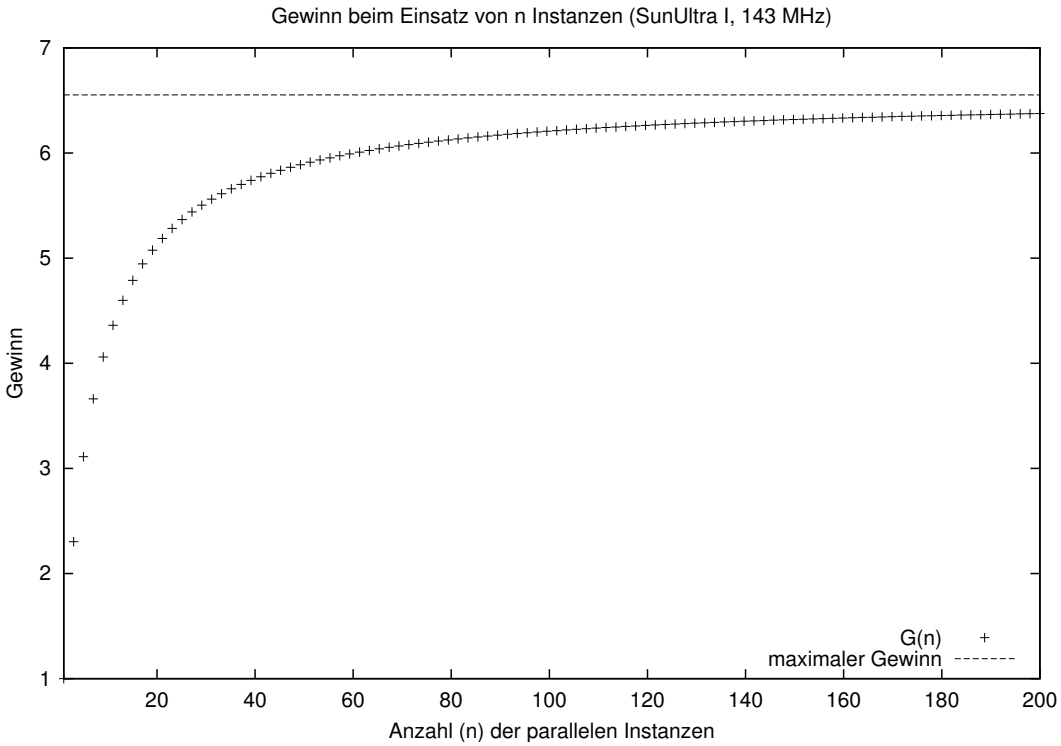


Abbildung 4.14: Potentieller Gewinn durch Parallelverarbeitung

Praktisch bedeutet dieses Ergebnis, daß beim Einsatz von sechs parallelen Packet Screen-Instanzen und einer daraus folgenden Leistungssteigerung von 340% ca. 46.719 Pakete pro Sekunde verarbeitet werden können. Dieser Durchsatz reicht aus, um für Pakete ab ca. 214 Byte Benutzerdaten den theoretisch möglichen Durchsatz auf einem Fast Ethernet zu erreichen. Die Prognose mit einer Paketlänge von 214 Byte ist gerechtfertigt, da Abbildung 4.12 für den Bereich von 32 bis 256 Byte kaum Abhängigkeiten von der Paketlänge aufzeigt.

4.6.5 Auswirkungen des Einsatzes schnellerer Hardware

Interessant ist die Antwort auf die Frage, ob der Einsatz schnellerer Hardware sinnvoll ist. Intuitiv ist schnellere Hardware sinnvoll, da sicherlich mehr Pakete als bei einer gleich großen Anzahl langsamerer Instanzen verarbeitet werden können.

Durch die obigen analytischen Ergebnisse sind jedoch genauere quantitative Aussagen möglich. Angenommen die eingesetzte Hardware ist doppelt so schnell und B_{sel} und B_{fil} sind jeweils nur noch halb so groß, dann ergibt sich als maximaler Paketdurchsatz nach Formel 4.13 von $Pmax_i fast(1) \approx 27.455 DG/s$. Dies ist wie erwartet doppelt so viel, wie der Durchsatz einer nur halb so schnellen Instanz. Interessanter ist allerdings, daß bereits zwei

dieser doppelt so schnellen Instanzen ausreichen, um in etwa den gleichen Durchsatz wie sechs Instanzen zu erreichen ($47.640 DG/s$ gegenüber $46.719 DG/s$, vgl. Abbildung 4.15).

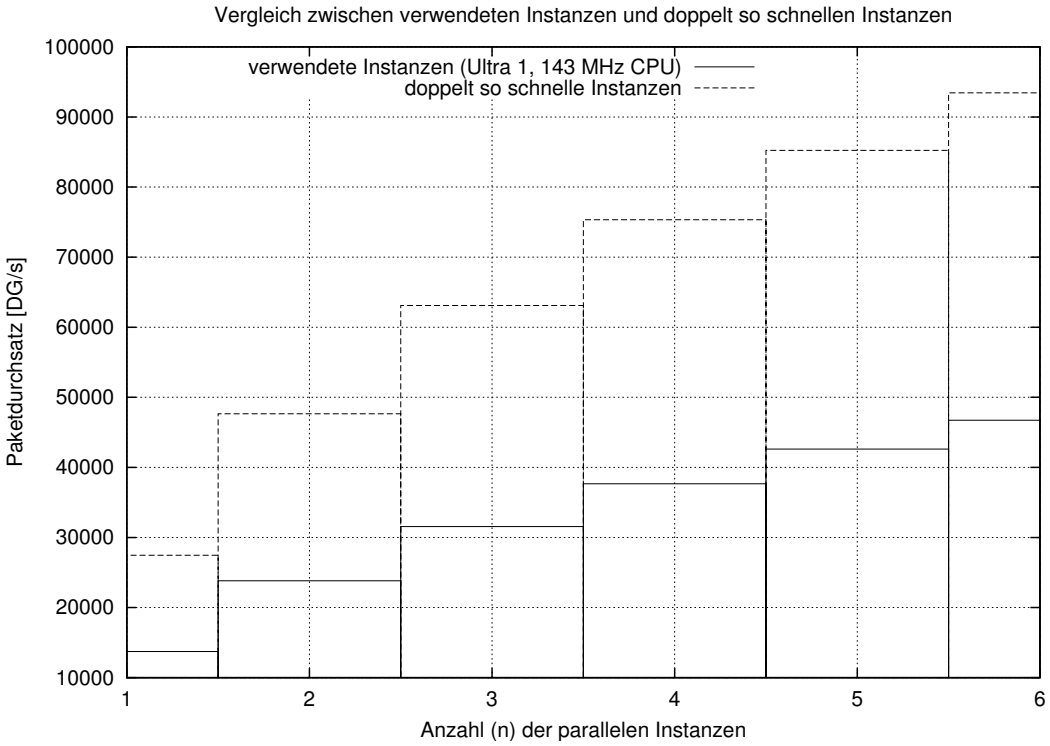


Abbildung 4.15: Paketdurchsatz im Vergleich mit doppelt so schnellen Instanzen

Die Interpretation dieses Ergebnisses liegt auf der Hand: Wenn möglich, besser einige schnelle Instanzen als viele langsame Instanzen einsetzen, da bei höherer Anzahl an Instanzen zunehmend Zeit für die Selektion verbraucht wird. Andererseits kann die Ausfallwahrscheinlichkeit der Packet Screen durch jede zusätzliche Instanz verringert werden (vgl. Kapitel 6).

4.6.6 Auswirkungen des Einsatzes aufwendiger Filteroperationen

Werden im Mittel viele Filteroperationen durchgeführt, dann verschiebt sich das Verhältnis von B_{sel} zu B_{fil}. B_{fil} wird größer, d.h. es können weniger Pakete pro Sekunde gefiltert werden. Interessant ist nun die Frage, ob sich hier der Einsatz weiterer paralleler Instanzen lohnen würde.

Durch zusätzliche Filterregeln wird der Aufwand für die Verarbeitung der selektierten Pakete (B_{fil}) größer. Der Selektionsaufwand ist hingegen unabhängig von der Anzahl der

Filterregeln. Es bietet sich daher an, die Abhängigkeit des Paketdurchsatzes von der Anzahl der Filterregeln in Formel 4.13 in B_{fil} auszudrücken. Man erhält, unter der Annahme einer linearen Abhängigkeit des Filteraufwands von der Anzahl der Filterregeln, somit Gleichung 4.22, in der B_{fil*} den Aufwand pro Filterregel für jedes Paket ($c[s/(DG * FR]$) berücksichtigt, so daß $B_{fil*} = B_{fil}$ gdw. die Anzahl der Filterregeln Null ($FR = 0$) ist.

$$Pmax_i(n, FR) = \frac{1}{nB_{sel} + B_{fil*}}, \quad B_{fil*} = B_{fil} + c * FR \tag{4.22}$$

Da B_{fil} und B_{sel} bereits bestimmt sind, können B_{fil*} und c mit Hilfe der empirischen Werte der Meßreihen bestimmt werden. Bei einer Meßreihe mit 100 Filterregeln wurde auf einer einzigen Instanz (keine Parallelisierung) ein Durchsatz von 4028 Paketen pro Sekunde erreicht. c ergibt sich somit zu:

$$\begin{aligned} 4028\ B_{sel} + 4028\ (B_{fil} + 100c) &= 1 \\ c &\approx 1,175E^{-6} \end{aligned} \tag{4.23}$$

Dieser Aufwand ist sehr gering (etwa $0,158B_{sel}$), fällt jedoch pro Filterregel bei jedem Paket an.

Bei 100 Filterregeln ergibt sich damit $B_{fil*} \approx 1,175E^{-4} + B_{fil}$. In Tabelle 4.4 ist der auf Basis dieses Filteraufwands berechnete Paketdurchsatz in Abhängigkeit von der Anzahl der Filterregeln im Vergleich mit dem gemessenen Durchsatz dargestellt. Die Abweichungen der Meßwerte von den berechneten Werten sind bei allen Meßpunkten kleiner als 5%, teilweise sogar erheblich genauer.[21] Das analytische Modell und die Meßergebnisse bestätigen sich somit gegenseitig.

Aus Gleichung 4.18 ergibt sich dann, daß es sich sogar lohnen würde, 8 Instanzen parallel zu schalten. $G(n_0 = 8, FR_0 = 100) \approx 6,09$ ist auch ein beträchtlicher Gewinn und bestätigt deutlich die Eignung des gewählten Algorithmus bei aufwendigen Filteroperationen.

In Abbildung 4.16 ist aufgrund der berechneten Werte für B_{fil}, B_{sel} und c eine Prognose für den erreichbaren Gewinn durch die Parallelverarbeitung dargestellt. Der Gewinn ist in Abhängigkeit von der Anzahl der konfigurierten Filterregeln (X-Achse, logarithmisch skaliert) und der Anzahl der parallelen Instanzen dargestellt. Es fällt auf, daß bei vielen konfigurierten Filterregeln und wenigen Instanzen beinahe ein linearer Gewinn erreicht wird. Bei wenigen konfigurierten Filterregeln lohnt sich der Einsatz von acht oder mehr Instanzen hingegen kaum. Es sei denn, man möchte neben einem hohen Gewinn auch eine Leistungstransparenz im Fehlerfall erreichen.

Fällt eine Instanz aus und es werden die in den folgenden Kapiteln vorgestellten Techniken zur automatischen Rekonfiguration eingesetzt, dann ist auch im Fehlerfall eine Leistungstransparenz gegeben. Der Ausfall von beispielsweise einer von 10 Instanzen bei 50 konfigurierten Filterregeln schmälert den Gewinn nur unmerklich um weniger als 6%.

[21] Die mit '–' markierten Werte wurden für die Kalibrierung des linearen Gleichungssystems aus Formel 4.20 bzw. für die Berechnung von 'c' mit Formel 4.23 benutzt. Daher kann es bei diesen Meßpunkten zu keinen Abweichungen zwischen berechnetem `rcv` und gemessenem `rcv` kommen.

n	FR	$rcv(n)$ $[DG/s]$	berechn. $rcv(n)$	Abweichung [%]
1	0	13728	13728	—
1	10	11420	11064	3,12
1	50	6538	6229	4,73
1	100	4028	4028	—
1	200	2398	2360	1,58
2	0	23662	23820	$< 1{,}0$
2	10	20045	19704	1,7
2	50	11888	11651	1,99
2	100	8067	7711	4,41
2	200	4613	4600	$< 1{,}0$
3	0	31553	31553	—
3	10	27623	26639	3,56
3	50	17228	16413	4,73
3	100	11417	11091	2,86
3	200	6661	6728	1,01

Tabelle 4.4: Gemessener vs. berechneter Datendurchsatz, 32 Byte UDP-Daten

4.6.7 Auswirkungen der Verbindungsparallelität

Wird die Selektion auf den einzelnen Instanzen auf Basis des in Abschnitt 5.2.4 vorgestellten Algorithmus durchgeführt, dann resultiert daraus eine sog. Verbindungsparallelität: Alle Dateneinheiten derselben Kommunikationsbeziehung werden von derselben Instanz bearbeitet. Die Vor- und Nachteile wurden bereits in Abschnitt 4.2.2 diskutiert. Der Hauptnachteil ist, daß abhängig von der Anzahl der zeitgleich zu bedienenden Kommunikationsbeziehungen einzelne oder mehrere Instanzen nicht an der Auftragsbearbeitung teilhaben können.

Im einzelnen sind drei Fälle zu unterscheiden, bei denen die Anzahl der zeitgleichen Kommunikationsbeziehungen (m) im Verhältnis zu der Anzahl der parallelen Instanzen (n) betrachtet werden können:

$n \geq m$: Ist die Anzahl der Instanzen gleich oder größer als die Anzahl der zeitgleich aktiven Kommunikationsbeziehungen, dann wird der bestmögliche Durchsatz erreicht, wenn die Hash-Funktion jede Kommunikationsbeziehung auf eine andere Instanz abbildet.

$n < m$: Ist die Anzahl der Instanzen kleiner als die Anzahl der zeitgleich aktiven Kommunikationsbeziehungen, dann sollte durch die Hash-Funktion eine möglichst gleichmäßige Aufteilung der Kommunikationsbeziehungen auf die aktiven Instanzen erfolgen. Hieraus erfolgt im Idealfall und bei ($n \ll m$) eine gleichmäßige Belastung aller Instanzen. In ungünstigeren Fällen kann es vorkommen, daß einer Instanz mehrfach hintereinander langandauernde Kommunikationsbeziehungen zugeordnet werden, so

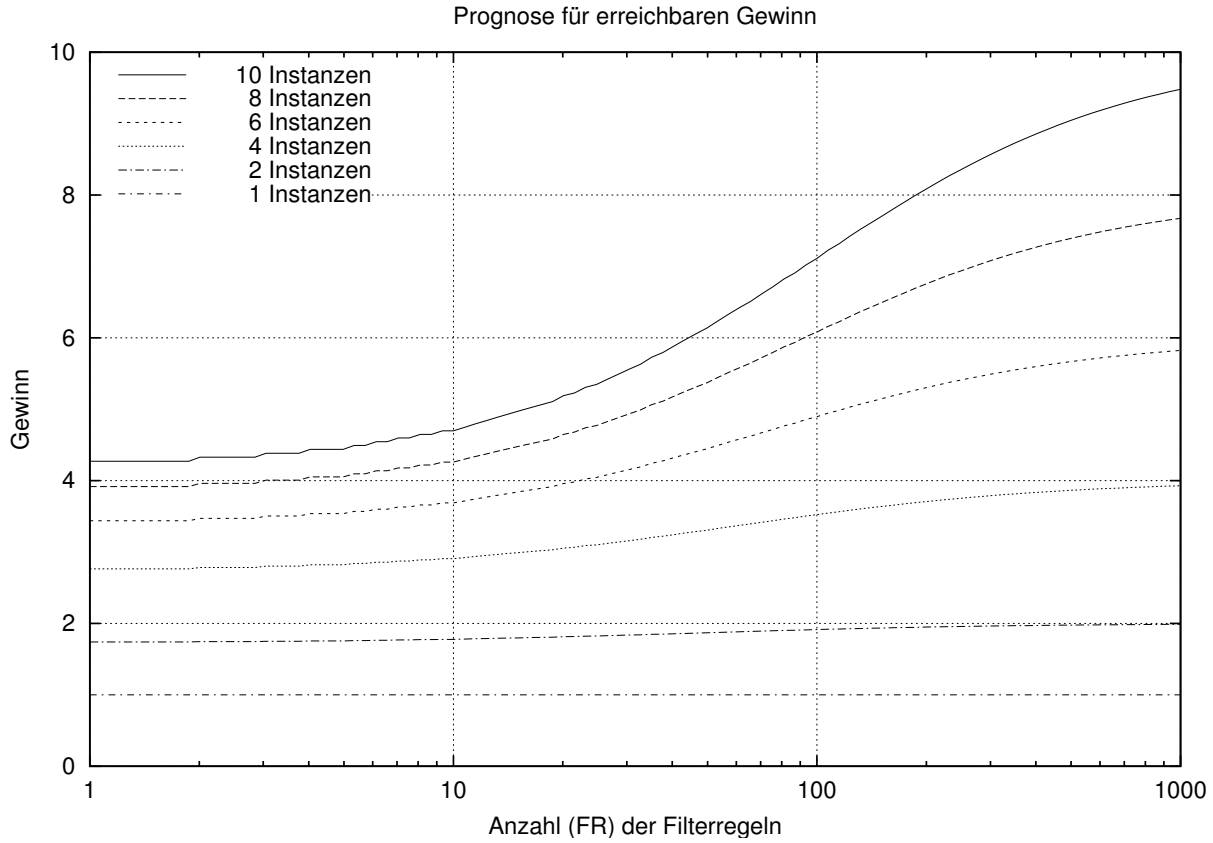


Abbildung 4.16: Prognose für Gewinn durch Parallelverarbeitung für 'Sun Ultra 1, 143 MHz CPU'–Instanzen

daß die Auslastung dieser Instanz im Mittel deutlich höher ist als die der anderen Instanzen. Diese Art von Problemen kann nur durch einen zentralen Verteiler gelöst werden, der abhängig von der aktuellen Last auf den Instanzen eine Verteilung vornimmt.

Die periodische Ermittlung der Belastung der einzelnen Instanzen würde aber ihrerseits zu einer Erhöhung der Last beitragen. Angesichts der Dominanz von eher kurz andauernden Verbindungen im Internet ist fraglich, ob sich dieser Aufwand lohnt. Abgesehen davon stellt ein zentraler Verteiler selbst einen potentiellen Engpaß und in jedem Fall einen neuen "single point of failure" dar (vgl. Abschnitt 4.3.3).

4.7 Zusammenfassung

In diesem Kapitel wurde ein dezentraler Algorithmus zur Beschleunigung der Paketverarbeitung auf Packet Screens vorgestellt. Basierend auf diesem Algorithmus konnte ein

Prototyp einer parallelen Packet Screen implementiert werden. Der Prototyp nutzt die Möglichkeit der ohnehin vorhandenen Netzhardware (Hubs, Switches) zur schnellen Verteilung der Last an alle parallelen Instanzen, um anschließend eine dezentrale Lastaufteilung vorzunehmen. Dadurch lassen sich zusätzliche Fehlerquellen und neue Engpässe durch zentrale Lastverteiler vermeiden.

Leistungsmessungen an der prototypischen Implementierung dieser parallelen Packet Screen zeigen eine ganz erhebliche Leistungssteigerung und belegen die Eignung dieses Konzeptes. Die Meßergebnisse zeigten deutlich, daß der Durchsatz vor allem von der Anzahl der konfigurierten Filterregeln bestimmt wird. Die Paketlänge spielt kaum eine Rolle.

Darüber hinaus konnte mit einem analytischen Modell dieser parallelen Packet Screen gezeigt werden, daß der Algorithmus um so bessere Leistungsgewinne erzielt, je aufwendiger der Filteraufwand pro Dateneinheit ist. Die durchgeführte, umfassende Validation des analytischen Modells bestätigte dessen außergewöhnlich hohe Realitätsnähe, die durch Verhaltensvergleiche von Modell und Realsystem (implementierter Prototyp) nachgewiesen werden konnte. Das analytische Modell dokumentiert seinerseits die uneingeschränkte Plausibilität der Meßergebnisse und ermöglicht aufgrund der erfolgreichen Modellvalidation somit Prognosen in Hinblick auf andere Hardware und/oder noch mehr parallele Instanzen.

Bevor im folgenden Abschnitt auf einige wichtige Variationsmöglichkeiten des vorgestellten Selektionsalgorithmus eingegangen wird, soll hier noch einmal betont werden, daß das erste Optimierungsziel, die Erhöhung der Leistungsfähigkeit, mit sehr gutem Erfolg erreicht worden ist. Der gewählte Ansatz ist darüber hinaus praxistauglich. Durch die einfache Replizierung der Filterfunktionen auf mehrere Instanzen und eine hiervon unabhängige Beschleunigung durch Verteilung und Selektion auf der darunterliegenden Netzebene ist die Parallelisierung einfach und skalierbar (mit geringen funktionalen Einschränkungen im Bereich des “stateful packet inspection”, vgl. Kapitel 5) für beliebige PS-Software realisierbar. Die parallele Packet Screen bleibt aufgrund dieser einfachen Möglichkeit zur Konfigurierung der Richtlinien (Kopieren der Filterfunktionen auf weitere Instanzen) darüber hinaus beherrschbar. Der Administrator braucht die Parallelisierung bei der Zusammenstellung der Richtlinien/Filterregeln nicht zusätzlich zu berücksichtigen.

Kapitel 5

Variationen des Selektionsalgorithmus als Basis für eine flexible Parallelverarbeitung

Der Selektionsalgorithmus bestimmt maßgeblich die Art der Parallelverarbeitung. In diesem Kapitel wird gezeigt, wie durch Variieren dieses Selektionsalgorithmus das Einsatzspektrum der parallelen Packet Screen erweitert werden kann. Bei der Implementierung des Prototypen wurde eine sehr einfache Form des Selektionsalgorithmus verwendet. Die Verteilung wird beim gewählten Algorithmus lediglich in Abhängigkeit von der Anzahl der parallelen Instanzen der Packet Screen und der IP-Prüfsumme vorgenommen (vgl. Abschnitt 4.3.5). Dies ist in der Praxis aus mehreren Gründen nicht immer wünschenswert. In den folgenden Abschnitten werden daher wichtige Variationen des Selektionsalgorithmus vorgestellt. Abschnitt 5.1 beschreibt eine Anpassung des Algorithmus an heterogene Hardware. Abschnitt 5.2 beschreibt die Notwendigkeit, für bestimmte Filteraufgaben eine Selektion gemäß der Verbindungsparallelität (vgl. Abbildung 4.5) durchzuführen und die dafür erforderlichen Anpassungen des Algorithmus.

5.1 Unterschiedliche Leistungsfähigkeit der Prozessoren

Eine implizite Annahme in Abschnitt 4.3.5 ist, daß alle Prozessoren (Workstations) gleich leistungsfähig sind. Die Annahme ist für die oben vorgestellten Implementierungen richtig, da gleichartige Workstations verwendet wurden (vgl. Abschnitt 3.4).

In der Praxis ist es jedoch häufig wünschenswert, heterogene Geräte verschiedener Leistungsklassen zu verwenden.[1] In diesem Fall sollte der Selektionsalgorithmus an die Leistungsfähigkeit der Geräte angepaßt werden. Das Ziel der Anpassung muß es sein, die Anzahl der Pakete entsprechend der relativen Leistungsfähigkeit der einzelnen Prozessoren zuzuordnen. Durch diese Zuordnung kann verhindert werden, daß unter hoher Last die leistungsschwachen Prozessoren Dateneinheiten verlieren[2], während es auf den leistungsstärkeren Prozessoren zu Leerlauf kommt.

Voraussetzung hierfür ist eine Leistungsbewertung der zu verwendenden Maschinen. Eine einfache Leistungsbewertung könnte in Form einer Lastmessung erfolgen, bei der der maximale Paketdurchsatz einer Maschine pro Sekunde ermittelt wird. Die Leistungsbewertung liefert dann eine relative Leistungsfähigkeit in Prozent (%), bei der 100% dem Gesamtdurchsatz aller zu verwendenden Prozessoren entspricht.

Der Selektionsalgorithmus sollte die Dateneinheiten entsprechend der relativen Leistungsfähigkeit selektieren. Dies kann beispielsweise durch eine Aufteilung des Wertebereiches der Hash-Funktion entsprechend der prozentualen Leistungsfähigkeit erreicht werden. Die Hash-Funktion muß demnach Werte im Bereich $[1, \cdots, 100]$ liefern.

Die einzelnen Prozessoren werden zusätzlich mit einer Tabelle der relativen Leistungsfähigkeit konfiguriert, so daß durch einfache Tabellenzugriffe eine Selektion erfolgen kann. Der Algorithmus läßt sich im Pseudocode folgendermaßen wiedergeben:

```
/* Folgende globale Definitionen sind
 * auf allen Prozessoren zu konfigurieren
 */

/* Anzahl der parallelen Prozessoren = n,
 * IDs von 0 bis (n-1)
 */
#define ANZAHL n

/* Tabelle mit Prozentangaben zur
 * relativen Leistungsfaehigkeit
 */
const int reltab[ANZAHL+2] = {
  0,
  R0,
  R0+R1,
  R0+R1+R2,...,
  100-(Rn-1),
  100};
```

[1] Die schnell voranschreitende Entwicklung neuer Hardware und die damit verbundene Ersetzung alter Geräte durch neue Systeme führt sehr schnell zu heterogenen Konstellationen.

[2] Verluste treten beim Überlaufen von Warteschlangen im Kommunikationssubsystem der Rechner, die eintreffende Dateneinheiten nicht schnell genug verarbeiten können, auf.

```
20
   /*
    * myid       : Nummer des Prozessors
    * Datagramm : Die zu verarbeitende Dateneinheit
    *
25  * Rueckgabewert: 0 == nicht selektiert
    *                1 == selektiert
    */

   int  select  (int myid, struct datagram* ip)
30 {
     int hashval;

     hashval = ntohs(ip->header_check_sum) % 100;
     if  ( ( reltab[myid] <= hashval) &&
35         ( hashval < reltab[myid +1]) ){
       return 1;
     }
     else{
       return 0;
40   }
   }
```

Bei n parallelen Prozessoren (hier immer Workstations) wird die relative Leistungsfähigkeit in R_0 bis R_{n-1} angegeben. Die Summe über alle R_i ist 100% ($\sum_{i=0}^{n-1} R_i = 100\%$). Die Hash-Funktion liefert nun Werte unabhängig von der Anzahl der Prozessoren im Wertebereich $[1, \cdots, 100]$. Die einzelnen Workstations prüfen, ob der gelieferte Wert in ihrem Intervall liegt. Dazu sind zwei Tabellenzugriffe erforderlich, die in der **if**()-Abfrage in den Zeilen 34 und 35 kodiert sind.

Durch diese einfache Änderung des Selektionsalgorithmus ist nun auch die Verwendung von unterschiedlich leistungsfähiger Hardware möglich. Denkbar wäre sogar der Einsatz von verschiedenartiger PS-Software. Bei einer solchen Diversifizierung kann nicht davon ausgegangen werden, daß die Filterregeln auf allen Instanzen gleich schnell abgearbeitet werden. Auch in diesem Fall kann eine Anpassung der ungleichen Verarbeitungsgeschwindigkeiten der verschiedenen PS-Programme durch den oben vorgestellten Algorithmus erfolgen.

5.2 Dynamisches Filtern und Verbindungsparallelität

Eine Packet Screen, die *dynamisches Filtern* unterstützt, ist in der Lage, aufgrund von eintreffenden Dateneinheiten und konfigurierten Regeln, die Liste der Filterregeln dynamisch zu verändern. Diese Technik wird vor allem für zwei Anwendungsfälle eingesetzt:

Zugriff auf RPC Dienste: Ein "Remote Procedure Call" (RPC) ist ein essentieller Mechanismus in verteilten Systemen [53, S. 127ff]. Häufig ist der freie Zugriff auf RPC-

Dienste eine wesentliche Kommunikationsanforderung[3], die bei der Installation eines Firewalls beachtet werden muß. Zugriffskontrolle auf RPC-basierte Dienste erfordern auf Packet Screens dynamisches Filtern.

Rückkoppelung mit IDS-System: Ein "Intrusion Detection System" kann zusätzlich zu einem Firewall installiert werden, um einen erfolgreichen Einbruch oder einen gefährlichen Einbruchsversuch zu erkennen. Das IDS-System kann dann den Firewall beauftragen, durch eine Anpassung der Filterregeln die Gefahr abzuwenden bzw. abzumildern (vgl. "Recognition", Abschnitt 2.2.3).

5.2.1 Dynamisches Filtern für "Remote Procedure Call"-Zugriffskontrolle

Das Kernproblem bei der Zugriffskontrolle auf "Remote Procedure Call, RPC"-Dienste durch eine Packet Screen besteht in der dynamischen Zuordnung der RPC-Dienste auf einem Endsystem zu freien Dienstkennungen (bei TCP/UDP: Portnummern). Ein neuer Dienst wählt dazu eine freie Portnummer auf dem jeweiligen System und meldet sich anschließend mit der Portnummer und einem eindeutigen Bezeichner bei einem Koordinator ("Portmapper") an.

Clients müssen bei einem RPC-Aufruf in zwei Schritten vorgehen. Zuerst wird der Portmapper kontaktiert, um über einen eindeutigen Bezeichner die Portnummer des eigentlichen Dienstes zu erfragen. Der Portmapper teilt dem Client den gewünschten Dienstzugangspunkt mit und der Client kontaktiert den Server über den neu erlernten Dienstzugangspunkt. Eine ausführliche Darstellung dieser Konzepte ist beispielsweise in [30, 52] zu finden und braucht hier nicht vertieft zu werden. Entscheidend ist, daß dem Dienst nach einem Neustart (Ursachen beispielsweise: Timeout, Reboot, Störung, etc.) möglicherweise vom Betriebssystem eine andere Portnummer zugeordnet wird. Mit dieser neuen Portnummer meldet sich der Dienst erneut beim Koordinator an. Der Zugriff auf zulässige RPC-Dienste kann daher auf der Packet Screen nicht durch Konfigurieren von statischen Filterregeln mit a priori bekannten Portnummern freigeschaltet werden.

Um dennoch auf ungefährliche oder erforderliche[4] RPC-Dienste zugreifen zu können, hat sich in der Praxis folgende Vorgehensweise etabliert:

1. Ein Client kontaktiert den "Portmapper" auf einem Server und erfragt über den eindeutigen Bezeichner die Portnummer des gewünschten Dienstes.

[3] Es sprechen eine Reihe von Sicherheitsrisiken gegen die Verwendung von RPCs über einen Firewall. Es gibt jedoch Anforderungen, die nur durch den Einsatz von RPC-Diensten erfüllt werden können.

[4] Es gibt häufig einen Konflikt zwischen Sicherheitsanforderungen und Kommunikationsanforderungen. Dieser "Tradeoff" soll durch die etwas umständliche Formulierung angedeutet werden.

2. Die Packet Screen analysiert die Antwort des "Portmappers" und merkt sich die Adresse von Client und Server sowie die Portnummer des registrierten RPC-Dienstes dieses Servers.

3. Der Client kontaktiert den RPC-Dienst. Die Packet Screen empfängt zuvor die Dateneinheiten.

4. Die Packet Screen erkennt die Senderadresse (SA) und die Sendeportnummer (SP) und die Empfängeradresse (EA) und Empfangsportnummer (EP), die der Client (C) in der Dateneinheit angegeben hat. SA, EA und EP entsprechen den in Schritt 2 gelernten Informationen.

5. Die Packet Screen erzeugt Filterregeln, die Dateneinheiten von $\{SA_C, SP_C\}$ nach $\{EA_C, EP_C\}$ passieren lassen und Filterregeln, die Antworten des Servers an den Client passieren lassen: $\{SA_S = EA_C, SP_S = EP_C\}$ nach $\{EA_S = SA_C, EP_S = SP_C\}$.

6. Die Packet Screen startet einen Zeitgeber, der den Ablauf der Gültigkeitsdauer dieser Regeln signalisiert.

7. Nach Ablauf des Zeitgebers entfernt die Packet Screen die generierten Filterregeln wieder. Hiermit wird ein fortschreitendes "Durchlöchern" der Firewall durch anhaltende RPC-Zugriffe vermieden.

Hieraus ergibt sich bei der parallelen Packet Screen ein Problem, wenn der in Abschnitt 42 vorgestellte Algorithmus zur Selektion verwendet wird. Die dynamisch generierten Filterregeln werden nicht auf allen Instanzen gleichzeitig konfiguriert, sondern nur auf der Instanz, die zufällig die Antwort des Koordinators und die Dateneinheit des Clients mit $(\{SA_C, SP_C\}, \{EA_C, EP_C\})$ empfängt. Ist dies nicht die selbe Instanz, wird bereits die Dateneinheit aus Schritt 3 verworfen. Auch wenn diese Nachricht den Server erreichen sollte, kann bereits nach der ersten Dateneinheit eine Antwort vom Server kommen ($\{SA_S = EA_C, SP_S = EP_C\}$ nach $\{SA_C, SP_C\}$), die potentiell von einer anderen Instanz selektiert wird. Diese Instanz hat jedoch keine Anpassungen vorgenommen und würde die Dateneinheit filtern (z.B. vernichten). In Abbildung 5.1 ist dieser Ablauf noch einmal graphisch ab Schritt 3 für eine PS mit zwei Instanzen dargestellt. Eine Einführung in die dort verwendete, graphische Notation befindet sich in Anhang A. Mit $DG(A, B, C, D, DATA)$ wird ein Datagramm mit der Nachricht $DATA$ bezeichnet, das von dem Rechner mit IP-Adresse A und Portnummer B an den Rechner mit IP-Adresse C und Portnummer D gesendet wird.

5.2.2 Dynamisches Filtern für "Intrusion Detection System"-Rückkoppelung

Der zweite wichtige Einsatzbereich für dynamisches Filtern ist die Anpassung der Zugriffsrechte durch eine IDS- oder Monitor-Komponente. Immer wenn das IDS-System "ver-

dächtige" Netzaktivitäten registriert, kann es den Firewall instruieren, den Initiator dieses Verkehrs durch Ändern der Filterregeln von weiteren Zugriffen auf das Netz abzuhalten. Dieser Fall ist unkritisch. Durch die oben dargestellten "Multicast"-Mechanismen kann eine IDS-Komponente alle Instanzen erreichen. Dieser Fall braucht daher nicht weiter behandelt zu werden.

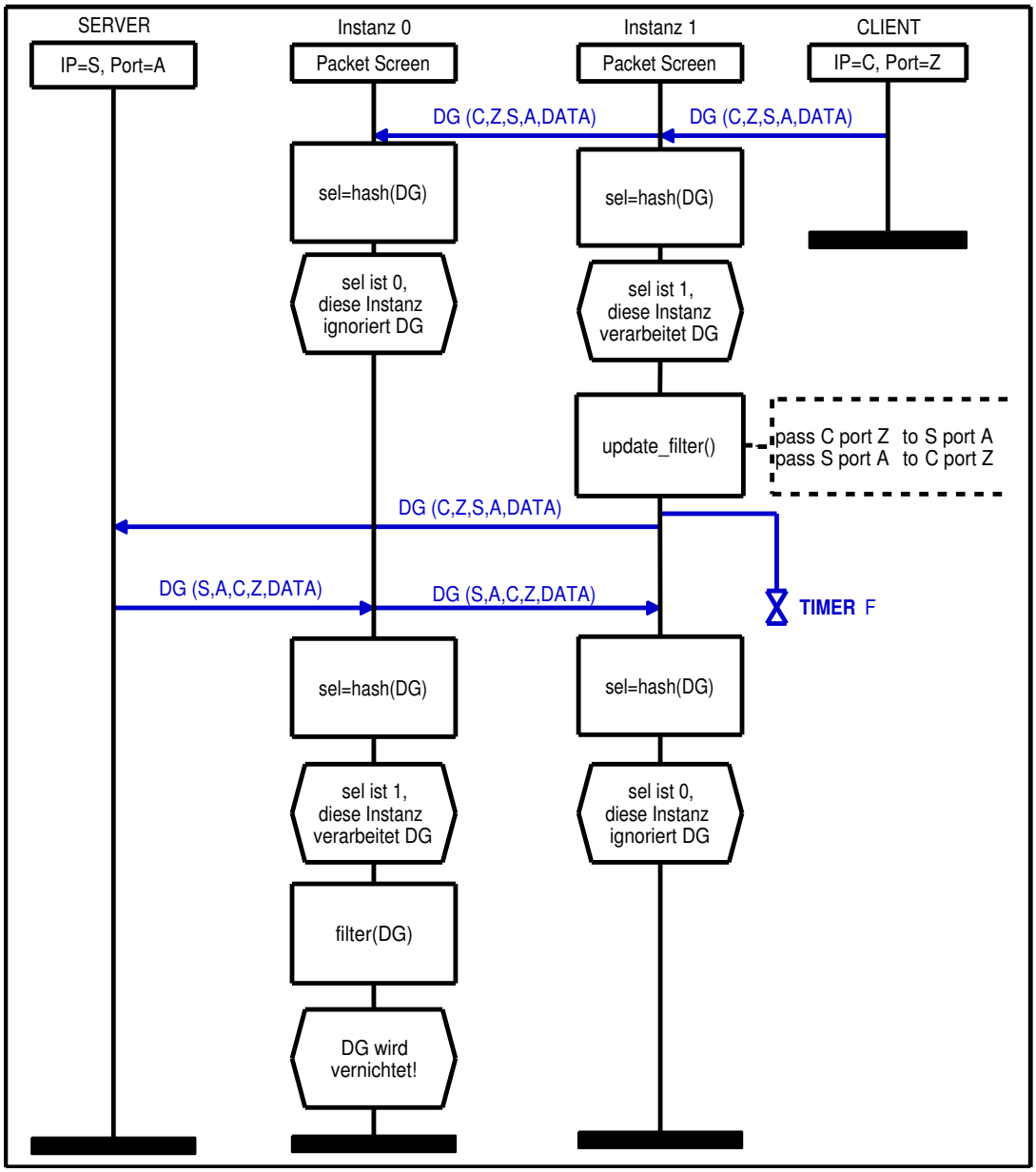


Abbildung 5.1: Probleme beim dynamischen Filtern

5.2.3 Anforderung: Unterstützung dynamischer Filter

Die in Abschnitt 5.2.1 beschriebenen Probleme entstehen durch die völlig unabhängige Verteilung der Dateneinheiten auf alle parallelen Instanzen. Um sicherzustellen, daß nur die Instanz Antworten eines Servers selektiert, die zuvor auch die Anfrage des Clients bearbeitet hat, ist es erforderlich, den Selektionsalgorithmus an "Kommunikationsbeziehungen" anzupassen.

Eine Kommunikationsbeziehung ist durch die Dienstzugangspunkte (SAPs) in den miteinander kommunizierenden Endsystemen eindeutig festgelegt. In der TCP/IP-Protokollfamilie ist eine Kommunikationsbeziehung durch Angabe von Paaren[5] aus IP-Adresse und Portnummer bestimmbar. Bei der Verwendung von TCP als Transportprotokoll entspricht die Kommunikationsbeziehung der TCP-Verbindung.

Ein Selektionsalgorithmus, der dynamisches Filtern unterstützt, muß daher eine eindeutige Zuordnung aller Dateneinheiten einer Kommunikationsbeziehung zu derselben Instanz vornehmen. Dadurch ist sichergestellt, daß alle zur Anfrage des Clients gehörenden Dateneinheiten von derselben Instanz selektiert werden und alle dazugehörenden Antworten des Servers ebenfalls von dieser Instanz bearbeitet werden. Da diese Instanz aufgrund der ersten Dateneinheit der Client-Anfrage dynamisch die Filterregeln angepaßt hat, können Anfragen und Antworten nun ungehindert den Firewall passieren.

Die in Abschnitt 42 diskutierte Hash-Funktion muß daher so geändert werden, daß sie als Eingabewerte die SAPs der Kommunikationsbeziehung benutzt.

5.2.4 Algorithmus zur Unterstützung dynamischer Filter

Eine einfache Erweiterung des Algorithmus aus Abschnitt 42 ermöglicht die Selektion von Dateneinheiten entsprechend der Zugehörigkeit zu einer Kommunikationsbeziehung.

Der Algorithmus muß eine Zuordnung zu den Instanzen in Abhängigkeit von den "SAP"-Informationen (IP-Adresse und Portnummer) vornehmen. Da diese Informationen in Abhängigkeit von der Senderichtung in den Dateneinheiten kodiert sind, muß eine Sortierung der SAP-Informationen vor dem Hashing erfolgen. Durch die Sortierung wird erreicht, daß Client-Anfragen und Antworten des Servers derselben Instanz zugeordnet werden, da die Symbole (hier die SAPs) in der gleichen Reihenfolge von der Hash-Funktion verarbeitet werden:

```
/* Folgende globale Definitionen sind auf
 * allen Prozessoren zu konfigurieren
 */

/* Anzahl der parallelen Prozessoren = n,
```

[5] Bei einer Punkt-zu-Punkt-Kommunikationsbeziehung gibt es genau zwei SAPs oder auch "Sockets" im TCP/IP-Jargon. Bei einer Multicast-Kommunikationsbeziehung gibt es SAPs entsprechend der Anzahl der Teilnehmer.

```
 * IDs von 0 bis (n-1)
 */

#define ANZAHL n

/* Die ID des jeweiligen Prozessors */
#define MYID x


/* Prototyp einer Hash-Funktion,
 * die SAPs als Parameter erhaelt */
extern int hash (unsigned long ip1, unsigned short port1,
      unsigned long ip2, unsigned short port2);



/*
 * myid       : Nummer des Prozessors
 * Datagramm : Die zu verarbeitende Dateneinheit
 *
 * Rueckgabewert: 0 == nicht selektiert
 *                1 == selektiert
 */

int select (int myid, struct datagram* ip)
{
  int hashval;
  unsigned long ips, ipe;
  unsigned short ports, porte;

  /* vereinfachte Darstellung fuer den Zugriff auf
   * die SAP-Informationen einer IP-Dateneinheit
   * mit eingebetteter UDP- oder TCP-Dateneinheit!
   */
  ips = ntohl(ip->source_addr), ipe = ntohl(ip->destination_addr);
  ports = ntohs(ip->source_port), porte = ntohs(ip->destination_port);

  if ( ips < ipe )
    hashval = hash(ips, ports, ipe, porte) % ANZAHL;
  else
    hashval = hash(ipe, porte, ips, ports) % ANZAHL;
  if ( hashval == myid ){
    return 1;
  }
  else{
```

```
    return 0;
  }
}
```

Ein Beispiel für eine Hash-Funktion, die nach Angaben in [6] eine sehr ausgewogene Verteilung erreicht, ist hier in leicht angepaßter Form in Pseudocode widergegeben:

```
/* Die folgende Hash-Funktion ist angelehnt
 * an "hashpjw()" aus:
 * Aho, A.V. et al.: "Compilerbau, Teil 1"
 * , Addison-Wesley 1988, S. 535
 */

#define PRIME 211

/* Paket-Attribute fuer Selektion
 * identifizieren eine Kommunikations-
 * beziehung.
 */
struct saps {
  unsigned long ips;
  unsigned long ipe;
  unsigned short ports;
  unsigned short porte;
};

int hash (unsigned long ip1, unsigned short port1,
    unsigned long ip2, unsigned short port2)
{
  struct saps kommbez;
  char *p;
  int c, nbyte=sizeof(struct saps);
  unsigned long h, g;

  /* Ausrichten der Daten im Speicher:
   * Beachte: Funktionsparameter koennten
   * auch in Registern uebergeben werden.
   * Hashing erfordert die Ausrichtung im
   * Speicher als struct saps !!!
   */
  kommbez.ips = ip1;
  kommbez.ipe = ip2;
  kommbez.ports = port1;
  kommbez.porte = port2;


  for (p = (char *)&kommbez, c = 0, h = 0; c < nbyte; p++, c++)
```

```
    {
      h = (h ■ 4) + (*p);
      if (g = (h & 0xf0000000))
  {
45    h = h ^ (g ■ 24);
      h = h ^ g;
  }
    }
  return (h % PRIME);
50 }
```

5.2.5 Weitere Anwendungsmöglichkeiten der Verbindungsparallelität

Die oben diskutierten Beispiele sind wichtige Anwendungen der Verbindungsparallelität für Packet Screens. Die Selektion auf Basis der Kommunikationsbeziehungen durchzuführen, ermöglicht jedoch jenseits der Packet Screen weitere Anwendungen. Auf diese Anwendungen wird allerdings erst in Kapitel 10 eingegangen. Erst dann kann aufgrund der noch offenen Ergebnisdarstellung der Transfer der vorgestellten Selektionsmethoden in andere Anwendungsfelder erfolgen.

5.2.6 Mehrstufige Selektion

Die Selektion kann auch mehrstufig erfolgen. Beispielsweise kann in einer ersten Selektion, basierend auf dem Algorithmus aus Abschnitt 5.2.4, eine Vorauswahl getroffen werden, die die Kommunikationsbeziehungen verschiedenen Gruppen zuordnet. Diese Zuordnung könnte Kommunikationsbeziehungen entsprechend der betroffenen Sicherheitsdomänen Gruppen von Instanzen zuordnen.

Innerhalb der Gruppen könnte mit einer zweiten Selektion eine feinere Unterteilung auf die Gruppenmitglieder erfolgen, beispielsweise indem anhand der Dienstkennungen eine Zuordnung entsprechend der Leistungsfähigkeit der Instanzen erfolgt. So würde es sich anbieten, interaktive Sitzungen (Telnet oder “Internet Relay Chat”) an leistungsschwache Instanzen zu delegieren, während Dienste mit (voraussichtlich) hohem Datenvolumen und “burst”-artigem Verkehrsaufkommen den leistungsfähigeren Instanzen zugeordnet werden.

Ein solches Vorgehen kombiniert die beiden oben vorgestellten Selektionsstrategien und bietet dadurch neue Funktionalität. Andererseits ergibt sich hieraus ein weiterer “Tradeoff”: die mehrstufige Selektion verursacht zusätzlichen Aufwand, der letztendlich die Leistungsfähigkeit im Vergleich zu einer einstufigen Selektion verringert. Die zusätzliche Funktionalität verringert die Leistungsfähigkeit.

5.3 Zusammenfassung

Es wurden zwei wichtige Variationen des Selektionsalgorithmus vorgestellt. Aufgrund der schnellen Weiterentwicklung in der Mikroelektronik ist es zwingend erforderlich, eine Lastverteilung auch an der Leistungsfähigkeit der eingesetzten Instanzen zu orientieren. Andernfalls wäre es kaum möglich, eine beschädigte Instanz alleine zu ersetzen. Man müßte alle Instanzen durch neue, untereinander baugleiche Instanzen ersetzen. Dies wäre ein erheblicher Nachteil der diskutierten Lösung.

Eine Möglichkeit, unterschiedlich leistungsfähige Hardware einzusetzen, ohne zusätzliche Synchronisation zwischen den beteiligten Instanzen einzuführen und gleichzeitig auf einen zentralen Verteiler zu verzichten, wurde in Abschnitt 5.1 vorgestellt.

Eine weitere wichtige Variation mit speziellen Anwendungsfeldern bei der Packet Screen ist die Selektion basierend auf der Kommunikationsbeziehung. Anstelle alle Dateneinheiten als voneinander unabhängig zu betrachten, werden alle Dateneinheiten, die zu einer Kommunikationsbeziehung gehören (verbindungslos oder verbindungsorientiert) von derselben Instanz selektiert. Dies ermöglicht, wie oben gezeigt wurde, u.a. die Bearbeitung von dynamisch generierter Filterregeln. Die verbindungsorientierte Lastverteilung hat jedoch auch außerhalb der Zugriffskontrolle weitere Anwendungsfelder. Dies wird in Kapitel 10 diskutiert.

Durch eine mehrstufige Selektion können die Vorteile verschiedener Selektionsvarianten kombiniert werden. Diese zusätzliche Funktionalität verringert jedoch die Leistungsfähigkeit durch den damit einhergehenden Mehraufwand.

Kapitel 6

Ausfallsicherheit der parallelen Packet Screen

Im vorigen Kapitel wurde dargestellt, daß durch den parallelen Einsatz mehrerer Packet Screens eine erhebliche Durchsatzsteigerung bei der Paketverarbeitung erreicht werden kann. Ein Hauptentwurfsziel für überlebensfähige Systeme wurde damit erreicht: die Leistungsfähigkeit ist erheblich verbessert worden.

In diesem Kapitel soll auf die Wechselwirkung zwischen der vorgenommenen Leistungsoptimierung und der Verfügbarkeit der parallelen Packet Screen eingegangen werden.

Es wird gezeigt, daß der Einsatz der Parallelverarbeitung zur Leistungssteigerung der Packet Screen auch einen entscheidenden Nachteil hat: Die Ausfallwahrscheinlichkeit des parallelen Systems ist nicht identisch mit der des sequentiellen Systems, und es gilt zudem die kontraintuitive Erkenntnis, daß die Ausfallwahrscheinlichkeit mit der Anzahl der parallelen Komponenten zunimmt. In Abschnitt 6.1 wird dargestellt, wie sich der Ausfall einer oder mehrerer Instanzen der parallelen Packet Screen qualitativ und quantitativ auswirkt und wie dieses Phänomen durch eine Zuverlässigkeitsanalyse beschrieben werden kann.

Es ist unmittelbar einsichtig, daß der Ausfall einer oder mehrerer Komponenten nicht unbedingt kritisch ist. Wenn die verbleibenden Komponenten den Ausfall erkennen und sich durch eine Rekonfiguration an die neue Situation anpassen, kann der Einsatz paralleler Komponenten die Ausfallwahrscheinlichkeit des Gesamtsystems sogar verringern. Der Einsatz paralleler Komponenten erhöht somit nicht nur den Durchsatz, sondern ist auch geeignet, die Überlebenswahrscheinlichkeit des Systems zu erhöhen. Dies ist eine sehr wünschenswerte Eigenschaft, da so ein Mechanismus (hier die Parallelverarbeitung) gleich mehrere Anforderungen umsetzen kann.

Diese wünschenswerten Systemeigenschaften (Verfügbarkeit, Zuverlässigkeit, Leistungsfähigkeit, usw.) ergeben sich jedoch nicht automatisch. Vielmehr sind einige zusätzliche Anstrengungen zu unternehmen, damit sich die in erster Linie angestrebte Durchsatzsteigerung durch Parallelverarbeitung nicht negativ auf Verfügbarkeit und Sicherheit auswirkt.

Grundsätzlich muß eine Ausfallerkennung durch die beteiligten Komponenten derart erreicht werden, daß ein Ausfall einer Komponente von den parallelen Komponenten erkannt wird, um geeignete Maßnahmen automatisch einleiten zu können. Grundlage einer algorithmischen Ausfallerkennung ist ein Fehlermodell, das die möglichen Fehler auflistet und in die zu berücksichtigenden Fehler und die zu vernachlässigenden Fehler klassifiziert. Ein Fehlermodell für den Prototypen wird daher in Abschnitt 6.2 vorgestellt und in Abschnitt 6.3 werden aus diesem Fehlermodell Anforderungen an eine fehlertolerante parallele Packet Screen abgeleitet.

In der Literatur sind einige Methoden (vgl. Abschnitt 6.4) und Protokolle (Abschnitt 6.5) bekannt, die eine Ausfallerkennung gewährleisten sollen. Leider gibt es bei diesen Protokollen entweder praxisuntaugliche Annahmen oder die Protokolle sind sehr komplex und daher zu "schwergewichtig". In Abschnitt 6.5 wird auf die bekannten Verfahren in diesem Bereich eingegangen und in Abschnitt 6.6 werden implementierbare Ausfallerkenner beschrieben.

Auf die bei überlebensfähigen Systemen ebenfalls wichtige Wechselwirkung zwischen Leistungsfähigkeit und Verfügbarkeit einerseits und der Sicherheit des Systems andererseits wird in den darauf folgenden Kapiteln eingegangen.

6.1 Zuverlässigkeit der Packet Screen und Auswirkungen durch den Ausfall von Prozessoren

Der Einsatz von mehreren parallel angeordneten Prozessoren, die sich nach einem der oben diskutierten Algorithmen die Last teilen, ist nicht unproblematisch. Im folgenden Abschnitt wird kurz die Terminologie aus dem Bereich der "Zuverlässigkeitsanalyse" eingeführt und ein Modell zur analytischen Beschreibung der Ausfallwahrscheinlichkeit der parallelen Packet Screen erstellt. Dieses Modell ist hinreichend, um die Zuverlässigkeit der parallelen Packet Screen im Vergleich mit einer sequentiellen (nur eine Instanz) Packet Screen zu beschreiben und die Abhängigkeiten der Zuverlässigkeit von der Anzahl der Instanzen aufzuzeigen.

Anschließend wird in Abschnitt 6.1.3 der Ausfall einer Komponente simuliert, um die Auswirkungen des resultierenden systematischen Paketverlustes zu veranschaulichen. Diese Simulationsergebnisse und eine darauf aufbauende Analyse motivieren die Notwendigkeit, mit geeigneten Mechanismen die Zuverlässigkeit der parallelen Packet Screen zu erhöhen.

6.1.1 Zuverlässigkeit und Kenngrößen

Mit *Zuverlässigkeit* beschreibt man die Eigenschaft eines Systems, sich innerhalb eines bestimmten Beobachtungszeitraums gemäß seiner Spezifikation zu verhalten[1]. Zuverlässigkeit

[1] Dies ist eine vereinfachte Darstellung der Definition der Zuverlässigkeit nach DIN-40041.

ist somit genau wie Sicherheit eine wünschenswerte Eigenschaft eines Systems und somit als Ziel des (Netz-)Managements (bzw. der Teildisziplin Fehlermanagement) anzusehen.

Für die Bewertung der Zuverlässigkeit benötigt man jedoch quantifizierbare Größen, die einen Vergleich mit anderen Systemen oder eine Prognose über das zu erwartende Verhalten des Systems in der Zukunft erlauben. Die wichtigste Kenngröße aus dem Bereich des Fehlermanagements ist daher die sogenannte *Verfügbarkeit*, die die Wahrscheinlichkeit angibt, ein System zu einem bestimmten Zeitpunkt in einem funktionsfähigen Zustand anzutreffen (vgl. DIN-40042, [61]).

Für die Beurteilung eines Systems ist weiterhin die Ausfallwahrscheinlichkeit bzw. die hierzu komplementäre Überlebenswahrscheinlichkeit eine geeignete Zuverlässigkeitskenngröße. Mit ihr wird die Wahrscheinlichkeit $P(t)$ angegeben, daß ein System zu einem Zeitpunkt $t \geq t_0$ ausgefallen ist bzw. noch "überlebt" hat.

Üblicherweise bezeichnet man nun bei Systemen (S), die aus mehreren Komponenten (S_i) bestehen, mit $X_i(t)$ den Zustand der Komponente S_i zum Zeitpunkt t:

$$X_i(t) = \begin{cases} 0, & \text{falls Komponente } S_i \text{ bei } t \text{ intakt ist} \\ 1, & \text{falls Komponente } S_i \text{ bei } t \text{ defekt ist} \end{cases} \tag{6.1}$$

6.1.2 Modelle zur Fehleranalyse

Bei der folgenden theoretischen Abschätzung soll gezeigt werden, daß es einen Zusammenhang zwischen der Fehlerwahrscheinlichkeit (Wahrscheinlichkeit für das Auftreten eines Fehlers) und der Anzahl der parallelen Instanzen gibt. Zu beachten ist hierbei:

- Die Ausfallwahrscheinlichkeit einer Instanz bzw. die Ausfallwahrscheinlichkeit einer sequentiellen Packet Screen ist nicht bekannt. Hierfür fehlen empirische Tests, die als Grundlage für die Bestimmung von Angaben wie etwa "Mean Time between Failure", "Mean Time to Repair", Ausfallrate usw. benutzt werden könnten.

- Es gibt auch keine empirischen Werte, um Aussagen über lastabhängiges Fehlverhalten zu machen. Intuitiv einsichtig ist, daß stark beanspruchte Komponenten eher ausfallen als weniger beanspruchte. Der Umkehrschluß, daß eine parallele Packet Screen aufgrund der geringeren Belastung der einzelnen Instanzen im Dauerbetrieb deshalb weniger ausfallanfällig ist, als eine einzelne Instanz, die alle Dateneinheiten alleine verarbeiten müßte, kann mangels empirischer Untersuchungen zu lastabhängigem Fehlverhalten nicht angenommen werden. Es können daher lediglich Aussagen über "strukturelle Ausfallwahrscheinlichkeiten" gemacht werden, da hier lediglich die Anordnung der Komponenten in die Bewertung einfließt.

Ein einfaches Modell zur Beschreibung von "strukturellem Fehlverhalten" kann durch Angabe eines Fehlerbaums erreicht werden. Der Zustand des Systems S wird dadurch in

Abhängigkeit von den Zuständen der einzelnen Komponenten (S_i, hier den einzelnen Instanzen) beschrieben. Der Zustand der sequentiellen Packet Screen ist lediglich abhängig vom Zustand der einen Instanz. Für die parallele Packet Screen gilt, daß diese genau dann fehlerfrei arbeitet, wenn alle Instanzen fehlerfrei arbeiten. In Abbildung 6.1 ist dies graphisch durch einen "ODER-Operator" ausgedrückt. Das System ist nach Gleichung 6.1 genau dann intakt ($X_S = 0$), wenn alle Instanzen intakt sind ($X_i = 0$).

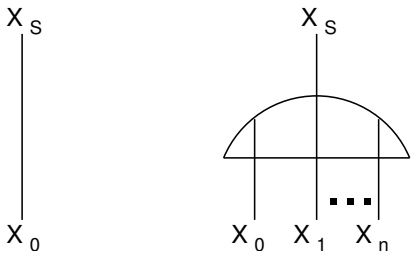


Abbildung 6.1: Fehlerbaum für sequenzielle und parallele Packet Screen

Für die parallele Packet Screen ergibt sich somit die folgende Abhängigkeit des Systemzustandes (X_S) von den einzelnen Zuständen der Komponenten (X_i):

$$X_S = X_1 \vee X_2 \vee X_3 \vee \cdots \vee X_n \tag{6.2}$$

Versagt eine Komponente, dann fällt die parallele Packet Screen aus, bzw. es ergeben sich die bereits genannten systematischen Paketverluste. Derartige Systeme, bei denen alle n Komponenten intakt sein müssen, damit das Gesamtsystem S als verfügbar bzw. intakt angesehen werden kann, werden als *"n aus n"* Systeme ($n\ of\ n$) bezeichnet.

Aufgrund der Struktur kann nun die Ausfallwahrscheinlichkeit der parallelen Packet Screen im Vergleich zu einer einzelnen Packet Screen angegeben werden. Hierzu ist es nicht erforderlich, empirische Daten für die Ausfallwahrscheinlichkeit einer einzelnen Komponente zu benutzen, da lediglich bestimmt werden soll, in welchem Verhältnis die Ausfallwahrscheinlichkeit zur Anzahl der parallelen Instanzen steht.

Für die folgende Betrachtung wird der übliche Zusammenhang zwischen den Indikatorvariablen X_i und den beiden Elementarereignissen $a_i := \{X_i = 0\}$ und $b_i := \{X_i = 1\}$ angenommen (vgl. [111]). a_i und b_i sind somit die beiden einzigen, disjunkten Elementarereignisse des Ereignisraums $\Omega_i := \{S_i \text{ ist intakt}, S_i \text{ ist defekt}\}$. Die Ereignisse können genau wie die Indikatorvariablen auch in Abhängigkeit vom Zeitpunkt betrachtet werden. Das Elementarereignis $a_i(t)$ bezeichnet dann das Ereignis, daß die Systemkomponente S_i zum Zeitpunkt t intakt ist ($X_i(t) = 0$).

Die Wahrscheinlichkeit, daß das System "parallele Packet Screen" intakt ist, wird als Überlebenswahrscheinlichkeit bezeichnet und ist in diesem Fall (vgl. [203]):

$$A_{n\ of\ n}(n,t) = P\left(\bigcap_{i=1}^{n} a_i(t)\right) \tag{6.3}$$

Das System ist verfügbar, wenn alle Subsysteme (S_i, die parallelen Instanzen) zum Zeitpunkt t noch intakt sind ($X_i(t) = 0$). Unter der Voraussetzung, daß alle Subsysteme unabhängig voneinander sind und Fehler in einem Subsystem nicht weitere Fehler in einem anderen Subsystem nach sich ziehen, ist die Überlebenswahrscheinlichkeit und somit die Verfügbarkeit des Systems durch das Produkt der Überlebenswahrscheinlichkeiten der einzelnen Subsysteme gegeben:

$$A_{n\ of\ n}(n,t) = \prod_{i=1}^{n} P(a_i(t)) \tag{6.4}$$

Da keine empirischen Werte für die Überlebenswahrscheinlichkeit vorliegen, wird als Beispiel angenommen, daß nach der Inbetriebnahme einer einzelnen Komponente (S_i) zum Zeitpunkt t_0 diese zu einem späteren Zeitpunkt $t_1 > t_0$ mit einer Wahrscheinlichkeit von 95% noch intakt ist:

$$P(a_i(t_1)) = 0,95 \tag{6.5}$$

Dann ergibt sich aus Formel 6.4 die Überlebenswahrscheinlichkeit des Systems in Abhängigkeit von der Anzahl der parallel geschalteten Komponenten zum Zeitpunkt t_1:

$$\begin{aligned} A_{n\ of\ n}(n,t_1) &= \prod_{i=1}^{n} P(a_i(t_1)) \\ A_{n\ of\ n}(n,t_1) &= (P(a_i(t_1)))^n \\ A_{n\ of\ n}(n,t_1) &= 0,95^n \end{aligned} \tag{6.6}$$

Beispielsweise sinkt die Überlebenswahrscheinlichkeit beim Einsatz einer einzelnen Komponente von 95% auf 81,5% beim Einsatz von vier parallelen Komponenten.

Die Ausfallwahrscheinlichkeit (als Komplement der Überlebenswahrscheinlichkeit) steigt durch die zusätzlichen parallelen Instanzen an. Dieses Ergebnis ist lediglich von der Struktur der parallelen Packet Screen abhängig und nicht von der Überlebenswahrscheinlichkeit einer einzelnen Instanz.

6.1.3 Ausfall einer Instanz der parallelen Packet Screen

Der Ausfall eines Prozessors kann zu beträchtlichen Leistungseinbußen führen. In Abbildung 6.2 ist dargestellt, wie sich der Ausfall eines Prozessors auswirkt. Die Empfangsleistung für vier parallele Instanzen ist mit *4 von 4* markiert. Fällt eine Instanz aus, ist nur noch der mit *3 von 4* markierte Durchsatz erreichbar. Bei einer unidirektionalen UDP-basierten Kommunikation erkennt man, daß erwartungsgemäß etwa 1/4 aller Dateneinheiten verloren gehen. Es sind aber noch zwei weitere Phänomene erkennbar: Der Durchsatz der drei verbleibenden Instanzen erreicht nie den Sendedurchsatz, auch nicht für große Dateneinheiten. Dies ist eine Auswirkung des Selektionsalgorithmus: da die drei

verbleibenden Instanzen immer noch davon ausgehen, daß vier Instanzen aktiv sind, werden von ihnen somit immer 1/4 der Dateneinheiten mißachtet. Darüber hinaus ist der Durchsatz der drei verbleibenden Instanzen auch für alle Datagrammlängen schlechter als der Durchsatz einer Packet Screen, die aus drei richtig konfigurierten Instanzen besteht (*rcv, 3 von 3*).

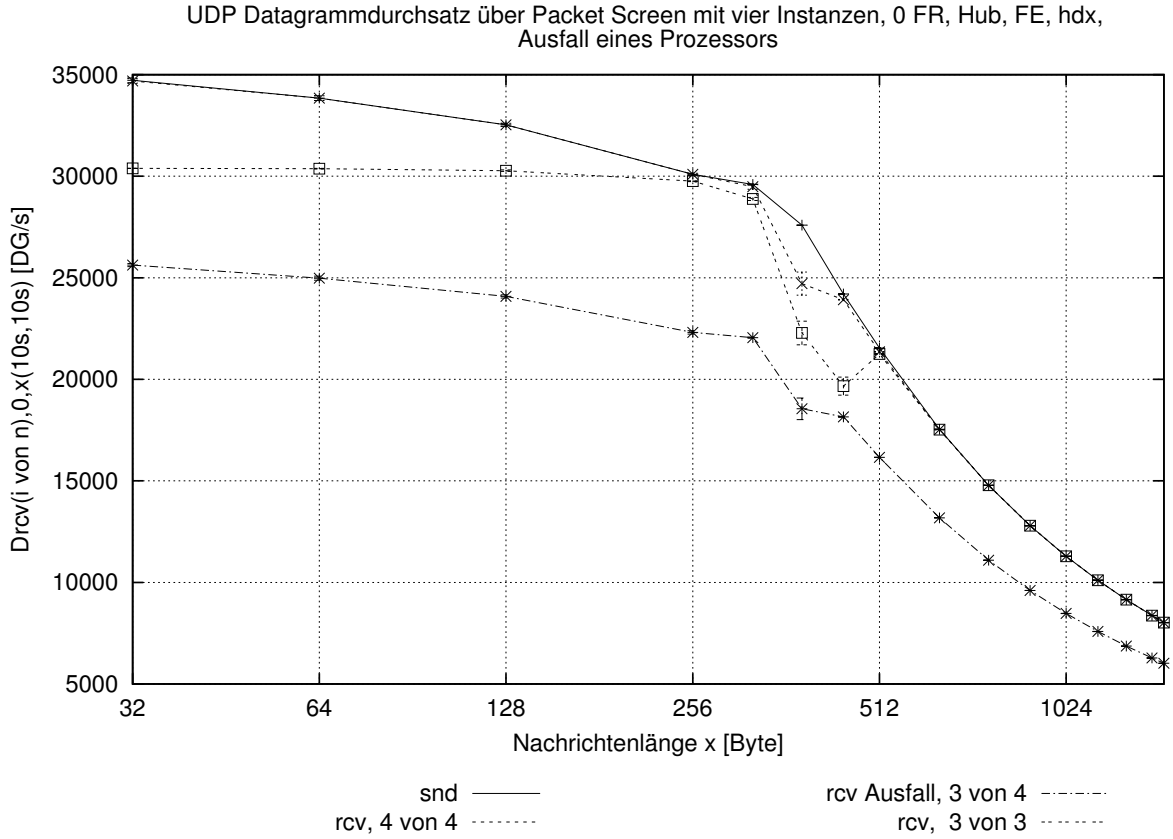


Abbildung 6.2: Durchsatzeinbuße bei Ausfall einer von vier Instanzen

Im folgenden wird gezeigt, daß dieses Phänomen unabhängig von der Anzahl der Instanzen ist und somit der Ausfall einer Instanz immer zu unverhältnismäßig vielen, systematischen Paketverlusten führt.

Die Anzahl der Pakete (P_{lost}), die durch den Ausfall von Prozessoren verloren gehen, ist bei einer angenommenen gleichmäßigen Verteilung der Dateneinheiten auf alle Prozessoren somit direkt proportional zur Anzahl der ausgefallenen Prozessoren ($Inst_{defect}$) bzw. umgekehrt proportional zur Gesamtzahl der eingesetzten Prozessoren (n):

$$P_{lost} = Pmax_n(n) * \frac{Inst_{defect}}{n} \tag{6.7}$$

Bei einer großen Anzahl paralleler Instanzen mag der Ausfall einer Komponente im Spezialfall der parallelen Packet Screen nicht besonders tragisch sein, da Paketverluste in IP-Netzen üblich sind und Protokolle, wie beispielsweise TCP, mit entsprechenden Mechanismen eine erneute Übertragung veranlassen. Allerdings werden einzelne Dateneinheiten bei einem Komponentenausfall durch den zugrundeliegenden Selektionsalgorithmus mit einer gewissen (algorithmischen) Systematik verloren.

Diese *systematischen Paketverluste* wirken sich besonders dramatisch aus, wenn es zusätzliche Abhängigkeiten zwischen den Dateneinheiten gibt. Ein Beispiel hierfür sind TCP-Verbindungen, bei denen die übertragenen Dateneinheiten bestätigt werden. Durch den Ausfall eines Prozessors tritt nun der Effekt auf, daß auch Bestätigungen verloren werden. Dies kann dazu führen, daß einer oder beide Kommunikationspartner keine weitere Daten mehr senden können, da durch die ausbleibenden Bestätigungen die Fenstergröße ("sliding window"-Mechanismus bei TCP) voll ausgeschöpft ist. Der Durchsatz auf diesen Verbindungen bricht dann drastisch ein. Bei einer Messung konnte empirisch ermittelt werden, daß bereits der für TCP-Verbindungen erforderliche drei-Wege-Verbindungsaufbau scheitern kann. In diesem Fall entspricht der Ausfall einer Instanz aus Sicht des Benutzers[2] einem Totalausfall der Packet Screen, da keine Verbindung aufgebaut werden kann und somit keine Datenübertragung möglich ist.

Offensichtlich ist daher bereits aus rein funktionalen Gründen eine Ausfallerkennung mit einer anschließend durchzuführenden Rekonfiguration der verbleibenden Instanzen erforderlich. Darüber hinaus wird im folgenden gezeigt, daß auch eine Leistungssteigerung durch die Rekonfiguration nach Erkennung eines Ausfalls erreicht wird.

In Formel 4.14 wurde bereits der erreichbare Paketdurchsatz in Abhängigkeit von der Anzahl der parallelen Instanzen angegeben. Durch den Ausfall einer Instanz ($Inst_{defect} = 1$) sinkt dieser Durchsatz auf P_{crash}:

$$\begin{aligned} P_{crash}(n) &= Pmax_n(n) - P_{lost} \\ &= Pmax_n(n) - \frac{Pmax_n(n)}{n} \\ &= Pmax_n(n)(1 - \frac{1}{n}) \end{aligned} \tag{6.8}$$

Wird nun der Ausfall der Komponente erkannt und die $(n-1)$ verbleibenden Instanzen rekonfigurieren ihren Selektionsalgorithmus entsprechend neu, dann steigt auch wieder der Paketdurchsatz an, denn es gilt, unter Nutzung des analytischen Modells aus Abschnitt 4.6, unabhängig von der Anzahl der eingesetzten Prozessoren ($n > 1$):

$$Pmax_n(n-1) > Pcrash(n) \tag{6.9}$$

[2] Der Initiator der TCP-Verbindung.

Beweis: Zu zeigen ist lediglich, daß:

$$\frac{(n-1)}{(n-1)B_{sel}+B_{fil}} > \frac{n}{nB_{sel}+B_{fil}} - \frac{1}{nB_{sel}+B_{fil}} \tag{6.10}$$

und dies ist offensichtlich erfüllt, da:

$$\frac{n-1}{(n-1)B_{sel}+B_{fil}} > \frac{n-1}{nB_{sel}+B_{fil}} \quad , n \geq 2 \tag{6.11}$$

Mit anderen Worten, die Rekonfiguration ist nicht nur funktional erforderlich, sondern ermöglicht auch eine Leistungssteigerung gegenüber einer verteilten Architektur, die nicht auf einen Ausfall reagiert.

6.2 Fehlermodell und Fehlersemantik

Im Bereich der "Fehlertoleranten Systeme" ist es gängige Praxis, vor dem Systemdesign eines (fehlertoleranten) Systems, ein Fehlermodell aufzustellen. Das Fehlermodell legt fest, welche Art von Fehlern das System tolerieren soll. Hierbei werden auf Grundlage von empirischen Werten oder Modellanalysen wahrscheinliche Fehlerquellen identifiziert. Andere Fehler werden nicht ausgeschlossen aber meist als zu unwahrscheinlich abgetan; diese Fehler sind somit "vernachlässigbar" (negligible).

Aufgrund des Fehlermodells ist es dann möglich, neben dem Verhalten des Systems im fehlerfreien Betrieb, auch das Verhalten des Systems bei Auftreten der wahrscheinlichen Fehler zu spezifizieren. Man erstellt als Teil der Systemspezifikation die sog. Fehlersemantik.

Die Fehlersemantik legt somit das Verhalten des Systems nach dem Auftreten eines Fehlers fest. Da die Fehler und das Verhalten des Systems bei Auftreten der Fehler durch die Fehlersemantik mit in die Systemspezifikation einbezogen werden, arbeiten diese Systeme per Spezifikation trotz Ausfall einzelner Komponenten weiterhin *korrekt*. Ein System, das beispielsweise eigenes Fehlverhalten erkennt und sich daraufhin abschaltet, arbeitet in diesem Sinne korrekt.

Ziel muß es aber weiterhin sein, nicht nur korrekte Systeme im obigen Sinne zu spezifizieren, sondern letztendlich sollen diese Systeme trotz auftretender Fehler die Aufträge der Benutzer (weiter)verarbeiten. Fehlertolerante Systeme werden daher möglichst so entworfen, daß sie trotz auftretender Fehler ihre Dienste erbringen. Das fehlertolerante System *maskiert* sozusagen den Fehler [58].

6.2.1 Fehlermodell für die parallele Packet Screen

In diesem Abschnitt sollen potentielle Fehlerquellen identifiziert werden. Daraus läßt sich, wie oben dargestellt, ein Fehlermodell ableiten und das Verhalten der parallelen Packet

Screen bei Auftreten eines Fehlers spezifizieren. Eine allgemeine Darstellung des Fehlerbegriffs[3] und der gängigen Techniken im Bereich der fehlertoleranten Systeme wird beispielsweise in [58, 147] gegeben.

Bei der Fehlerbehandlung ist es das Ziel, solche Fehler zu maskieren, die durch die Parallelisierung erst möglich werden. Es werden also keine Fehler behandelt, die beispielsweise bei einer herkömmlichen (sequentiellen) Packet Screen auftreten können. Insbesondere wird hier nicht auf die Korrektheit der Packet Screen Software oder des zugrundeliegenden Betriebssystems eingegangen. Vielmehr soll das erhöhte Ausfallrisiko durch die Parallelisierung beschrieben und in einem adäquaten Modell aufgezeigt werden.

6.2.1.1 Potentielle Hardware- und Softwarefehler

Das Schema der parallelen Packet Screen aus Abbildung 4.8 ist hier um mögliche Hardwarefehler ergänzt, die im folgenden weiter diskutiert werden.

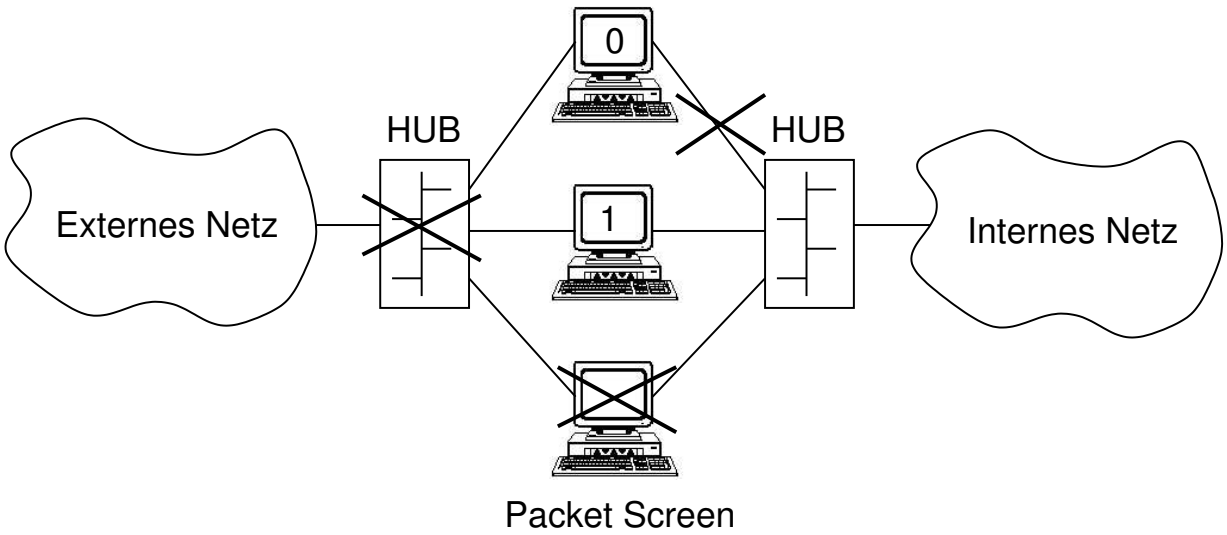


Abbildung 6.3: Potentielle Hardwarefehler bei der parallelen Packet Screen

Defekte Netzkomponente: Der Ausfall eines Hubs oder eines Switches würde die gesamte Kommunikation des internen Netzes mit dem externen Netz unterbinden. Dieser Fehler sollte erkannt werden, beispielsweise um durch entsprechende Meldungen den Operator zu informieren oder auch, um die parallele Packet Screen in einen Wartezustand ("stand by") zu schalten, bis der Fehler behoben ist.

[3] So wie es im deutschen nur das Wort "Sicherheit" für die englischen Begriffe "safety" und "security" gibt, ist das deutsche Wort "Fehler" im Bereich der fehlertoleranten Systeme semantisch deutlich überstrapaziert: Es gilt "error" von "fault" und "failure" zu unterscheiden [147].

Der Ausfall der Netzkomponenten ist jedoch nicht sehr wahrscheinlich, so daß nicht über eine redundante Verkabelung der Packet Screens mit beiden Netzen nachgedacht werden muß. Dies würde i.d.R. auch spezielle Komponenten [218] erfordern. Außerdem erhöht sich die Ausfallwahrscheinlichkeit der aktiven Netzkomponenten (Switches, Hubs) durch den Einsatz der parallelen Packet Screen (vermutlich) nicht.

Kabelbruch: Ein Anschlußkabel, das eine Schnittstelle der Netzkomponente mit einer Schnittstelle des Rechners verbindet, könnte ausfallen. Beim Einsatz von Hubs ist bei einem Kabelbruch die Verbindung in jedem Fall unterbrochen. Beim Einsatz eines Switches könnte im "full-duplex"-Betrieb ein Kabelbruch in einer Ader des Kabels zu einem einseitigen Ausfall der Verbindung führen. Es wäre denkbar, daß nur noch unidirektional vom Switch zum Rechner oder vom Rechner zum Switch kommuniziert werden kann.

In jedem Fall muß dieser Fehler erkannt werden, da er zu erhöhten Paketverlusten führt. Immer dann, wenn die durch Kabelbruch nicht mehr erreichbare Workstation ein Paket selektieren würde, geht diese Dateneinheit verloren, da alle anderen Rechner diese Dateneinheit nicht bearbeiten (vgl. Abschnitt 4.3.5). Es treten somit vermehrt "omission failures" auf.

Der Kabelbruch steht stellvertretend für eine ganze Klasse von Fehlern. In der Wirkung ist der Kabelbruch identisch mit dem Ausfall einer Schnittstelle auf der Netzkomponente (Hub, Switch) oder der Rechnerschnittstelle. In jedem Fall gehen Dateneinheiten verloren. Der Rechner ist grundsätzlich in der Lage, diesen Fehler zu erkennen, da typischerweise die Gerätetreiber moderner Netzschnittstellen selbst Auskunft über den Zustand der Verbindung zum Rechnernetz geben können. Eine Instanz ist daher in der Lage, diesen Fehler zu melden und/oder als "fail-stop failure" (s.u.) zu maskieren.

Defekter Rechner: Wenn eine der Workstations ausfällt, tritt ein sog. "fail-stop failure" auf. Die Workstation kann den angeforderten Dienst nicht mehr erbringen und wechselt in einen Zustand, der prinzipiell von den anderen Rechnern der parallelen Packet Screen erkannt werden kann, da Nachrichten nicht mehr beantwortet werden. So lange mindestens ein Rechner/Prozessor der parallelen Packet Screen noch verfügbar ist, wird dieser "fail-stop failure" als vermehrt auftretender "omission failure" maskiert. Es tritt ein erhöhter Paketverlust auf, die parallele Packet Screen filtert jedoch weiterhin Dateneinheiten (vgl. Formel 6.7).

Der qualitative Unterschied zwischen Rechnerausfall und Kabelbruch besteht darin, daß der Rechnerausfall durch die anderen Rechner der parallelen Packet Screen erkannt werden muß, während der Kabelbruch von dem jeweils betroffenen Rechner selbst erkannt werden kann.[4]

[4] Zumindest kann ein Rechner theoretisch den Kabelbruch erkennen. Praktisch ist dafür jedoch erst die Funktionalität der Gerätetreiber zu erweitern (vgl. Abschnitt C.3)

Softwarefehler: Auf den einzelnen Rechnern wird die Filterfunktionalität durch Prozesse, bzw. durch in Ausführung befindliche Programme erbracht. Diese Programme können beispielsweise aufgrund von Programmierfehler zufällig oder auch deterministisch ausfallen.

Auch in diesem Fall treten letztendlich "omission failures" auf, so lange noch wenigstens ein paralleler Prozeß weiterhin die Filterregeln abarbeitet. Ohne weitere Maßnahmen ist dieser Fehler in seinen Auswirkungen identisch mit dem Rechnerdefekt und kann durch die parallelen Instanzen erkannt werden.

Wenn parallel zum Server mit den Filterfunktionen auf dem selben Prozessor ein Monitor zur Überwachung der Serverfunktionalität installiert wird, dann ergeben sich jedoch weitere Möglichkeiten: Im einfachsten Fall kann der Monitor nach dem Erkennen eines Programmabbruchs den Fehler an die parallelen Instanzen melden. In diesem Fall entspricht der Programmabbruch in seinen Auswirkungen dem Kabelbruch. Alternativ kann der Monitor auch einen Neustart des Programms veranlassen, so daß der "crash"-Fehler als "omission"-Fehler maskiert wird. Erfolgt diese Reaktion schnell genug, kann auf eine Rekonfiguration verzichtet werden.

6.2.1.2 Fehlersemantik der parallelen Packet Screen

Aus der obigen Diskussion wird deutlich, daß bei n parallelen Instanzen alle auftretenden Fehler so lange als "omission failure" maskiert werden können, bis auch die letzte Instanz ausfällt. Bezüglich der oben genannten Fehler ist die parallele Packet Screen somit bereits ohne jede weitere Ausfallsicherung $n-1$ fehlertolerant [205] bezüglich Komponentenausfällen (Kabel, Rechner oder Prozessor) und genügt während dieser Zeit einer *Verlust-Fehler-Semantik* ("omission-failure semantic").

Fallen alle n parallelen Instanzen aus, dann hat die parallele Packet Screen einen totalen *Ausfallfehler* und somit *"crash-failure semantic"*.[5]

6.2.2 Bewertung der Fehlersemantik

Die Packet Screen ist eine Sicherheitskomponente. Die oben als nicht vernachlässigbar angesehenen Fehler führen entweder zu "omission failures" oder zu "crash failures". Es kommt insbesondere nicht zu sog. "response failures", also falschen Berechnungen auf der Packet Screen. Dies ist strenggenommen eine sichere Fehlersituation. Egal ob es zu Kabelbrüchen oder Serverausfällen kommt: die Sicherheit ist durch den Ausfall einer der Komponenten nicht gefährdet ("fail safe"), es werden keine Dateneinheiten weitergeleitet, die im fehlerfreien Zustand gefiltert worden wären (keine Möglichkeit für "response/value failure").

[5] In [205] wird ein wiederholt auftretender "omission failure" als "crash failure" klassifiziert. Bei Ausfall aller parallelen Instanzen gehen alle Dateneinheiten verloren, ein wiederholt auftretender "omission failure"!

Allerdings kann der Ausfall einer Komponente zu einem ungenügenden Audit führen. Gerade die ausgefallene Instanz könnte möglicherweise sicherheitsrelevante Informationen in den Paketinhalten erkennen, wenn sie noch aktiv wäre. Diese Beeinträchtigung ("degradation") des Audits muß allerdings im direkten Vergleich mit einer sequentiellen Instanz gesehen werden. Fällt eine (sequentielle) Packet Screen aus, ist sie auch nicht mehr in der Lage, ein Audit durchzuführen. Bei der parallelen Packet Screen führt ein Ausfall einer Instanz somit möglicherweise zu einer Einschränkung des Audits im Vergleich zu einem fehlerlosen Betrieb; das Audit der parallelen PS ist im Vergleich mit einer sequentiellen Packet Screen aber auch beim Ausfall einer oder mehrerer Instanzen immer noch überlebensfähig und somit auch besser.

Weiterhin fällt auf, daß durch den Ausfall einer Komponente und den daraus folgenden "omission failures" kein kritischer Zustand erreicht wird. Eine IP-basierte Packet Screen, die aufgrund zu hoher Last Dateneinheiten verliert, zeigt ein ähnliches Verhalten. Außerdem wird von IP ("best effort"-Datagrammdienst) die vollständige und richtige Übertragung weder zwischen Endsystemen noch zwischen Vermittlungsrechnern garantiert. Das oben dargestellte Fehlerverhalten der parallelen Packet Screen wäre somit, bis zu einem gewissen Grad, sogar tolerierbar bzw. transparent für den Benutzer.

Andererseits ist es natürlich das Ziel, durch Parallelverarbeitung eine Beschleunigung herbeizuführen. Unter *systematischen* Paketverlusten durch Ausfall einer Komponente leidet der Durchsatz, insbesondere der Durchsatz von verbindungsorientierten Protokollen, bei denen sowohl Nutzdateneinheiten als auch Bestätigungen verloren werden. Es ist also unmittelbar einsichtig, daß der Ausfall einer Komponente erkannt und der Verteilungsschlüssel (vgl. Abschnitt 42) neu berechnet werden muß, um diese *systematischen* Paketverluste zu unterbinden.

Ein weiteres Problem entsteht bei einem Ausfall einer Komponente, wenn der Selektionsalgorithmus verbindungsorientiert arbeitet und beispielsweise alle Dateneinheiten mit gleichen Sender- und Empfängeradressen von derselben Instanz selektiert werden. Fällt nun eine dieser Instanzen aus, dann werden einzelne Verbindungen überhaupt nicht mehr gefiltert ("crash failure" aus Benutzersicht). Dieses Verhalten ist selbstverständlich nur kurzfristig tolerierbar. Auch hier muß eine Anpassung des Selektionsalgorithmus eine Übernahme der Verbindungen durch andere Instanzen ermöglichen.

6.3 Anforderungen an eine fehlertolerante Packet Screen

Aus der Diskussion im letzten Abschnitt lassen sich unmittelbar Anforderungen an eine fehlertolerante parallele Packet Screen ableiten. Darüber hinaus ergeben sich weitere Anforderungen, die einen einfachen Umgang mit der parallelen Packet Screen ermöglichen (s.u.).

Allgemeine Anforderungen werden im folgenden nicht formuliert. Es geht hier um die zusätzlichen Anforderungen, die ein fehlertolerantes Verhalten ermöglichen.

6.3.1 Anforderungen an die Fehlerbehandlung

Der Ausfall einer Netzkomponente (Hub oder Switch) sollte dazu führen, daß die parallele Packet Screen in einen Wartezustand übergeht. Dieser Wartezustand wird bis zur Reparatur der Netzkomponente beibehalten, danach soll die parallele Packet Screen selbständig wieder anlaufen.

Ein als Kabelbruch klassifizierbarer Fehler[6] muß erkannt werden. Die vom Kabelbruch betroffene Instanz stellt im allgemeinen Fall den Betrieb ein und wartet auf Reparatur. Ist wie im Fall der Packet Screen jede Instanz mit mindestens zwei Schnittstellen ausgerüstet, kann zusätzlich über die redundante Anbindung eine Warnmeldung an die anderen Instanzen ausgegeben werden. Die parallelen Instanzen müssen sich bei Empfang der Fehlerbenachrichtigung erneut konfigurieren und den Selektionsalgorithmus an die neue Anzahl verfügbarer Instanzen anpassen. Stellt die betroffene Instanz eine Reparatur fest (z.B. das Kabel wurde ausgewechselt), sollte sie sich wieder an der Verarbeitung beteiligen können.

Ein Fehler durch Ausfall einer Instanz muß durch ein geeignetes Protokoll erkannt werden. Dieses Protokoll muß die verbleibenden Instanzen darüber hinaus in die Lage versetzen, sich an die neue Situation anzupassen. Da die ausgefallene Instanz selbst nicht in der Lage ist, eine entsprechende Fehlermeldung abzusetzen, sollte mindestens eine verfügbare Instanz diesen Ausfall mit entsprechenden Log-Meldungen einem Operator melden.

6.3.2 Weitere Anforderungen

Auf die Bedienung und Konfiguration der parallelen Packet Screen wurde bisher lediglich indirekt bei der Beschreibung der Algorithmen zur Selektion der Pakete eingegangen (vgl. Abschnitt 4.3.5).

Die Instanzen müssen vor der Inbetriebnahme mit eindeutigen IDs konfiguriert werden und laufend über die aktuelle Gesamtzahl der aktiven Instanzen informiert werden. Will man nach einem Ausfall eine reparierte Instanz in Betrieb nehmen, müßten alle anderen Instanzen manuell vom Operator konfiguriert werden. Dieses Vorgehen ist nicht nur fehleranfällig, sondern dauert auch vergleichsweise lange. Während dieser Zeit kann es durch die unterschiedlichen Sichten der Instanzen sowohl zu Paketverdoppelungen als auch zu Paketverlusten kommen. Dieses Verhalten ist zwar nicht kritisch aber zumindest unerwünscht. Eine Rekonfiguration muß daher schnell erfolgen, um Paketverdoppelungen oder Paketverluste während dieser Rekonfigurationszeit zu minimieren.

Hieraus ergibt sich unmittelbar die Forderung, daß eine reparierte Komponente/Instanz automatisch in die Gruppe der parallelen Instanzen aufgenommen wird und sich die Instanzen selbständig rekonfigurieren. Damit ist auch der Fall abgedeckt, daß eine weitere

[6] Wie bereits im letzten Abschnitt angedeutet wurde, verbergen sich hinter “Kabelbruch” auch andere Fehler. Neben dem Ausfall einer Schnittstelle und einem Kabelbruch im klassischen (engeren) Sinn, kann es sich hierbei auch um ein gelöstes Kabel handeln: Operator bleibt mit dem Fuß am Kabel hängen und bemerkt nicht, daß sich der Stecker aus der Schnittstelle löst.

Instanz in die Gruppe der parallelen Instanzen aufgenommen werden soll, um die parallele Packet Screen zu erweitern. Verallgemeinert man diesen Fall, dann läßt sich auch ein automatischer "Bootstrap" durchführen: Die parallele Packet Screen, bestehend aus n Instanzen, konfiguriert sich nach der möglicherweise gleichzeitigen Inbetriebnahme der Instanzen/Server selbständig. Dieser Punkt ist in Hinblick auf eine Beherrschbarkeit des überlebensfähigen Systems wichtig. Die automatische Konfiguration fördert die Akzeptanz durch den Administrator, da kein (Management-)Mehraufwand für die neue Funktionalität erforderlich ist. Darüber hinaus ist die automatische Konfiguration besser geeignet als die manuelle Variante, um eine fehlerfreie Initialisierung aller Instanzen bei der Inbetriebnahme zu erreichen.

Zusammenfassend ergeben sich daher die folgenden Anforderungen:

A1 Ausfälle von Netzkomponenten müssen erkannt werden. Die parallele Packet Screen geht in einen Wartezustand über.

A2 Kabelbrüche müssen erkannt werden, so daß sich die verbleibenden Instanzen rekonfigurieren können. Die betroffene Instanz wartet auf die Reparatur.

A3 Instanz- bzw. Rechnerausfälle müssen erkannt werden. Die parallelen Instanzen müssen sich gegenseitig überwachen, um sich in diesem Fall automatisch zu rekonfigurieren.

A4 Automatischer "Bootstrap" und automatische Einbindung neuer und reparierter Instanzen muß erfolgen. Die parallele Packet Screen muß auch den Fall erkennen, daß neue Instanzen in die Gruppe aufgenommen werden sollen, um anschließend eine Rekonfiguration vorzunehmen.

Für die automatische Erkennung eines Ausfalls und die anschließende Rekonfiguration muß ein Protokoll entwickelt werden. Offensichtlich muß es außerdem einen Dienst geben, der die Aufnahme einer neuen Instanz in die Gruppe der parallelen Instanzen bereitstellt. Derartige Protokolle sind im Bereich der fehlertoleranten Systeme als "group membership protocols" (GMPs) bekannt und zumindest theoretisch gut verstanden [55].

In dem folgenden Abschnitt wird untersucht, welche Verfahren es im Bereich der GMPs gibt, und wie sie für den Aufbau einer fehlertoleranten parallelen Packet Screen eingesetzt werden können.

6.4 Abhängigkeiten der Mechanismen im Bereich der fehlertoleranten Systeme

Bevor im nächsten Abschnitt direkt auf die Verwaltung der Gruppenzugehörigkeit eingegangen wird, soll hier kurz aufgezeigt werden, in welchem Verhältnis dieser Mechanismus zur anderen Mechanismen bzw. Diensten im Bereich der fehlertoleranten Systeme steht.

Die Verwaltung der Gruppenzugehörigkeit ist einer von mehreren allgemeinen Mechanismen, die mit anderen Mechanismen in einer sog. "depends on"-Relation stehen [58]. Man sagt, ein Mechanismus oder allgemein ein Dienst u, *ist abhängig von* (depends on) einem anderen Dienst v, wenn u zur Erfüllung seines Auftrages auf die Hilfe des Dienstes v angewiesen ist. Man bezeichnet u auch als *höherwertigen* Dienst und bildet die "depends on"-Relation daher graphisch meist von "oben nach unten" ab (vgl. Abbildung 6.4). Die

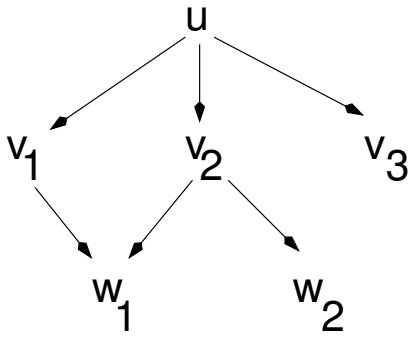


Abbildung 6.4: u hängt ab von $v_1, \ldots, v_3$. v_1 und v_2 sind selbst abhängig von den Diensten w_1 bzw. w_2

ISO-Schichtenstruktur für Rechnernetze (vgl. [134]) ist ein bekanntes Beispiel für derartige Dienstnehmer/Dienstleister- Abhängigkeiten.

Im Bereich der fehlertoleranten Systeme haben sich mehrere dieser "depends on"-Relationen zwischen allgemeinen Mechanismen herausgestellt. In Anlehnung an [170, 54, 130] zeigt Abbildung 6.5 die gängigen Abhängigkeiten für verteilte Anwendungen nach dem Automatenmodell. Dieses Modell wird unten kurz vorgestellt, da die parallele Packet Screen zusammen mit den noch zu entwickelnden Mechanismen für die Fehlertoleranz unter dieses Paradigma eingeordnet werden muß.

In der Abbildung sind die Dienste für die Fehlertoleranz fett dargestellt. Die Inschriften an den "depends on"-Pfeilen benennen den Mechanismus des höherwertigen Dienstes, für den eine Funktion des darunterliegenden Dienstes benötigt wird.

Neben dem Automatenmodell gibt es noch das "Objekt/Aktion"-Modell, das "Primary/-Backup"-Modell und das "Konversationsmodell", auf die hier nicht näher eingegangen wird. Diese alternativen Programmierparadigmen für fehlertolerante Systeme zeichnen sich durch andere "depends on"-Relationen aus, die nicht in Abbildung 6.5 eingezeichnet sind.

Das Automatenmodell

Beim Automatenmodell [205] betrachtet man in der Regel gleichartige Prozesse, die auf mehrere verteilte Komponenten repliziert werden. Sie kommunizieren untereinander mittels Nachrichten und ihr interner Zustand läßt sich vollständig durch die Sequenz der einge-

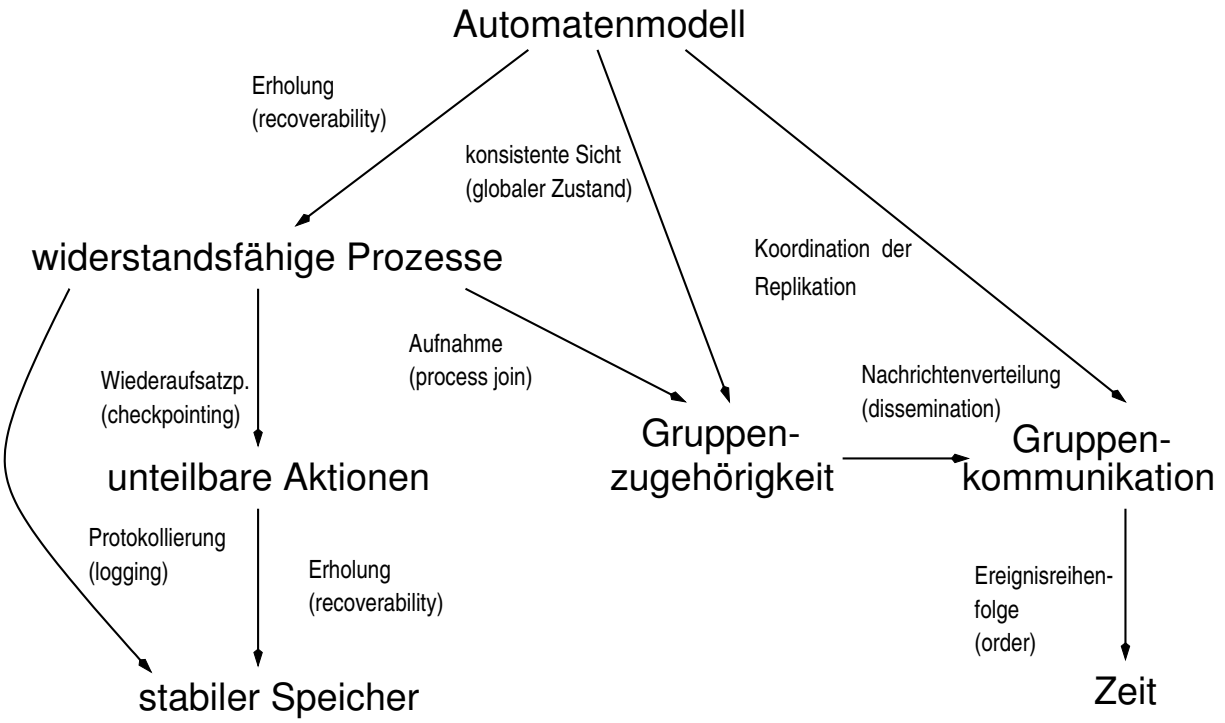


Abbildung 6.5: Typische Abhängigkeiten beim Automatenmodell (vgl. [170, 54, 130])

gangenen Nachrichten beschreiben. Fehlertoleranz wird dadurch erreicht, daß die Replica untereinander als aktive Redundanz (für den zu erbringenden Dienst) fungieren.

Für die Koordination der Replikation wird daher meist Gruppenkommunikation benötigt, die die Ausbreitung von Nachrichten an alle replizierten Prozesse unterstützt. Um Nachrichten von verschiedenen Prozessen in eine partielle oder gar totale Ordnung zu bringen, benötigen die Gruppenkommunikationsmechanismen in der Regel einen Zeit(stempel)dienst.

Die gemeinsame Sicht auf einen globalen Zustand erreichen die Prozesse durch das Gruppenmanagement, das sich wiederum der Gruppenkommunikation bedient. Ist nach einem Ausfall eines Prozesses eine Reparatur mit Wiederaufnahme in die Gruppe geplant, dann müssen die Prozesse selbst auch widerstandsfähig sein. Beispielsweise zur selbständigen Wiederaufnahme einer Teilberechnung, die dem Prozeß innerhalb der Gruppe zugeteilt wurde.

Zu Abbildung 6.5 ist noch anzumerken, daß dort nicht alle grundsätzlich denkbaren Abhängigkeiten aufgenommen worden sind. Die dort dargestellten Abhängigkeiten und Fehlertoleranzmechanismen werden in der Diskussion in den folgenden Kapiteln wieder auftauchen. Ihre Funktionen werden jedoch in Hinblick auf die Überlebensfähigkeit, also in Hinblick auf Leistungsfähigkeit und Sicherheit, neu bewertet. Die Fehlertoleranz ist eben nur eines von drei Optimierungszielen. Zentral für die folgende Ausführungen ist die Verwaltung

der Gruppenzugehörigkeit. Auf die Verwaltung der Gruppenzugehörigkeit wird daher im Detail eingegangen.

6.5 Verwaltung von Gruppenzugehörigkeit

In der Literatur zu *fehlertoleranten Systemen* ist das Herstellen und Verwalten von Prozeß- bzw. Prozessoren-Gruppen ein altbekanntes, zentrales Problem. Ansätze zur Lösung dieses Problems werden im folgenden mit dem Namen "Gruppenmanagement" bezeichnet. Die dazu verwendeten Protokolle werden als "Gruppenmanagement-Protokolle" (GMPs) bezeichnet. Es gibt zwei Klassen von Algorithmen, diejenigen für asynchrone verteilte Systeme (vgl. beispielsweise [5, 28, 44, 132, 168, 94]) und diejenigen für synchrone verteilte Systeme (vgl. beispielsweise [16, 17, 56, 57, 50, 230]).

Bei den Algorithmen für asynchrone verteilte Systeme ist die Grundannahme, daß es keine zeitlichen Beschränkungen weder für Nachrichtenlaufzeiten noch für die relativen Prozessorgeschwindigkeiten gibt. Dieses Modell entspricht im wesentlichen lose gekoppelten Rechnern, die mit einem LAN oder WAN untereinander verbunden sind, sog. *"Workstation-Cluster"*. Das zentrale Problem bei der Fehlererkennung in asynchronen Netzen ist, zwischen einem tatsächlich ausgefallenen Prozessor und einem sehr langsamen Prozessor zu unterscheiden. Nebenläufigkeit ist die Folge unvorhersehbarer Last in einem System (vgl. [44]). Die Einigung auf die Gruppenzugehörigkeit ist bereits bei nur einem ausgefallenen Prozessor in asynchronen Umgebungen nicht mehr deterministisch zu lösen (vgl. [88]). Eine Möglichkeit, dieses Problem zu umgehen, besteht im Einsatz von *unzuverlässigen Fehlererkennern*, die sich möglicherweise irren, also ein Gruppenmitglied fälschlicherweise als ausgefallen/aktiv erkennen (beispielsweise [3, 5, 44]).

Dagegen wird bei Algorithmen für synchrone verteilte Systeme davon ausgegangen, daß die maximalen Nachrichtenlaufzeiten bekannt sind und es obere Schranken für die relativen Prozessorgeschwindigkeiten gibt. Dieses Modell paßt sehr gut zu Multiprozessorsystemen, die über identische Prozessoren verfügen und über Kommunikationskanäle mit a priori bekannter Länge kommunizieren (beispielsweise Bus-basierte Systeme). Die hierfür erstellten Modelle basieren häufig auf der Idee, sowohl Prozessoren als auch die sie verbindenden Kommunikationskanäle als kommunizierende endliche Automaten (vgl. [17,15,205]) zu modellieren. Die Analyse der vorgestellten Algorithmen erlaubt dann Aussagen über die Dauer zur Etablierung einer Gruppe und zur Fehlererkennung (beispielsweise [56, 57]) in Abhängigkeit von der Netzgröße und der Topologie.

In vielen realen verteilten Systemen sind die engen zeitlichen Grenzen der synchronen Modelle nicht implementierbar. Gleichzeitig ist die völlige Loslösung von zeitlichen Randbedingungen wie im asynchronen Modell nicht hinreichend, um in verteilten Systemen zeitnah reagieren zu können. Es ist ein realistisches Modell gefordert, das neben theoretischen Aus-

sagen zur Implementierbarkeit aufgrund der empirisch fundierten Annahmen über heutige verteilte Systeme[7] auch eine Implementierung auf Basis dieser Systeme ermöglicht.

6.6 Das zeitgesteuerte, asynchrone Systemmodell

Durch den Zugriff auf lokale Uhren ist eine Zeitüberwachung bei der Nachrichtenübertragung auf fast allen heutzutage eingesetzten Rechnern möglich und gebräuchlich. Rechner mit lokalen Uhren sind somit zwar nicht synchron, eine Abschätzung der Nachrichtenlaufzeiten ist aber sehr wohl auf Basis dieser Uhren möglich. Typischerweise wird dieses "timeout"-Verfahren dazu benutzt, um bei verbindungsorientierten Protokollen einen Abbruch herbeizuführen, falls der Kommunikationspartner zu lange nicht auf Nachrichten reagiert (beispielsweise bei TCP).

Unter dem Namen "timed asynchronous distributed system model" [59] werden neuerdings diese Erkenntnisse auch dazu verwendet, Gruppenmanagement in asynchronen Systemen zu ermöglichen. Die eigentliche Idee ist hierbei, daß die Gruppenmitglieder periodisch Nachrichten austauschen und somit bekanntgeben, daß sie noch verfügbar sind. Bleiben diese Nachrichten eine bestimmte Zeit lang ("timed") aus, dann können die anderen Gruppenmitglieder von *einem Fehler ausgehen*. Diese GMPs können sich irren. Nach wie vor gelten die Ergebnisse aus [88], so daß nie sicher geschlossen werden kann, ob ein System ausgefallen ist oder nur vorübergehend überlastet ist.

GMPs, die nach dem "timed asynchronous distributed system model" arbeiten, stellen allerdings die zur Zeit pragmatischte Lösung für das *Gruppenmanagement* dar. Die anderen Ansätze zur *unzuverlässigen Fehlererkennung* (beispielsweise [3,5,44]) betonen die Herstellung des Konsens innerhalb der Gruppe. Ein Gruppenmanagement für überlebensfähige Systeme muß nach Auffassung des Autors vielmehr eine intakte Gruppe herstellen, als eine maximale Gruppe mit möglichst starker Übereinkunft.

Konkret bedeutet dies, daß das Gruppenmanagement die Gruppe neu berechnen muß, falls eine Instanz innerhalb einer gewissen Zeit nicht in der Lage ist, auf Nachrichten zu reagieren. Es ist dann völlig unerheblich[8], ob diese Instanz ausgefallen ist, kurzfristig überlastet ist oder unter einem transienten Fehler leidet. Für die Anwendung, die den Dienst der Gruppe benötigt, ist diese Instanz in jedem Fall nicht mehr von Nutzen. Das Gruppenmanagement muß diese Instanz daher aus der Gruppe ausschließen, um zu vermeiden, daß weiterhin Aufträge an diese Instanz vergeben werden.

---

[7] Heutige verteilte Systeme sind in vielen Fällen die oben genannten "Workstation-Cluster". Dies trifft auch auf die parallele Packet Screen zu. Diese Systeme sind eigentlich als "asynchron" zu klassifizieren. Eine definitive Ausfallerkennung ist für diese Systeme aber nicht entscheidbar [88, 67], so daß eine pragmatische Lösung mit Fehlentscheidungen leben muß.

[8] Für die Diensterbringung ist es völlig unerheblich. Natürlich muß die Ursache eines Fehlers/Problems gesucht werden. Dies ist aber nicht die Aufgabe des Gruppenmanagements.

Anstatt einer universellen Lösung zur Ausfallerkennung muß in überlebensfähigen Systemen die anwendungsspezifische Berechnung, also die Diensterbringung durch die Gruppe, im Vordergrund stehen. Ein frühzeitiger, möglicherweise fälschlicher Ausschluß eines intakten Gruppenmitglieds ist daher deutlich besser, als eine langanhaltende Ungewißheit mit ebenso lange andauernder Dienstverzögerung. Der Ansatz der zeitgesteuerten, asynchronen Systeme ist am besten als Grundlage für ein überlebensfähiges Gruppenmanagement geeignet.

Es wird letztendlich eine dienstspezifische, obere Schranke festgelegt, nach der ein Gruppenmitglied *als ausgefallen erklärt und aus der Gruppe ausgeschlossen wird*, sollte es sich zwischenzeitlich nicht als verfügbar zu erkennen geben [59, 85, 86, 169].

6.7 Zusammenfassung

In diesem Kapitel wurde argumentiert, daß durch die Parallelisierung die Ausfallwahrscheinlichkeit der Packet Screen erhöht wird. Ohne weitere Maßnahmen, dieser erhöhten Ausfallgefahr entgegen zu wirken, würde die Steigerung der Leistungsfähigkeit auf Kosten der Zuverlässigkeit erfolgen. Wie bereits eingangs argumentiert, ist eine solche einseitige Optimierung grundsätzlich zweifelhaft beim Entwurf aller nicht-trivialen verteilten Systeme.

Es wurde darüber hinaus gezeigt, welche qualitativen und quantitativen Auswirkungen der Ausfall einer Instanz der parallelen Packet Screen nach sich zieht: Es kommt zu systematischen Paketverlusten, unter denen besonders die verbindungsorientierten Kommunikationsbeziehungen leiden. Darüber hinaus sinkt die Leistungsfähigkeit durch den Ausfall unter die Leistungsfähigkeit einer Packet Screen mit gleich vielen, adäquat konfigurierten Instanzen ab. Auch hierfür wurden wieder Messungen eingesetzt, die als Grundlage für ein analytisches Modell zur Betrachtung der Auswirkungen von Instanz-Ausfällen dienten. Mit diesem Modell konnte gezeigt werden, daß eine Rekonfiguration des Selektionsalgorithmus nach einem Instanzausfall sowohl die funktionale Verfügbarkeit wieder herstellt, als auch die Leistungsfähigkeit entsprechend der neuen Anzahl an Instanzen optimiert.

Diese Rekonfiguration nach einem Ausfall erfordert jedoch den Einsatz zusätzlicher Mechanismen, und es wurde der Einsatz von Verfahren zur Erhöhung der Verfügbarkeit diskutiert. Durch ein Gruppenmanagement kann kontrolliert werden, welche Instanz zu welchem Zeitpunkt verfügbar ist. Der Ausfall einer oder mehrerer Instanzen kann erkannt werden, und die anschließende Rekonfiguration stellt wieder ein funktionsfähiges, verteiltes System her.

Das Gruppenmanagement ist ein wesentlicher Mechanismus im Bereich der fehlertoleranten Systeme. Die Einordnung in eine Reihe anderer, gängiger Mechanismen zur fehlertoleranten Auslegung von Systemen zeigte die typischen Abhängigkeiten dieser Mechanismen untereinander auf.

Das Gruppenmanagement ist somit ein konzeptioneller Dienst für den fehlertoleranten Systementwurf, muß aber, wie gezeigt wurde, in Abhängigkeit vom Systemmodell (synchron oder asynchron) betrachtet werden.

Schließlich wurde herausgestellt, daß für lokale Workstation-Cluster, wie beispielsweise die parallele Packet Screen, ein verläßliches Verfahren eingesetzt werden muß, das sowohl dem asynchronen Systemmodell gerecht wird, gleichzeitig aber auch innerhalb einer festzulegenden, relativ engen Zeitspanne entscheiden kann, ob Gruppenmitglieder ausgefallen sind. Zu diesem Zweck wurde das zeitgesteuerte, asynchrone Systemmodell vorgestellt: Bleiben Lebenszeichen einer Instanz über längere Zeit aus, wird diese Instanz als ausgefallen betrachtet. Dies mag ein Irrtum sein, verhindert aber, daß Gruppenmitglieder, die offensichtlich nicht in der Lage sind, einfache Nachrichten zu versenden, weiterhin als produktive Gruppenmitglieder geführt werden.

Im folgenden Abschnitt wird näher auf die speziellen Anforderungen der parallelen Packet Screen an ein Gruppenmanagement eingegangen.

Kapitel 7

Leichtgewichtiges, sicheres Gruppenmanagement-Protokoll

> " [...], the notion of adding redundancy and diversity is touted as a way of over"-coming reliability risks. Unfortunately, the added complexity itself often introduces new risks. Furthermore, distributed control could be a way of overcoming the risks of centralized systems. Unfortunately, distributed control often introduces many further risks. " —Peter G. Neumann [180]

Die Diskussion zur Ausfallsicherheit hat gezeigt, daß ein Gruppenmanagement-Protokoll benötigt wird. Die parallele Packet Screen ist ausfallgefährdeter als das ursprüngliche System. Sowohl zur Erhöhung der Verfügbarkeit als auch zur Steigerung der Leistungsfähigkeit ist daher auf Basis eines Gruppenmanagement-Protokolls die zeitnahe Reaktion auf Ausfälle und beendete Reparaturen einzelner Instanzen notwendig.

Bereits im letzten Abschnitt wurde darauf hingewiesen, daß in lose gekoppelten Netzen, d.h. in typischen verteilten Systemen, lediglich der zeitgesteuerte, asynchrone Ansatz sinnvoll und somit praktikabel ist.

Im folgenden Abschnitt werden die Defizite der vorhandenen Realisierungen solcher Protokolle besprochen, um den Entwurf und die Implementierung eines leichtgewichtigen Gruppenmanagement-Protokolls zu motivieren (vgl. Abschnitt 7.1).

In Abschnitt 7.2 wird anschließend ein einfaches Protokoll entworfen, das den parallelen Komponenten erlaubt, einen Ausfall oder die Reparatur einer Komponente zu erkennen und sich daraufhin dynamisch zu rekonfigurieren. Hauptziel bei diesem Entwurf wird es sein, einen möglichst leichtgewichtigen Ausfallerkenner zu spezifizieren, der die parallelen Instanzen möglichst wenig zusätzlich belastet, da die Ausfallerkennung der Erhöhung der Verfügbarkeit dient, gleichzeitig aber nicht der erreichten Leistungssteigerung entgegenwirken soll. Bei der Spezifikation von Protokollen für überlebensfähige Systeme wird vielfach auf formale Beschreibungen, formale Spezifikations- und Verifikationsmethoden zurückgegriffen. Ein Teil der Spezifikation wird daher auch unter zu Hilfenahme von formalen

Methoden durchgeführt, um die Vorteile und Nachteile dieser Vorgehensweise zu verdeutlichen. Das resultierende leichtgewichtige Gruppenmanagement-Protokoll (GMP) ist bereits in der Lage, aufwandsarm die Gruppenzugehörigkeit festzustellen, hat aber noch erhebliche Sicherheitsmängel. In Abschnitt 7.3 wird kurz auf die "klassische" Protokollvalidation eingegangen und in Abschnitt 7.4 wird die Verfügbarkeit der parallelen Packet Screen beim Einsatz des GMPs erneut bewertet.

Neben der Funktionalität für eine möglichst aufwandsarme Ausfallerkennung müssen zusätzliche Funktionen zur Erhöhung der Sicherheit in den Protokollautomaten eingebracht werden, da das Protokoll selbst abgesichert werden muß. So kann verhindert werden, daß die Erhöhung der Verfügbarkeit zu einer Herabsetzung der Sicherheit führt. Ein wesentlicher Aspekt des Entwurfs ist daher, geeignete Mechanismen zu spezifizieren, um den in Abschnitt 8.2 aufgeführten Sicherheitsanforderungen zu genügen. In Kapitel 8 wird daher in einer eingehenden Sicherheitsanalyse schließlich untersucht, welche Probleme im Detail auftreten können, falls ein Angreifer versucht, Schwächen des Protokolls auszunutzen. Diese Angriffsmöglichkeiten ("intrusion scenarios", vgl. Abschnitt 2.2.4) führen dann zu einem Katalog von zusätzlichen Mechanismen, die schließlich zu einer sicheren Implementation dieses GMPs führen.

7.1 Funktionsbedingter Aufwand beim Gruppenmanagement

Die Hauptaufgabe der Packet Screen ist die Filterung und Weiterleitung von Dateneinheiten. In Abschnitt 4.3 wurde daher ein Algorithmus zur dynamischen Zuordnung von einzelnen Paketen auf die verschiedenen Instanzen der parallelen Packet Screen gewählt. Die Meßergebnisse und die Leistungsanalyse aus Abschnitt 4.6 haben gezeigt, daß dieser Ansatz sehr gut die Paketverarbeitung beschleunigt.

Aufgrund der mit der Parallelisierung einhergehenden Erhöhung der Ausfallwahrscheinlichkeit (vgl. Abschnitt 6.1) wurde die Notwendigkeit für ein GMP motiviert. Eine Ausfallerkennung ermöglicht die dynamische Anpassung des Selektionsalgorithmus (vgl. Abschnitt 42) an die jeweils aktuelle Anzahl verfügbarer Instanzen. In Abschnitt 6.5 wurde schließlich motiviert, daß die sog. "timed asynchronous" GMPs konzeptionell sehr gut zu einem lose gekoppelten System wie das der parallelen Packet Screen passen.

Es stellt sich somit die Frage, warum nicht einfach ein Prototyp (beispielsweise aus [169, 200]) für die Ausfallerkennung bei der parallelen Packet Screen eingesetzt wird. Der Grund hierfür ist, daß diese Protokolle noch viel zu aufwendig sind und somit im Widerspruch zum Optimierungsziel "hohe Leistungsfähigkeit" stehen. Hierfür gibt es mehrere Faktoren, die im wesentlichen darauf zurückzuführen sind, daß diese Protokolle mit dem Anspruch entwickelt wurden, universelle Gruppenmanagement-Lösungen zu unterstützen:

- Unabhängigkeit von der Netztechnologie und der Topologie: Viele GMPs werden in Hinblick auf die Unabhängigkeit von einer bestimmten Netztechnologie entwor-

fen. Daraus folgt, daß bestimmte Mechanismen, die die Verwaltung der Gruppenzugehörigkeit erleichtern könnten, nicht zum Einsatz kommen. Ein Beispiel ist der Verzicht auf die "broadcast"- und "multicast"-Funktionen, die viele (aber nicht alle) Netztechnologien bieten (vgl. beispielsweise [101, 137]). Sucht man Realisierungen, die unabhängig von einer bestimmten Topologie sind, ergeben sich meist ähnliche Einschränkungen. So läßt sich die Gruppenzugehörigkeit in einer Ringtopologie sehr leicht und sogar "selbst-stabilisierend" feststellen (vgl. [66,231]). Das gleiche Problem ist für beliebige (heterogene) Topologien jedoch ungleich schwerer.

- Übereinstimmung bezüglich Gruppenzugehörigkeit: Das sog. "consensus"-Problem (beispielsweise [88, 5, 4]) stellt sich letztendlich bei jedem GMP. Die Frage ist, wie genau bzw. wie zeitnah erreicht ein Protokoll eine Übereinstimmung aller Gruppenmitglieder über die Gruppenzugehörigkeit. Dieses Problem hat verschiedene Ausprägungen: Zum einem kann gefordert werden, daß die Übereinstimmung innerhalb einer bestimmten Zeit nach Eintritt einer Veränderung erreicht wird. Zum anderen kann gefordert werden, daß schon beim Erreichen dieser einheitlichen Sicht zwei beliebige Gruppenmitglieder nie unterschiedliche Sichtweisen auf die Gruppenzugehörigkeit haben (vgl. [127]).

  Das Problem wird deutlich, wenn man sich vorstellt, daß zwei Prozessoren quasi gleichzeitig in die Gruppe aufgenommen werden möchten. Im ersten Fall beschränkt sich das "consensus"-Problem darauf, daß nach einer festzulegenden oberen Schranke $\triangle t$ alle Gruppenmitglieder (die bisherigen und die beiden neuen) die Menge aller Gruppenmitglieder kennen. Im zweiten Fall wird darüber hinaus gefordert, daß alle bisherigen Gruppenmitglieder übereinstimmend entweder zuerst den einen oder den anderen neuen Prozessor als neues Gruppenmitglied anerkennen. Es darf also nicht der Fall eintreten, daß zwei Gruppenmitglieder die Bewerber in unterschiedlicher Reihenfolge aufnehmen.

  Viele in der Literatur bekannte GMPs wählen daher den Ansatz, einen dedizierten Gruppenverwalter (auch "Leader" genannt) zu wählen. Dieser Gruppenverwalter stellt dann die Gruppenmitgliedschaft periodisch oder bei Bedarf fest. Dadurch wird es möglich, $\triangle t$ sehr klein zu halten. Außerdem kann dieser "Leader" mehrere neuen Bewerber sequentiell in die Gruppe aufnehmen, so daß es nicht zu unterschiedlichen Sichten auf die Gruppenzugehörigkeit kommt. GMPs basierend auf einem solchen "Leader" (vgl. beispielsweise [92, 222]) sind daher in der Literatur häufig anzutreffen, wenn hohe Anforderungen an die "Übereinstimmung" gestellt werden.

  Ein Nachteil an diesen GMPs mit Gruppenleiter ist, daß die Wahl zum Gruppenleiter ("leader election") selbst ein nicht-triviales algorithmisches Problem darstellt. Dies gilt insbesondere dann, wenn wieder die Unabhängigkeit von der Netztechnologie und der Topologie gefordert ist (s.o.) oder das Protokoll in anderer Hinsicht effizient sein soll (beispielsweise selbststabilisierend wie in [20]). Neben der Ernennung/Wahl eines Gruppenleiters ist darüber hinaus Vorsorge für den Ausfall eben dieses Gruppenleiters zu treffen. Dies erfolgt beispielsweise durch einen "Kronprinzen" (stellvertretenden

Gruppenleiter, vgl. [127,196]) oder dadurch, daß eine oder mehrere Gruppenmitglieder eine Überwachung des Gruppenleiters vornehmen (beispielsweise [127]). Alles in allem werden die GMPs hierdurch aufwendig.

Eine weitere Verschärfung des "consensus"-Problems ergibt sich bei der Anforderung, daß alle verfügbaren (d.h. nicht ausgefallenen) Prozessoren auch Mitglied der Gruppe werden. Die Verschärfung tritt dadurch ein, daß nach Ablauf von $\triangle t$ nicht nur eine Übereinkunft aller Gruppenmitglieder über die Zusammensetzung der Gruppe besteht, sondern es auch keinen weiteren Prozessor gibt, der noch nicht Mitglied der Gruppe ist, obwohl er sich rechtzeitig beworben hat [131].

- Erweiterte Funktionalität: Ein weiterer Faktor, der GMP komplizierter werden läßt, sind zusätzliche Funktionen. Neben der eigentlichen Aufgabe, die Zugehörigkeit zu einer Gruppe festzustellen, werden weitere Funktionen implementiert. Ein Beispiel für eine häufig anzutreffende zusätzliche Funktionalität ist der sog. "atomic broadcast". Hierbei erledigt das GMP sozusagen nebenbei die "atomare Auslieferung von Nachrichten" innerhalb der Gruppe (beispielsweise [8, 45, 57, 168]). Diese zusätzliche Funktionalität garantiert, daß jede Nachricht entweder alle Gruppenmitglieder erreicht oder gar kein Gruppenmitglied. Zusätzlich wird erreicht, daß die Reihenfolge des Nachrichtenempfangs bei allen Gruppenmitgliedern identisch ist. Diese Eigenschaft ist für bestimmte Prozeßgruppen nützlich (beispielsweise redundante Datenbankmanager), um eine konsistente Sicht —nicht nur auf die Gruppenzugehörigkeit— auch auf bestimmte, gruppeninhärente Datenstrukturen aufrecht zu erhalten.

  Ein solcher zusätzlicher Aufwand lohnt nur bei sehr aufwendigen oder schwer reproduzierbaren Berechnungen, die innerhalb einer Prozeß-Gruppe vorgenommen werden. Ein Beispiel hierfür sind transaktionsorientierte Systeme, wie verteilte Datenbanken. Außerdem muß beachtet werden, daß diese zusätzliche Funktionalität auch zu weiteren Angriffsmöglichkeiten führt [185], was wiederum Mehraufwand für eine Absicherung nach sich zieht.

- Komplexes Fehlermodell: GMPs werden teilweise mit dem Anspruch entwickelt, auf alle denkbaren Fehler reagieren zu müssen. Dies wird in der Literatur als das "Byzantine Fehlermodell" diskutiert [143, 205]. Ein realistisches Fehlermodell (vgl. Abschnitt 6.2.1), das nur die wahrscheinlich auftretenden Fehler berücksichtigt, kann zu einer deutlichen Verringerung des algorithmischen Aufwands beitragen, ist dann aber nicht mehr "universell" einsetzbar. Dieses komplexe Fehlermodell führt vor allem auch bei sicheren GMPs zu ganz erheblichem Aufwand. Nimmt man das "Byzantine Fehlermodell" an, bedeutet dies unter Sicherheitsaspekten betrachtet, daß jedes Gruppenmitglied grundsätzlich auch ein Angreifer sein kann, der falsche Nachrichten einspielt bzw. dessen Interesse es ist, den Gruppenkonsens zu stören. Fast alle "sicheren GMPs" implementieren daher mehrphasige Mechanismen, die Konsens auf Grundlage von Mehrheitsentscheidungen herbeiführen und zusätzlich aufwendige kryptographische Mechanismen (asymmetrische Kryptographie) einsetzen [136, 199, 195, 197, 196].

Die bisher entwickelten GMPs sind für die parallele Packet Screen viel zu aufwendig. Entscheidend für den Einsatz bei der parallelen Packet Screen ist, daß kurzfristig ein Verlust von einigen Dateneinheiten aufgrund eines Ausfalls toleriert werden kann. Datenverluste treten in Netzen ständig und aus den verschiedenartigsten Gründen auf. Typischerweise werden diese Verluste durch Mechanismen der Protokolle kompensiert (beispielsweise durch Wiederholung der Übertragung). Es ist daher nicht erforderlich, erweiterte Funktionalität ("atomic broadcast") zu implementieren, um beispielsweise während einer Rekonfiguration der parallelen Instanzen die zeitgleich eintreffenden Dateneinheiten zwischenzuspeichern.

Beim "consensus"-Problem kann ähnlich argumentiert werden: Die Gruppenzugehörigkeit muß im normalen Betrieb (keine Ausfälle, keine neuen Bewerber) selbstverständlich eindeutig sein. Schließlich wird über die Kenntnis der Gruppenzugehörigkeit auf jeder Instanz lokal der Verteilungsschlüssel für die Selektion der Dateneinheiten berechnet (vgl. Abschnitt 42). Treten jedoch fast gleichzeitig mehrere Ausfälle oder neue Bewerber auf, darf es kurzzeitig durchaus zu unterschiedlichen Sichten auf die Gruppenzugehörigkeit kommen. Es können dann zwar Paketverdoppelungen oder auch kurzzeitig Paketverluste auftreten; die Sicherheit[1] ist dadurch aber nicht akut gefährdet, da alle Instanzen dieselben Richtlinien umsetzen.[2] Es kann also ein leichtgewichtiger Algorithmus gewählt werden, der Konsens herstellt, ohne jedoch gleichzeitig kurzfristig abweichende Meinungen bezüglich der Gruppenzugehörigkeit zu verhindern.

Darüber hinaus kann der Aufwand für das Gruppenmanagement durch Ausnutzen der "Multicast"-Funktionen weiter verringert werden. Bereits für die Verteilung der Dateneinheiten an die verschiedenen Instanzen wurde die "Multicast-Fähigkeit" von (Fast)Ethernet ausgenutzt. Da somit die Netztechnologie ohnehin schon auf vergleichbare Zugriffsverfahren, die Gruppen-Kommunikation unterstützen, eingeschränkt ist, kann diese Eigenschaft gleichermaßen auch für eine Reduzierung des Nachrichtenaufkommens beim Gruppenmanagement benutzt werden.

Bei dem hier gewählten, pragmatischen Ansatz wird somit auf komplizierte Algorithmen verzichtet, um eine leichtgewichtige und dadurch effiziente Prozessorgruppen-Verwaltung zu ermöglichen. So kann erreicht werden, daß der Leistungsgewinn durch die Parallelisierung nicht gleich wieder für die nun ebenfalls notwendige Gruppenverwaltung aufgebraucht wird. Im folgenden Abschnitt wird der Entwurf dieses Protokolls diskutiert.

---

[1] Genau genommen ist die Verfügbarkeit (autorisierter Zugriff auf Daten und Betriebsmittel) als Teil des Sicherheitsaspektes während des Ausfalls gefährdet. Durch den Ausfall werden jedoch die Richtlinien nicht außer Kraft gesetzt. Dateneinheiten, die während des normalen Betriebs von der Packet Screen vernichtet worden wären, werden bei einem Ausfall immer noch vernichtet. Lediglich das Auditing ist vom Ausfall möglichweise betroffen.

[2] Es können somit nur ohnehin zulässige Dateneinheiten verdoppelt werden. Datenverluste können auch im normalen Betrieb auftreten.

7.2 Entwurf eines leichtgewichtigen Gruppenmanagement-Protokolls

Der Entwurf eines leichtgewichtigen GMPs beginnt mit einer Dienstspezifikation. Es werden die verschiedenen Dienstelemente diskutiert, die ein Benutzer zur Steuerung einer Instanz dieses GMPs benötigt. Danach werden einige Annahmen über die Umgebung formuliert, in der das Protokoll eingesetzt werden soll. Diese Annahmen sind erforderlich, um in einem ersten Entwurf konkrete Ereignisse benennen zu können, die nicht durch das Protokoll bzw. eine Instanz des Protokolls ausgelöst werden, auf die die Protokoll-Instanzen jedoch reagieren müssen.

Darauf erfolgt eine semiformale Diskussion der Ausfallerkennung (als Kernaufgabe des GM-Protokolls, vgl. Abschnitt 7.2.3) und der wesentlichen Mechanismen, die ein Protokoll zur Ausfallerkennung einsetzen muß (vgl. Abschnitte 7.2.3.1 und 7.2.3.2). Hieran schließt sich eine semiformale Beschreibung des Protokolls an (vgl. Abschnitt 7.2.4).

Anschließend erfolgt die eigentliche Protokollspezifikation, d.h. Syntax (PDU-Format), Semantik (Bedeutung der Informationen in den Dateneinheiten und die daraus folgenden Verarbeitungsschritte) und Timing (zeitlicher Ablauf bei der Kommunikation) werden beschrieben.

In beiden Fällen werden zur Spezifikation formale Methoden eingesetzt, die eine automatische Validierung bestimmter Eigenschaften und Mechanismen dieses Protokolls ermöglichen. Bei der Dienstspezifikation wird die halbformale, graphische Spezifikationsmethode "Message Sequence Charts, MSC" [122] verwendet. Diese Methode ergänzt (vgl. [125]) sich gut mit der für die Protokollspezifikation verwendeten, formalen, graphischen Spezifikationssprache "Specification and Description Language, SDL" [121]. Der Grund, diese beiden Spezifikationsmethoden einzusetzen, ist ihre hohe Verbreitung. Es handelt sich nicht um künstliche, ausschließlich in der Wissenschaft und Forschung eingesetzte Methoden. MSC und SDL werden seit langem auch von der Industrie in der Entwicklung eingesetzt. Eine Kurzeinführung in diese Methoden wird in Anhang A gegeben. Als weiterführende Literatur kann beispielsweise [123, 124, 125, 118, 81] empfohlen werden. Bei der Protokollspezifikation wird darüber hinaus auf eine graphische Darstellung für endliche Automaten zurückgegriffen, da auch dies eine etablierte und gut verstandene Spezifikationsmethode ist. Das Automaten-Modell ergänzt durch den höheren Abstraktionsgrad die SDL-Spezifikation.

Die oben dargestellte Vorgehensweise entspricht der "klassischen Protokollspezifikation", der schließlich eine Implementierung mit anschließender Verifikation[3] folgt (vgl. beispielsweise [120]).

[3] Der Autor ist der Auffassung, daß zwischen der Validation einer Spezifikation und der Überprüfung der Konformität einer Implementation mit eben dieser Spezifikation unterschieden werden muß. Der zweite Fall wird im folgenden als Verifikation (der Implementation) bezeichnet.

Bei der Protokollspezifikation für ein überlebensfähiges System müssen nach Auffassung des Autors jedoch weitere Schritte zwischen Spezifikation und Implementation als Teil einer Sicherheitsanalyse erfolgen. Die zusätzlich erforderlichen Schritte und die Vorgehensweise werden in Kapitel 8 als Teil der Sicherheitsanalyse für das nun zu entwerfende Protokoll dargestellt.

7.2.1 Dienstspezifikation Gruppenmanagement-Dienst

In Abschnitt 6.3 wurden Anforderungen an eine fehlertolerante parallele Packet Screen formuliert. Wesentlich war die automatische Erkennung eines Ausfalls, um eine Rekonfiguration veranlassen zu können. Diese Rekonfiguration könnte durch die Protokollinstanz selbst vorgenommen werden. Universeller ist die Lösung allerdings, wenn die Instanz einen Benutzer lediglich über die neue Gruppenzusammensetzung informiert und dieser Benutzer daraufhin entscheidet, welche Schritte einzuleiten sind.

Es ist also in Hinblick auf eine saubere Trennung zwischen einer dann generischen Gruppenverwaltung, und darauf aufbauenden beliebigen Prozeßgruppen-Benutzern erforderlich, eine Dienstschnittstelle einzuführen. An dieser Schnittstelle fordern die Benutzer die lokale Instanz auf, sich um die Aufnahme in eine Prozeßgruppe zu bemühen oder diese wieder zu verlassen. Der GM-Dienst versucht dieser Aufforderung durch entsprechende Protokollabläufe nachzukommen und informiert den Benutzer hierüber. Gibt es mehrere Prozesse auf einem Prozessor, kann ein Prozessor somit auch in mehreren Prozeßgruppen sein. Im folgenden wird jedoch nicht zwischen Prozeß- und Prozessorgruppen unterschieden. Ein Prozessor bzw. eine Instanz der Packet Screen soll maximal einer Prozessorgruppe angehören, so daß die Prozessorgruppe mit der Prozeßgruppe der Packet Screen Instanzen identisch ist.

Für den Prototypen werden folgende Dienstprimitiven vorgesehen:

JOIN_req: Der Benutzer fordert die lokale Instanz auf, einer Prozeßgruppe beizutreten. Da der Prototyp so simpel wie möglich gehalten werden soll, werden keine weiteren Parameter übergeben. Im allgemeinen Fall könnten hier Namen von Prozeßgruppen angegeben werden, falls ein Prozessor in mehr als einer Gruppe sein kann.

JOIN_ind(num,id): Der GM-Dienst teilt dem Benutzer mit, daß die lokale Instanz in eine Prozessorgruppe aufgenommen wurde. Als Parameter werden auch hier nur die nötigsten Informationen übergeben. *num* kennzeichnet die Gesamtzahl der Prozessoren in der Gruppe. *id* ist der Bezeichner der lokalen Instanz innerhalb dieser Gruppe. Dies sind gerade die für den in Abschnitt 42 vorgestellten Algorithmus zur verteilten Selektion der Dateneinheiten erforderlichen Parameter. Im allgemeinen Fall würde als Teil dieser Nachricht auch die Gruppenliste der Anwendung übergeben werden.

LEAVE_req: Mit dieser Aufforderung veranlaßt der Benutzer den GM-Dienst die Prozessorgruppe zu verlassen. Ein Administrator kann über diese Dienstprimitive beispielsweise eine Instanz für Wartungsarbeiten aus der Gruppe herausnehmen.

LEAVE_ind: Der GM-Dienst teilt dem Benutzer mit, daß die lokale Instanz nun nicht mehr zur Prozeßgruppe gehört. Bei einer allgemeinen Prozeßgruppenverwaltung sollte ein Benutzer immer damit rechnen, daß zwischen der Aufforderung (LEAVE_req) und der Bestätigung (LEAVE_ind) weitere Aufträge an die lokale Instanz gesendet werden. Die Semantik von LEAVE_ind ist somit: *ab jetzt ist die lokale Instanz nicht mehr in der Gruppe.*

STATUS_req: Zu jeder Zeit sollte es möglich sein, daß sich ein Benutzer über den augenblicklichen Zustand der Gruppe und der lokalen Instanz informiert. Hierzu wird die Dienstprimitive "STATUS_req" verwendet. Auch hier gilt wieder, daß bei einem allgemeiner spezifizierten GM-Dienst weitere Parameter übergeben werden müßten, beispielsweise der schon oben genannte "Name" einer Prozessorgruppe.

Diese Nachricht muß insbesondere auch dann von einer Instanz verarbeitet werden, falls noch an einer anderen Dienstaufforderung gearbeitet wird. Beispielsweise könnte eine solche Anfrage vom Benutzer kurze Zeit nach einem "JOIN_req" gestellt werden, um den Grund zu erfahren, warum ihm noch kein "JOIN_ind" gemeldet wurde.

STATUS_ind(state,num,id): Mit diesem Dienstelement informiert der GM-Dienst den Benutzer über den Zustand der lokalen Instanz. Der Parameter *state* zeigt den Zustand des Protokollautomaten an und *num* und *id* haben die gleiche Bedeutung wie beim "JOIN_ind".

Der Parameter *state* ist wichtig, um den Benutzer auch über mögliche Fehlerzustände informieren zu können und um anzuzeigen, ob *num* und *id* überhaupt gültige Werte liefern. Gehört die lokale Instanz beispielsweise keiner Prozeßgruppe an, haben *num* und *id* keinerlei Bedeutung. Im allgemeinen Fall würde auch mit jeder "STATUS_ind"-Nachricht die Gruppenliste an den Dienstnehmer übermittelt werden, da diese Nachricht auch unaufgefordert Auskunft über Veränderungen bei der Gruppenzusammensetzung gibt.

Diese sechs Dienstelemente bilden eine minimale Schnittstelle zwischen Benutzer und GM-Dienst. Offensichtlich ist, daß es die vier Dienstprimitiven (JOIN_req, JOIN_ind, LEAVE_req und LEAVE_ind) geben muß. Da dieser Dienst aber Teil eines überlebensfähigen Systems sein soll, ist es aufgrund der "Recognition"-Anforderung (vgl. Abschnitte 2.2.2–2.2.3) unbedingt erforderlich, mit Hilfe der Dienstelemente (STATUS_req, STATUS_ind) ein minimales Monitoring dieses essentiellen Dienstes zu ermöglichen.

In den folgenden Abbildungen werden Kommunikationsabläufe an der Schnittstelle zum GM-Dienst dargestellt. Der "Benutzer" ist ein beliebiger Prozeß (Benutzerprozeß BP), beispielsweise eine Anwendung, die eine graphische Oberfläche für die Konfigurierung der lokalen Instanz des GM-Dienstes implementiert. Abbildung 7.1 zeigt die Anmeldung bei der Gruppe und die Reaktion auf Status-Anfragen in Abhängigkeit vom Zustand der lokalen Instanz. In beiden Abbildungen dienen die eingetragenen Zustände (JOINED und NOTJOINED) lediglich der Illustration. Der vollständige Zustandsautomat wird weiter unten im Zuge der Protokollspezifikation vorgestellt.

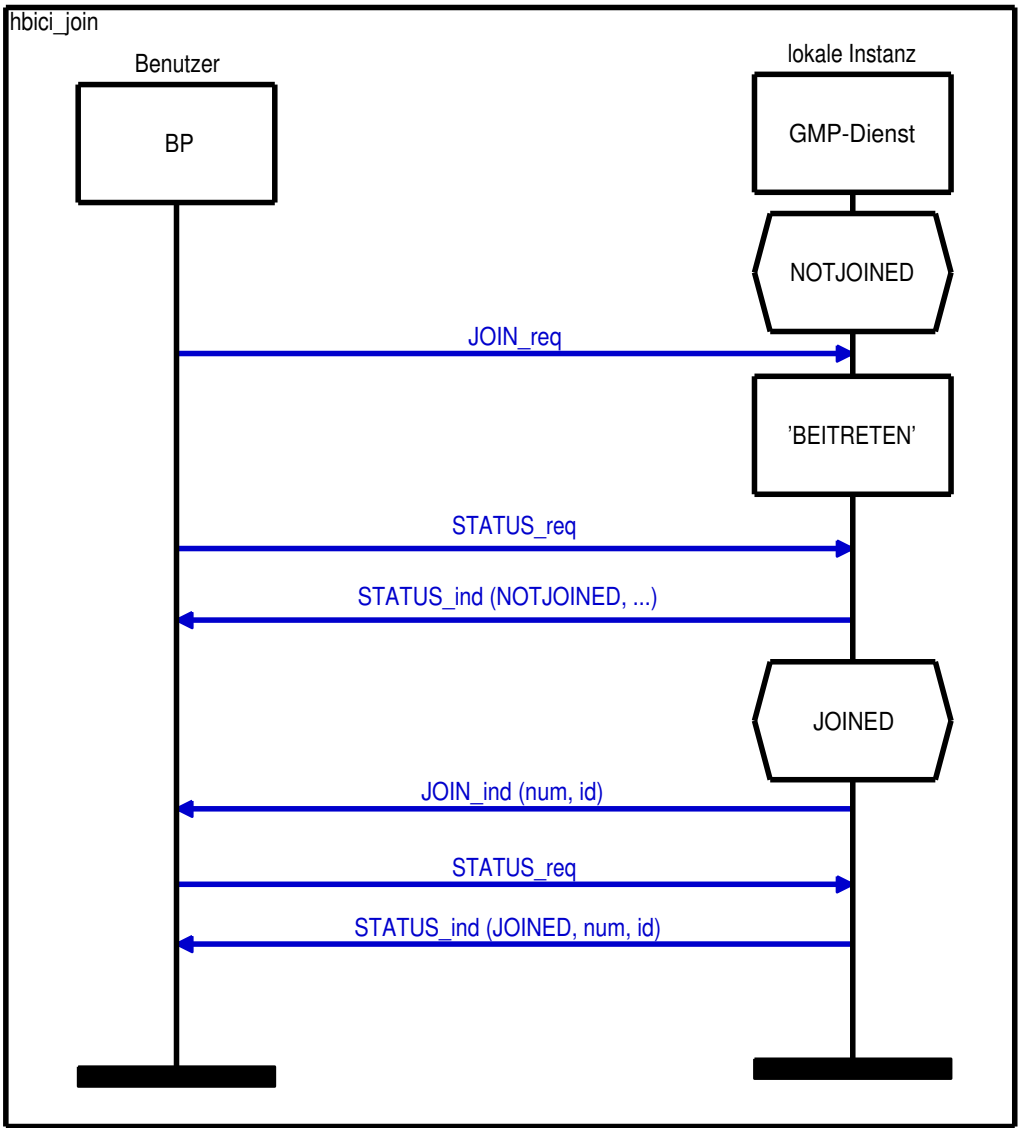


Abbildung 7.1: Benutzer initiiert Aufnahme in die Prozessorgruppe

Analog zur Aufnahme in die Gruppe erfolgt das Abmelden von der Gruppe. Auch hier kann wie in Abbildung 7.2 gezeigt, während der noch ausstehenden Abmeldung (nach LEAVE_req aber vor LEAVE_ind) eine beliebige Anzahl STATUS-Anfragen vom Benutzer gesendet werden. Während der Dienst sein Protokoll für die Abmeldung bemüht — angedeutet durch die Aktion 'AUSTRETEN' —, antwortet der Dienst weiterhin auf die STATUS-Anfragen und gibt den Zustand des Protokollautomaten bekannt (JOINED oder NOTJOINED).

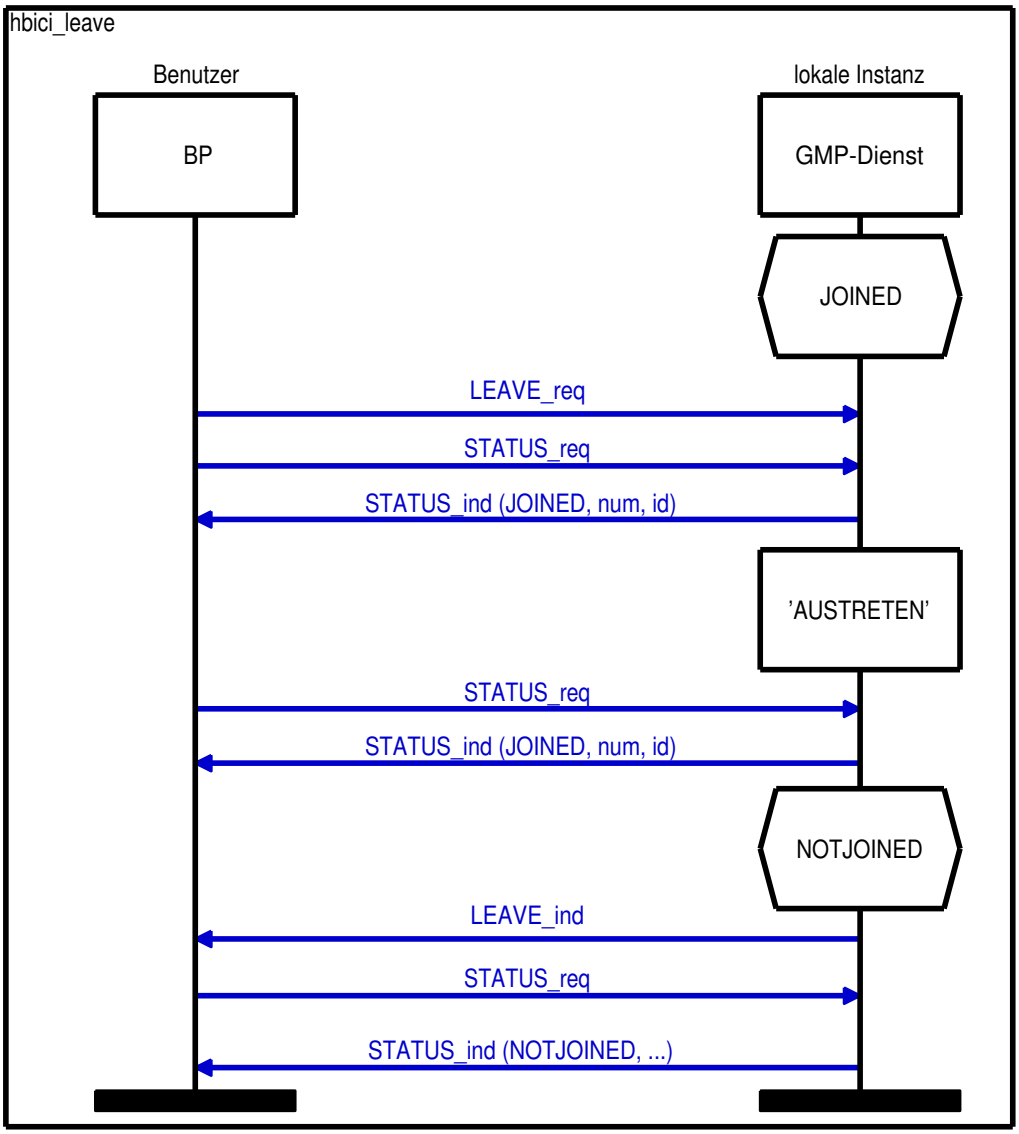


Abbildung 7.2: Benutzer initiiert Austritt aus der Prozessorgruppe

Die hier vorgestellten Abläufe sind noch nicht vollständig. Es wurden lediglich zwei mögliche Abläufe an der GM-Dienstschnittstelle vorgestellt, um die Semantik der Dienstprimitiven zu verdeutlichen. Die beiden vorgestellten Abläufe zeigen lediglich die Reaktion des Dienstes auf lokale Anfragen des Benutzers. Darüber hinaus muß der Dienst den Benutzer natürlich auch dann informieren, wenn sich die Anzahl der Instanzen ändert (Ausfall oder neuer Bewerber). Hierzu könnte sowohl ein "JOIN_ind" als auch ein "STATUS_ind" an den Benutzerprozeß gesendet werden. "STATUS_ind" ist jedoch universeller, da hiermit auch

lokale Probleme gemeldet werden können, die asynchron zu einer Gruppenveränderung auftreten. Beispiele für Probleme dieser Art wären das Erkennen eines lokalen Kabelbruchs oder das Erkennen eines Angriffs (vgl. Kapitel 8).

7.2.2 Technische Annahmen über die Umgebung

Es werden die folgenden technischen Annahmen über die Umgebung gemacht:

- Es wird angenommen, daß das Protokoll einen "Multicast"-Dienst benutzen kann, der eine Nachricht an eine festzulegende Gruppe von Kommunikationspartnern weiterleitet.
- Es wird angenommen, daß das benutzte Übertragungsmedium Nachrichten möglicherweise verliert, vertauscht oder dupliziert. Durch Übertragungsfehler verfälschte Nachrichten werden jedoch von einem "Link-Layer"-Protokoll erkannt und vernichtet.
- Nachrichten, die an eine Gruppe von Kommunikationspartnern gesendet werden, kommen demnach möglicherweise nur bei einigen oder sogar bei keinem Gruppenmitglied an (d.h. es gibt keinen "atomic broadcast").
- Es wird angenommen, daß Fehler im Übertragungsmedium durch ein "Link-Layer"-Protokoll erkannt werden und gemeldet bzw. abgefragt werden können.

Die eben aufgestellten Annahmen sind sehr realitätsnah. Die beschriebenen "Multicast/-Broadcast"-Eigenschaften sind typisch für heutige (Fast/Gigabit) Ethernetinstallationen. Auch beim Einsatz von ATM mit einer Umsetzung des "Multicast"-Dienstes mittels mehrerer "Punkt-zu-Mehrpunkt"-Verbindungen treffen obige Annahmen zu. Auch dort gibt es keine Garantie für eine Datenübertragung, verfälschte Dateneinheiten werden jedoch auf der ATM- oder der AAL-Ebene erkannt und vernichtet und Fehler im Übertragungsmedium werden durch das ATM-Management [219, S.211ff] automatisch erkannt und gemeldet. Ähnliche Funktionalität gibt es in Protokollen wie beispielsweise TokenRing oder FDDI [128, S.193ff].

7.2.3 Automatische Erkennung eines Ausfalls

Die wesentliche Aufgabe des Gruppenmanagement-Protokolls (GMP) besteht in der ständigen gegenseitigen Überwachung der einzelnen Instanzen. Das Ziel ist es, den Ausfall einer Instanz zeitnah zu erkennen (vgl. [105, 94, 44, 5, 4]), um eine möglichst aktuelle Liste von verfügbaren Gruppenmitgliedern zu pflegen. Natürlich ist es hierfür auch erforderlich, Mechanismen einzusetzen, die diese Gruppe etablieren und einen initialen Konsens herstellen. Das Hauptaugenmerk muß jedoch auf der Robustheit der Ausfallerkennung liegen. Die meiste Rechenzeit bei der Ausführung des Protokolls wird für diese Überwachungsfunktionen benötigt werden, da Gruppenneubildungen eher die Ausnahme sind.

Die Erkennung eines Ausfalls kann dadurch erreicht werden, daß die beteiligten Prozessoren periodisch Nachrichten versenden ("keep alive" Nachrichten oder "heartbeat" Signale). Bleiben diese Nachrichten ein bestimmte Zeit lang aus, kann man davon ausgehen, daß der entsprechende Prozessor ausgefallen ist. In diesem Fall kann eine Rekonfiguration eingeleitet werden.

Die Erkennung eines Ausfalls wirft jedoch einige Fragen auf. Es muß geklärt werden, wer den Ausfall einer Komponente erkennen soll. Wird eine Instanz ausgewählt, die die Erkennung leistet, dann wäre diese Instanz in der Lage, die Rekonfiguration der noch verfügbaren Prozessoren durchzuführen. Alternativ könnten sich die einzelnen Prozessoren auch gegenseitig überwachen und selbständig entscheiden, ob eine Rekonfiguration erforderlich ist.

Beide Ansätze haben sowohl Vor- als auch Nachteile. Der größte Nachteil der zentralen Instanz ist, daß sie selbst nicht redundant ausgelegt ist und somit ebenfalls ausfallgefährdet ist. Andererseits kann eine zentrale Instanz die einzelnen Prozessoren von dem zusätzlichen Aufwand einer gegenseitigen Überwachung befreien. Darüber hinaus ist es algorithmisch deutlich aufwandsärmer, durch eine zentrale Instanz die Rekonfiguration auf den verfügbaren Prozessoren anzustoßen. Da die Autorität für die Neukonfiguration ausschließlich bei der zentralen Instanz liegt, würde eine einzige Nachricht ausreichen, um auf den verbleibenden Prozessoren die Rekonfiguration *annähernd* zeitgleich auszulösen.

Ein weiterer wesentlicher Aspekt, der gegen einen zentralen Verwalter spricht, ist die herausgehobene Stellung dieses Verwalters in der Gruppe. Erstens ist der Verwalter aufgrund dieser außerordentlichen Stellung in der Gruppe am meisten gefährdet. Ein Angreifer kann potentiell den größten Schaden anrichten, wenn der Verwalter attackiert wird. Zweitens wäre der Aufwand des Verwalters für das Gruppenmanagement im Vergleich zu den anderen Gruppenmitgliedern größer. Das ist für sich genommen kein Problem, allerdings ist es ungünstig im speziellen Fall der parallelen Packet Screen, bei der eine lastunabhängige Verteilung der Aufträge an die Gruppenmitglieder vorgenommen wird (vgl. Abschnitt 42). Dadurch wäre der Prozessor, der zusätzlich den Verwalter-Prozeß auszuführen hat, überdurchschnittlich belastet.

Bei einer verteilten Überwachung entdeckt jeder Prozessor den Ausfall jedes anderen Prozessors und führt eigenständig eine Rekonfiguration durch. Der Hauptvorteil liegt darin, daß der Ausfall einer zentralen Überwachungsinstanz vermieden wird. Außerdem ist dieser Ansatz homogener, da eine Spezialisierung der Prozessoren vermieden wird (vgl. Abschnitt 7.1). Bei einer Gruppenneubildung fällt die gleiche Arbeit bei allen Gruppenmitgliedern an und es gibt keinen zentralen Schwachpunkt, der prädestiniert wäre für einen Angriff. Der Hauptnachteil dieser Lösung besteht in dem deutlichen Mehraufwand, um eine möglichst *synchrone* Rekonfiguration der verbleibenden Prozessoren nach einem Ausfall zu erreichen. Offensichtlich liegt die Schwierigkeit darin, daß die Prozessoren gleichzeitig erkennen, daß ein oder mehrere andere Prozessoren ausgefallen sind. Diese zeitgleiche Erkennung ist wiederum erforderlich, damit die Rekonfiguration ebenfalls zeitgleich erfolgt und es nicht durch "Nachzügler" zu inkonsistenter Verarbeitung kommt.

Für das zu entwickelnde Protokoll wird der dezentrale Ansatz gewählt, um ohne einen spezialisierten Prozeß auszukommen.

7.2.3.1 "Heartbeat"-Signale als "Multicasts"

Es wird ein einfacher (aufwandsarmer) Mechanismus benötigt, der es ermöglicht, den Ausfall einer Komponente zu erkennen. Da bereits auf "Broadcasts" bzw. "Multicasts" beim Entwurf der Ethernet-basierten parallelen Packet Screen zurückgegriffen wurde, kann durch diesen vom Netz angebotenen Dienst auch die Übertragung von periodischen "Heartbeat"-Signalen erfolgen. Durch Senden einer einzigen Dateneinheit kann ein Prozessor somit alle parallelen Prozessoren darüber informieren, daß er noch verfügbar ist. Der Aufwand für das Versenden von Nachrichten bleibt dadurch für die einzelnen Instanzen unabhängig von deren Anzahl konstant, da nicht jede Instanz einzeln adressiert werden muß. Für Hochgeschwindigkeitsnetze ist es ebenfalls kein Problem, die "Multi-" bzw. "Broadcast-" Dateneinheiten zu übertragen.

Bleiben die entsprechenden Signale aus, kann jeder Prozessor erkennen, welche Instanz ausgefallen ist. Hierzu ist es nur erforderlich, daß jeder Prozessor im periodischen "Heartbeat"-Signal (HB-Signal) seine eigene Kennung (ID) versendet. Die Feststellung, daß ein Prozessor defekt ist, darf allerdings nicht vom Ausbleiben eines einzigen HB-Signals abhängen. Ethernet garantiert die Übertragung von Dateneinheiten nicht, so daß bei Kollisionen oder durch Bitfehler ein HB-Signal eventuell nicht übertragen wird, obwohl der sendende Prozessor nicht ausgefallen ist.

7.2.3.2 Heuristik für die Ausfallserkennung

Es muß demzufolge ein Mechanismus gefunden werden, der auf Basis von regelmäßig versendeten Nachrichten, das *wiederholte* Ausbleiben dieser Nachrichten erkennt. Die grundlegende Heuristik liegt darin, das wiederholte Ausbleiben von HB-Nachrichten als Ausfall des Absenders zu werten, vereinzelt ausbleibende HB-Nachrichten jedoch als "omission/performance failures" (vgl. [58]) zu ignorieren.

Eine Instanz versendet an alle anderen Instanzen regelmäßig HB-Nachrichten und gibt somit bekannt, daß sie selbst noch aktiv, d.h. nicht ausgefallen, ist. Aus den empfangenen "Heart Beat"-Nachrichten der anderen Instanzen kann dann jede Instanz lokal die Gruppenzusammensetzung berechnen. Es ist wichtig, nicht sofort beim Ausbleiben einer Nachricht, den Ausfall der jeweiligen Instanz oder einen Kabelbruch anzunehmen. Entscheidend ist das "wiederholte" Ausbleiben der Nachrichten einer Instanz. Nur so kann ein Kabelbruch oder auch der Ausfall der Instanz ("crash failure") von dem Verlust einer oder mehrerer Nachrichten aufgrund von Überlastsituationen ("performance failure") unterschieden werden. Dieser Aspekt ist sehr wichtig, da die parallele Packet Screen gerade für Hochlastsituationen entwickelt wird. Ein hohes Nachrichtenaufkommen mit potentiellen "performance failures" ist daher sehr wahrscheinlich. Durch den folgenden Algorithmus

wird somit das Problem der "Nichtentscheidbarkeit" des Gruppenkonsens (vgl. [88]) umgangen. Es handelt sich um eine Heuristik, da nach einer bestimmten Zeit, die einer noch festzulegenden Anzahl ausbleibender Nachrichten entspricht, von den Gruppenmitgliedern *entschieden* wird, daß ein Gruppenmitglied ausgefallen ist (vgl. Abschnitt 6.6).

In diesem Abschnitt wird ein Algorithmus entwickelt, der die Erkennung eines Ausfalls der verteilten Komponenten sicherstellt. Die Schwierigkeit liegt hierbei in einer möglichst zuverlässigen Erkennung eines Ausfalls und läßt sich durch folgende Anforderungen charakterisieren:

1. Jeder Ausfall muß in kurzer Zeit von allen verfügbaren Prozessoren erkannt werden, um schnell eine Rekonfiguration durchzuführen. Die Erkennung sollte möglichst in "Echtzeit" erfolgen, d.h. es muß eine obere Schranke für die verstreichbare Zeit geben, bis ein Ausfall erkannt wird. Diese obere Schranke muß im Bereich weniger Sekunden liegen, um schnell die Rekonfiguration einleiten zu können.

2. Ein Ausfall sollte nur angezeigt werden, wenn tatsächlich eine Komponente ausgefallen ist. Es müssen geeignete Mechanismen gefunden werden, die die Wahrscheinlichkeit für einen falschen Alarm (fälschliche Annahme, ein Ausfall hätte stattgefunden) minimieren.

3. Die Ausfallerkennung muß hinsichtlich ihrer zeitlichen Annahmen konfigurierbar sein, um eine Anpassung an andere "Broadcast"-Netze zu ermöglichen. Das gesamte "Timing" der Ausfallerkennung sollte parametrisierbar gestaltet werden, um eine Anpassung an die Rechenleistung verschiedenartiger Endsysteme und Nachrichtenlaufzeiten in verschiedenen "Broadcast"-Netzen zu ermöglichen.

4. Die Ausfallerkennung muß in der Lage sein, trotz einzelner Verluste von "Heartbeat"-Signalen ("performance/omission failure") den Ausfall einer Komponente sicher zu diagnostizieren.

Entscheidend für die Verfügbarkeit ist die erste Anforderung. Wie in Abbildung 6.2 gezeigt wurde, führt der Ausfall einer einzigen Komponente zu erheblichen Leistungseinbußen bei der Kommunikation. Die Erkennung muß auf allen verfügbaren (nicht ausgefallenen) Komponenten schnell erfolgen.

Die zweite Anforderung soll verhindern, daß es zu vermeintlichen Ausfällen und anschließenden Rekonfigurationen kommt, weil fälschlicherweise ein Ausfall erkannt wurde. Da eine erneute Rekonfiguration wieder zu einer stabilen Gruppe führt, ist diese zweite Bedingung nicht ganz so kritisch wie die erste.

Die geforderte Parametrisierung kann leicht über die zwei Parameter $HB_{maxlost}$ und $HB_{intervall}$ erfolgen. Der erste Parameter gibt die Anzahl der tolerierbaren HB-Paketverluste an. Werden von einer Instanz mehr als $HB_{maxlost}$ aufeinanderfolgende HB-Pakete nicht empfangen, kann ein Ausfall angenommen werden. Der zweite Parameter $HB_{intervall}$ gibt das Zeitintervall an, in dem mindestens ein HB-Paket einer Instanz empfangen werden muß,

bevor ein Paketverlust angenommen wird. Im Mittel sollte also jede Instanz alle $HB_{intervall}$ Sekunden ein HB-Signal versenden. Die Latenzzeit $\Delta_l = (HB_{maxlost} + 1) * HB_{intervall}$ $[sec]$ gibt somit abhängig von der Wahl der beiden Parameter die Zeit für die Ausfallerkennung einer lokalen Instanz an. Die Abhängigkeit der Ausfallheuristik von diesen beiden zeitbezogenen Parametern macht das Protokoll zu einem zeitgesteuerten, asynchronen GMP (vgl. Abschnitt 6.6).

7.2.4 Semiformale Beschreibung der Funktionsweise des Protokolls

Das leichtgewichtige Gruppenmanagement-Protokoll (lGMP) soll zunächst die Anforderungen A1–A4 aus Abschnitt 6.3.2 erfüllen. Dies soll möglichst aufwandsarm, basierend auf den leichtgewichtigen Mechanismen zur Ausfallerkennung (vgl. Abschnitt 7.2.3), geschehen, da die Gruppenverwaltung eine zusätzliche Mehrbelastung ("Overhead") darstellt.

Die Anforderungen A1–A3 ergeben sich direkt aus dem Fehlermodell aus Abschnitt 6.2.1. Ausfälle von Netzkomponenten (A1) und Kabelbrüche (A2) sowie Ausfälle von Instanzen (A3) erkennt man durch das wiederholte Ausbleiben von Nachrichten (vgl. Abschnitt 7.2.3.2).

Für die weitere Diskussion ist folgende Definition erforderlich:

Definition (7.1): **Gruppenzusammensetzung**

> Eine Prozessorgruppe ist eine Menge G bestehend aus einer beliebigen Anzahl von Prozessoren, die über ihren Namen (Prozessorname) identifiziert werden.
>
> Die Gruppenzusammensetzung ist gegeben durch eine Prozessorgruppe G und eine Relation $\lessdot$ ("hat Priorität vor"), so daß für zwei verschiedene Prozessoren $p, q \in G$ gilt:
>
> $$\forall p, q, \text{ wobei } p \neq q : p \lessdot q \vee q \lessdot p$$

Die Festlegung der Priorität eines Prozessors ist hierbei beliebig bzw. implementationsspezifisch. Diese Gruppenzusammensetzung ermöglicht es jedem Prozessor, lokal eine Gruppenliste zu berechnen, die eine Reihenfolge von Gruppennamen auf der Prozessorgruppe G (totale Ordnung der Gruppenmitglieder) darstellt:

Definition (7.2): **Gruppenliste**

> Eine Gruppenliste $L = (p_0, p_1, p_2, \ldots, p_{n-1})$ bezeichnet die Anordnung der Prozessornamen einer Gruppenzusammensetzung der Prozessorgruppe G mit $|G| = n$, so daß gilt:
>
> $$\forall p_i, p_l \in G : p_i \lessdot p_l \rightarrow i < l$$

Außerdem sei die Funktion $ID : G \to \mathbb{Z}$ definiert durch

$$ID(p) = \begin{cases} i & \text{falls } p \in G \wedge p = p_i, \\ -1 & \text{falls } p \notin G \end{cases} \tag{7.3}$$

Das lokale Berechnen der Gruppenzusammensetzung hat gegenüber einer zentralen Berechnung dieser Zusammensetzung mit anschließender Verteilung der Information an die Gruppenmitglieder den Vorteil, daß keine Spezialisierung einzelner Prozesse vorgenommen werden muß. Es wird kein Gruppenverwalter mit den dazu notwendigen Algorithmen für die Gruppenverwalter-Wahl benötigt.

Da periodisch Nachrichten von allen Instanzen an alle anderen gesendet werden, wird zur Reduzierung des Nachrichtenaufkommens Gruppenkommunikation eingesetzt. (Fast)Ethernet und andere Protokolle unterstützen die Gruppenkommunikation bereits, so daß dieser Mechanismus aufwandsarm zur Verfügung steht.

Es gibt verschiedene Szenarien, wie mit Hilfe der "Heartbeats" auf Fehler geschlossen werden kann. Die verschiedenen Fehler erfordern auch eine unterschiedliche Reaktion der (verbleibenden) Instanzen.

7.2.4.1 Verhalten bei erkennbaren Fehlern

Aufgrund der heute üblichen aktiven Netzkomponenten ist ein Kabelbruch erkennbar. Ein über die typische "twisted pair"-Verkabelung an einen Switch angeschlossener Rechner kann durch Erfragen von Status-Information an seiner lokalen Netzschnittstelle erfahren, ob eine physikalisch intakte Verbindung zum Switch besteht. Eine Instanz kann somit durch periodisches Abfragen dieser Status-Informationen erkennen, ob ein Kabelbruch (A2) (vgl. Seite 140) oder ein Ausfall des Switches (A1) vorliegt. Die lokale Instanz kann allerdings den Kabelbruch nicht vom Ausfall des Switches unterscheiden.

Da die lokale Instanz jedoch in beiden Fällen der Gruppe nicht mehr nützlich ist, muß sie auch in beiden Fällen in einen Wartezustand übergehen, bis das Problem behoben wurde. Es ist also nicht erforderlich, aus lokaler Sicht (A1) und (A2) zu unterscheiden. Die betroffene Instanz stellt das Senden von "Heart Beat"-Nachrichten ein und wartet auf Reparatur.

Der Ausfall einer Instanz kann von den anderen Instanzen nur durch das wiederholte Ausbleiben der HB-Signale erkannt werden. Der Ausfall (A3) ist somit nicht unterscheidbar von einem Kabelbruch (A2) des Anschlusses der jeweiligen Instanz. Da aber in beiden Fällen die betroffene Instanz für die Gruppe nicht mehr von Nutzen ist ("crash failure" Semantik), können die anderen Instanzen die betroffene Instanz aus ihrer lokalen Gruppenzusammensetzung streichen.

7.2.4.2 Erhöhen der Robustheit I

Die oben diskutierten HB-Nachrichten teilen den anderen Instanzen bisher nur mit, daß die lokale Instanz noch aktiv ist. Die Robustheit der Ausfallerkennung kann weiter erhöht werden, wenn darüber hinaus die gesamte lokale Sicht auf die Gruppenzusammensetzung (Gruppenliste) mit jeder HB-Nachricht übertragen wird. Die anderen Instanzen können dann die verschiedenen Sichten abgleichen. Ein Vorteil dieser Methode ist beispielsweise, daß die Wahrscheinlichkeit für eine Fehlinterpretation bei ausbleibenden Nachrichten geringer ist. Wenn eine Instanz (I_α) von einer Instanz (I_β) wiederholt keine HB-Signale empfängt, müßte sie einen Ausfall der Instanz (I_β) annehmen. Empfängt Instanz (I_α) jedoch zwischenzeitlich eine Nachricht von Instanz (I_γ), in der Instanz (I_γ) mitteilt, daß Instanz (I_β) noch aktiv ist, sollte Instanz (I_α) weiterhin die Verfügbarkeit von Instanz (I_β) annehmen. Vielmehr deuten diese Verluste auf eine (kurzfristige) Häufung von "performance"-Fehlern hin, die jedoch noch keinen Ausfall anzeigen. Erst wenn keine Instanz mehr Instanz (I_β) in der mit den HB-Signalen versendeten Gruppenliste aufführt, sollten die Instanzen von einem Ausfall ausgehen. Diese zusätzliche Robustheit ergibt sich jedoch erst bei mehr als zwei Instanzen.

Über diesen Mechanismus kann eine Instanz darüber hinaus ein lokales Problem erkennen. Falls beispielsweise durch lokale Überlast wiederholt das Versenden von HB-Nachrichten nicht möglich war, dann erkennt die lokale Instanz dies bei Empfang von HB-Nachrichten anderer Instanzen, die die lokale Instanz nicht mehr in ihren Gruppenlisten aufführen. Die lokale Instanz sollte dann das Senden eigener HB-Nachrichten einstellen und die erneute Aufnahme in die Gruppe beantragen.

7.2.4.3 Aufnahme in die Gruppe

Soll eine weitere Instanz in die Gruppe aufgenommen werden, gibt es zwei Teilaufgaben zu lösen, um eine Übereinstimmung über die Gruppenzusammensetzung herzustellen.

Erstens muß die neue Instanz lernen, wieviele andere Teilnehmer es bereits in der Gruppe gibt und welche Position ($\lessdot$-Relation) die neue Instanz in der Gruppe haben wird, nachdem die Aufnahme abgeschlossen ist. Zweitens müssen alle anderen Gruppenteilnehmer darüber informiert werden, daß es einen neuen Bewerber gibt, der in die Gruppe aufgenommen werden möchte.

Ein einfacher Mechanismus, sowohl dem Bewerber vor der Aufnahme die Gruppenzusammensetzung mitzuteilen, als auch den Gruppenmitgliedern anzukündigen, daß ein neuer Bewerber in die Gruppe eintreten möchte, besteht in dem folgenden mehrstufigen Verfahren:

1. Der Bewerber sendet eine Nachricht per Multicast an alle bis dato noch unbekannten Gruppenmitglieder, daß er in die Gruppe aufgenommen werden möchte.

2. Jedes Gruppenmitglied sendet mit der nächsten HB-Nachricht eine Bestätigung, daß es den Aufnahmewunsch erkannt hat.

3. Aufgrund der Gruppenlisten in den HB-Nachrichten kann der Bewerber zweifelsfrei feststellen, wieviele Bestätigungen er braucht, bis alle Gruppenmitglieder den neuen Bewerber erkannt haben. Der Bewerber sendet seinen Beitrittswunsch ebenfalls periodisch, so daß bei Datenverlusten Gruppenmitglieder über eine darauffolgende Nachricht den Beitrittswunsch dennoch empfangen.

4. Hat der Bewerber von jedem Mitglied eine Bestätigung empfangen, kann er mit einer weiteren Multicast-Nachricht seinen Beitritt bekanntgeben.

5. Sollte diese Aufforderung zur Änderung der Gruppenzusammensetzung von einem Gruppenmitglied aufgrund von Datenverlusten oder anderen transienten Fehlern nicht empfangen werden, wird ihm spätestens mit dem nächsten HB-Signal die neue Gruppenzusammensetzung von irgend einem anderen Gruppenmitglied zugesendet.

Dieser einfache Algorithmus kommt ohne einen zentralen Verwalter aus, da aufgrund der Multicast-Kommunikation sichergestellt ist, daß jedes Gruppenmitglied seine lokale Sicht auf die Gruppenzusammensetzung aktualisieren kann.

7.2.4.4 Erhöhen der Robustheit II

Der oben vorgestellte Algorithmus ist bereits relativ simpel und robust. Die Robustheit kann jedoch weiter erhöht werden, indem Instanzen die Aufnahme in die Gruppe zurückstellen, falls kein Konsens bezüglich der Gruppenzusammensetzung herrscht. Dieser Fall tritt beispielsweise ein, wenn gerade ein Gruppenmitglied ausgefallen ist und dieser Ausfall noch nicht von allen Instanzen bemerkt wurde.

Hierfür ist die ständige Kontrolle der eintreffenden HB-Nachrichten durch einen Bewerber, der auf die Bestätigungen wartet, erforderlich. Falls ein Bewerber aufgrund der eintreffenden HB-Nachrichten feststellt, daß innerhalb der Gruppe keine einheitliche Sicht auf die Gruppenzusammensetzung besteht (keine Übereinstimmung), wechselt diese Instanz in einen Wartezustand und verhält sich für eine ausreichend lange Zeit passiv. Durch diesen Mechanismus wird erreicht, daß die Gruppe zuerst wieder Konsens herstellt und erst danach den neuen Bewerber aufnimmt.

Darüber hinaus muß ein weiterer Mechanismus dafür sorgen, daß nie zwei oder mehrere Instanzen quasi gleichzeitig die Aufnahme in die Gruppe beantragen. Hierbei könnte es zu unerwünschten Fluktuationen in den lokalen Gruppenzusammensetzungen der einzelnen Instanzen kommen.[4] Einen einfachen Mechanismus zur Serialisierung der Aufnahme

[4] An der Dienstschnittstelle würden sich diese schnell ändernden Gruppenzusammensetzungen durch eine Reihe von 'STATUS_ind'-Nachrichten vom HB-Dienst an den Benutzer äußern. Aus Sicht des Benutzers sind mehrere kurz aufeinanderfolgende Nachrichten mit Änderungshinweisen natürlich unerwünscht. Er ist nur an der letzten (abschließenden) Änderung interessiert. Da der Benutzer aber nicht weiß, daß unmittelbar nach einer Veränderung an der Gruppe eine oder mehrere weitere Veränderungen kommen, muß er jede einzelne Änderung nachvollziehen. Der Benutzer würde also zusätzlichen Overhead erzeugen, um seine lokalen Anpassungen (hier: die Rekonfiguration der Instanz) vorzunehmen, die einen Augenblick später bereits wieder revidiert werden müssen.

mehrerer neuer Instanzen in die Gruppe erhält man durch die Einführung von Prioritäten. Wollen zwei oder mehrere Instanzen gleichzeitig der Gruppe beitreten, dann darf die Instanz mit der höchsten Priorität zuerst beitreten, danach die Instanz mit der zweithöchsten Priorität usw.

Die Feststellung der Priorität soll auch wieder möglichst aufwandsarm erfolgen. In diesem Fall bietet sich wiederum der Prozessorname an. Wenn zwei verschiedene Prozessoren $(p, q, q \neq p)$ gleichzeitig versuchen, der Gruppe beizutreten, dann empfängt sowohl jedes Gruppenmitglied als auch beide neuen Bewerber die in Schritt (1) gesendeten Nachrichten von p und q. Jeder Prozessor kann nun lokal entscheiden, ob er p oder q in Schritt (2) bestätigt. Falls gilt $p \lessdot q$, dann wird p's Aufnahmeantrag bestätigt, andernfalls wird q's Aufnahmeantrag bestätigt.

Gleichzeitig können über dieselbe Relation sowohl p als auch q feststellen, wer von beiden höhere Priorität hat. Dies ist möglich, da auch in Schritt (1) wiederum die Multicast-Kommunikation verwendet wird und somit nicht nur alle Gruppenmitglieder, sondern auch alle potentiellen Gruppenmitglieder diese Nachrichten empfangen können. Falls gilt $p \lessdot q$, dann wird p den Algorithmus mit Schritt (3) fortsetzen und q muß warten, bis p in die Gruppe aufgenommen worden ist. q könnte dazu in einen Wartezustand wechseln und sich für eine ausreichend lange Zeit[5] passiv verhalten. Danach beginnt q wieder mit Schritt (1). Dieser Algorithmus funktioniert auch mit mehr als zwei gleichzeitigen Bewerbern. Allgemein gilt, daß sich der Bewerber mit der höchsten Priorität durchsetzt und in die Gruppe aufgenommen wird. Da er nun neues Gruppenmitglied und nicht mehr Bewerber ist, setzt sich in der nächsten Runde wiederum der Bewerber mit der höchsten Priorität durch. Durch diese Sequentialisierung der Aufnahme in die Gruppe können die oben beschriebenen Fluktuationen verringert werden.

Ein weiterer Sonderfall tritt ein, falls es trotz der bisher diskutierten Mechanismen zu einer inkonsistenten Sicht auf die Gruppenzusammensetzung kommt. Tritt beispielsweise der Fall ein, daß eine verfügbare Instanz (I_γ), die auch regelmäßig HB-Nachrichten sendet, in der Gruppenliste einer Instanz (I_α) enthalten ist, nicht jedoch in der Gruppenliste einer anderen Instanz (I_β), so daß gilt:

$$\gamma \in G_{I_\alpha}, \gamma \notin G_{I_\beta} \text{ wobei } \alpha \in G_{I_\beta} \wedge \beta \in G_{I_\alpha} \wedge \alpha, \beta \in G_{I_\gamma} \quad , \tag{7.4}$$

dann liegt offensichtlich eine inkonsistente Gruppenzusammensetzung vor. Dieser Fall kann eigentlich nur aufgrund eines bisher noch nicht antizipierten Fehlers bei der Protokollverarbeitung auftreten, der die Instanz I_β dazu veranlaßt hat, den Prozessor γ aus der Gruppenliste zu streichen. Eine robuste Implementation sollte hierauf reagieren können, zumal alle beteiligten Instanzen aufgrund der versendeten Gruppenlisten diesen Dissens erkennen können.

[5] Das genaue "Timing" wird im Zuge der Spezifikation weiter unten festgelegt. Die Wartezeit sollte relativ zur Intervallänge ($HB_{intervall}$) eingestellt werden. Sie sollte deutlich größer als die Zeit zwischen zwei HB-Nachrichten sein, um der anderen Instanz genügend Zeit einzuräumen, ausstehende Bestätigungen mit weiteren HB-Nachrichten zu empfangen.

Dies erfordert entweder eine Rekonfiguration von I_β, so daß Prozessor γ bzw. Instanz $I_\gamma \in G_{I_\beta}$ oder Instanz I_γ erkennt (aufgrund der HB-Nachrichten von I_β), daß ihr Prozessor $\gamma \notin G_{I_\beta}$. In diesem Fall könnte I_γ in einen Wartezustand gehen, so daß alle Gruppenmitglieder I_γ aus ihrer Gruppenliste streichen. I_γ könnte sich danach erneut bewerben. Da jedoch I_γ nicht defekt ist und somit ohnehin wieder der Gruppe beitreten würde, sollte die erste Reaktion gewählt werden. I_β nimmt I_γ in die eigene Gruppenliste auf und führt eine lokale Rekonfiguration durch. Da I_β, wie alle anderen Instanzen auch, die Gruppenliste von I_γ empfängt, in der sie selbst enthalten ist, kann I_β ein andauerndes Problem als Ursache für den Dissens ausschließen. Die Kommunikation kann nicht gestört sein, da HB-Nachrichten von I_β bereits von I_γ empfangen wurden, andernfalls könnte I_β nicht in der Gruppenliste von I_γ enthalten sein.

7.2.5 Protokollspezifikation

In diesem Abschnitt wird die eigentliche Protokollspezifikation durchgeführt. Dabei wird zuerst die grobe Struktur des Protokolls mit Hilfe eines endlichen Automaten entworfen. Diese Grobstruktur enthält bereits die wichtigsten Zustände einer Instanz des GM-Protokolls und die von ihr zu verarbeitenden Nachrichten. Anhand dieser Grobstruktur können bereits wesentliche Mechanismen des Protokolls diskutiert werden.

Anschließend erfolgt eine genaue Spezifikation unter Verwendung von SDL. Aus dieser Spezifikation kann ein ablauffähiges Modell des Protokollautomaten generiert werden. Mit diesem Modell können verschiedene Aspekte des Protokolls validiert werden, beispielsweise das Erreichen einer einheitlichen Sicht auf die Gruppenzugehörigkeit.

7.2.5.1 Protokollautomat

Jede Instanz benötigt einen Startzustand (“S”), in dem eine Initialisierung des Protokollautomaten vorgenommen wird. Drei weitere Zustände werden für die Unterscheidung zwischen Bewerber (“Want-Join-Sent” und “HB-Rec”) und Gruppenmitglied (“Wait-Beat”) benötigt. Ein weiterer Zustand wird für passives Abwarten eingeführt (“Backoff”). Der Protokollautomat wechselt in diesen Zustand, falls eine Reparatur erforderlich ist oder eine andere Instanz Vorrang bei der Aufnahme in die Gruppe hat. Schließlich wird noch ein Zustand für die Berechnung der aktuellen Gruppenliste benötigt (“Calc-Conf”).

In den folgenden Abbildungen sind an den Pfeilen, die die Zustandsübergänge symbolisieren, die eintreffenden und abgehenden Nachrichten eingezeichnet. Die Signale für die Dienstelemente aus Abschnitt 7.2.1 wurden nicht eingezeichnet, um die Abbildung nicht mit zu vielen Beschriftungen zu überladen. Der Protokollautomat zeigt daher nur die Interaktion mit anderen Instanzen, nicht jedoch die Interaktion mit dem Benutzer. Die Beschriftungen genügen folgender BNF:

```
IO:= INLIST '/' OUTLIST
```

```
INLIST:= LIST | EMPTY
OUTLIST:= LIST | EMPTY
LIST:= SIGNAL | SIGNAL ',' LIST
SIGNAL:= TIMER | MESSAGE
MESSAGE:= INTERN | EXTERN
TIMER:= 'timeouts' | 'timeoutb' | 'timeouth'
EXTERN:= 'WJ' | 'NonPRI-WJ' | 'PRI-WJ' | 'JM' | 'HB'
INTERN:= 'reconf' | 'c' | 'nc'
EMPTY:= '-'
```

Eine Beschriftung (IO) besteht aus einer Liste von eintreffenden Nachrichten, die einen Zustandswechsel auslösen. Danach folgt ein Trennsymbol ('/') gefolgt von einer Liste von ausgehenden Nachrichten, die im Moment des Zustandsüberganges abgesendet werden. Spontane Übergänge sind durch eine leere 'INLIST' bzw. durch das Symbol "-" angedeutet. Diese Beschriftung erfolgt in Anlehnung an die Beschriftungen in [207, 103, 51].

Weiterhin wird zwischen internen und externen Nachrichten sowie abgelaufenen Zeitgebern ('TIMER') unterschieden. Die internen Signale sendet der Protokollautomat an sich selbst. Die internen Signale werden zur besseren Unterscheidung klein geschrieben. Externe Signale werden an andere Instanzen versendet oder von diesen empfangen. Externe Signale werden in der Abbildung 7.3 groß geschrieben.

(a) Anmelden in der Gruppe (Bewerber) Nach einer Initialisierung im Start-Zustand ("S") sendet die Instanz eine "Want-Join"-Nachricht ("WJ") und geht in den Zustand "Want-Join Sent" über. In diesem Zustand wartet die Instanz darauf, ob ein zuvor gesetzter Zeitgeber abläuft und das interne Signal "timeouts" sendet. In diesem Fall weiß die Instanz, daß sie bisher die einzige Instanz in der Gruppe ist. Sie sendet (pro forma) eine "Join-Me"-Nachricht ("JM") und eine erste "Heart-Beat"-Nachricht ("HB") und geht in den Folgezustand "Wait-Beat" über. Dieser Ablauf entspricht den Schritten (1) gefolgt von (5) aus Abschnitt 7.2.4.3 und ist nur möglich, falls die Gruppe bisher noch nicht existiert.

Empfängt die Instanz im Zustand "Want-Join Sent" eine HB-Nachricht, sendet sie ein weiteres "WJ" und geht in den Zustand "HB-Rec" über. In diesem Zustand sammelt sie weitere HB-Nachrichten und prüft, ob die anderen Instanzen dem Aufnahmeantrag zustimmen (Schritte (1), (2) und (3)). Hat die Instanz von allen anderen Instanzen eine Bestätigung empfangen, wird das interne Signal "consens" ("c") generiert und die Instanz sendet eine "Join-Me"-Nachricht ("JM") und eine erste "Heart-Beat"-Nachricht ("HB") und geht in den Folgezustand "Wait-Beat" über.

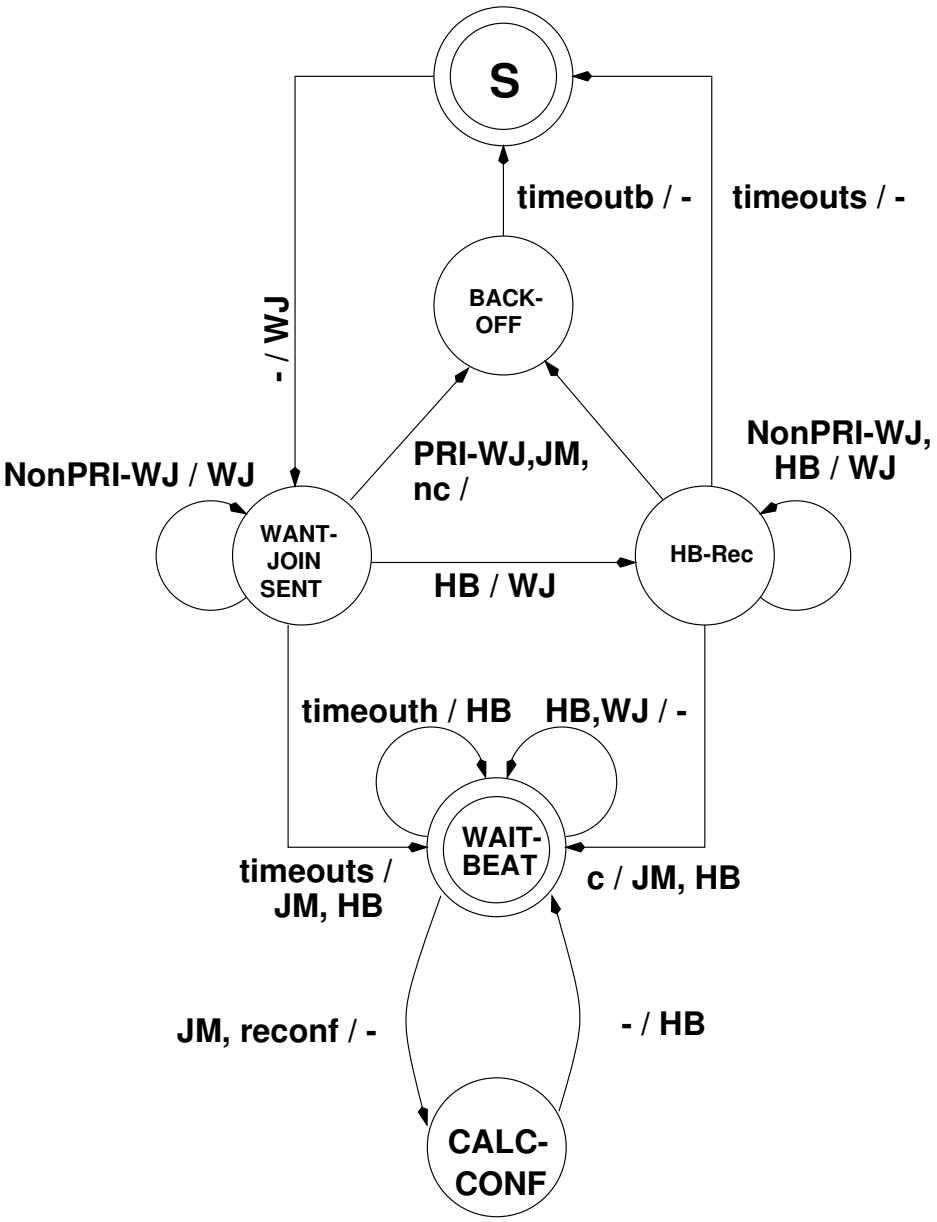


Abbildung 7.3: Zustandsautomat für ein leichtgewichtiges GMP

Es kann auch passieren, daß in den eintreffenden HB-Nachrichten verschiedene lokale Sichten (Gruppenlisten) mitgesendet werden. In diesem Fall erkennt die Instanz, daß z.Z. kein Konsens herrscht ("no consens"-Signal, ("nc")) und geht in den Wartezustand über. Emp-

fängt eine Instanz eine priorisierte[6] "Want Join"-Nachricht oder eine "Join Me"-Nachricht einer anderen Instanz, geht sie ebenfalls in den Wartezustand über ("Backoff"), da eine andere Instanz mit höherer Priorität gleichzeitig in die Gruppe aufgenommen werden möchte oder die Aufnahme einer anderen Instanz von den bisherigen Gruppenmitgliedern schon bestätigt wurde. Der Wartezustand wird nach Ablauf eines Zeitgebers, der das Signal "timeoutb" generiert, wieder verlassen. Die Instanz geht dann in den Start-Zustand über und versucht erneut die Aufnahme in die Gruppe.

Ein "Want-Join" von einer nicht-priorisierten Instanz ("NonPRI-WJ") wird sofort mit einem eigenen "Want-Join"-Signal beantwortet, damit die andere Instanz bemerkt, daß sie versucht, gleichzeitig mit einer höher priorisierten Instanz in die Gruppe aufgenommen zu werden. Diese WJ-Nachricht wird von der anderen Instanz beim Empfang als "PRI-WJ"-Nachricht interpretiert.

Hat eine Instanz im Zustand "HB-Rec" noch nicht alle Bestätigungen empfangen und der lokale Zeitgeber läuft ab (Signal "timeouts"), geht die Instanz wieder in den Start-Zustand und beginnt von neuem. Dieser Mechanismus verhindert eine Verklemmung, falls nach Empfang eines Teils der Bestätigungen alle anderen Instanzen ausfallen. Da in diesem Fall keine weiteren Nachrichten einträfen, würde die Instanz ohne den lokalen "Timeout" im Zustand "HB-Rec" verbleiben. Dies wäre ein klassischer "Deadlock" (Verklemmung), bei dem auf ein Ereignis gewartet wird, das nie eintritt.

(b) Anmelden in der Gruppe (Gruppenmitglied) Empfängt eine Instanz im Zustand "Wait-Beat" eine "WJ"-Nachricht eines Bewerbers, so bestätigt sie diese Nachricht mit der nächsten "HB"-Nachricht (Schritt (2)). Dazu muß sich jede Instanz den Prozessornamen des Bewerbers merken. Treten mehrere Bewerber gleichzeitig auf, bestätigt die Instanz nur dann die Nachricht, falls es noch keinen Bewerber gibt, oder der neue Bewerber höhere Priorität hat. Bewerber mit höherer Priorität verdrängen bei der Bewerbung somit zeitgleiche Bewerber mit niedrigerer Priorität.

Eine eintreffende "Join-Me"-Nachricht überführt den Protokollautomaten in den Zustand "Calc-Conf", in dem die neue Gruppenliste berechnet wird. Danach sendet die Instanz eine HB-Nachricht mit der neuen lokalen Gruppenliste und geht wieder in den Zustand "WB" über.

Das interne Signal "reconf" überführt die Instanz in den Zustand "Calc-Conf", falls beim Auswerten eintreffender HB-Nachrichten eine uneinheitliche Sicht auf die Gruppenzusammensetzung bemerkt wird. Dieser Fall tritt beispielsweise dann ein, wenn eine Instanz ein "Join-Me"-Signal eines neuen Bewerbers nicht erhält. Sie erfährt die neue Gruppenzusammensetzung mit einem der nächsten eintreffenden "HB"-Nachrichten (Schritt (5)).

[6] Es gibt nur einen Nachrichtentyp ("WJ"). Es ist Aufgabe der einzelnen Instanzen, empfangene WJ-Nachrichten entsprechend der $\lessdot$-Relation als "priorisiert" oder "nicht-priorisiert" zu klassifizieren. Die Beschriftung "PRI-WJ" bedeutet daher: Es wurde eine WJ-Nachricht von einer Instanz empfangen, die höhere Priorität hat. Analog bedeutet die Beschriftung "NonPRI-WJ": Es wurde eine WJ-Nachricht von einer Instanz empfangen, die geringere Priorität hat.

(c) Gegenseitiges Überwachen der Gruppe Im Zustand "WAIT-BEAT" angelangt, senden die einzelnen Instanzen mit einer Periodizität von $HB_{intervall}$-Sekunden "HB"-Nachrichten; ausgelöst durch einen periodisch ablaufenden Zeitgeber, der das interne Signal "timeouth" generiert. Die anderen Instanzen empfangen diese "HB"-Nachrichten und können so feststellen, ob noch Konsens bezüglich der Gruppenzusammensetzung herrscht.

Bleiben von einer Instanz mehr als $HB_{maxlost}$ aufeinanderfolgende "HB"-Signale aus, dann streichen die anderen Instanzen sie aus der Gruppenliste, die sie mit ihren HB-Nachrichten versenden. Damit wird den empfangenden Instanzen klar, welche Instanz von welchen anderen Instanzen noch "gesehen" wird. Erst wenn keine Instanz mehr die potentiell ausgefallene Instanz in ihren HB-Nachrichten in der Gruppenliste führt, wird das lokale Signal "reconf" ausgelöst. Die Instanz durchläuft den Zustand "CALC-CONF" und berechnet die neue Konfiguration. Die Latenzzeit bis zur Neuberechnung der Gruppenzusammensetzung nach tatsächlichem Ausfall einer Instanz beträgt demnach $(HB_{maxlost} + 1) * HB_{intervall}\ [sec]$. Darüber hinaus haben die verbleibenden Instanzen nach maximal $(HB_{maxlost} + 3) * HB_{intervall}\ [sec]$ wieder Konsens hergestellt.

Fallen zwei oder mehr Instanzen innerhalb von $(HB_{maxlost} + 1) * HB_{intervall}\ [sec]$ aus, dann wird auch in diesem Fall rekonfiguriert. Das Ausbleiben von Nachrichten einer dritten Instanz kann eine Rekonfiguration nicht verhindern, da die Entscheidung zur Rekonfiguration immer lokal gefällt wird. So ist sichergestellt, daß es zu keiner Verklemmung durch gleichzeitigen Ausfall mehrerer Instanzen kommen kann. Andererseits kann eine eintreffende Nachricht die Rekonfiguration verhindern, falls sie Informationen enthält, daß eine lokal als ausgefallen klassifizierte Instanz noch aktiv ist. Dies wurde bereits in Abschnitt 7.2.4.2 diskutiert.

Die ablaufenden Zeitgeber generieren die Signale "timeouts", "timeoutb" und "timeouth". Dies sind die wesentlichen Mechanismen, die einen Fortschritt des Protokolls trotz asynchroner Kommunikation garantieren. Durch diese Mechanismen wird das Protokoll zu einem zeitgesteuerten, asynchronen GMP (vgl. Abschnitt 6.5). Der skizzierte Protokollautomat strebt den akzeptierenden Zustand "WAIT-BEAT" an. Dies ist der Hauptzustand, in dem die etablierte Gruppe von allen verteilten Instanzen überwacht wird.

7.2.5.2 Zeitlicher Ablauf

Von wesentlicher Bedeutung für eine Protokollspezifikation ist neben der Beschreibung der Syntax (PDU-Format, s.u.) und der Semantik (Protokollautomat) auch der zeitliche Ablauf ("Timing") bei der Kommunikation.

In der obigen Diskussion wurde bereits darauf hingewiesen, daß die einzelnen Instanzen periodisch "Heart Beat"-Nachrichten versenden, um die eigene Verfügbarkeit anzuzeigen. Dies geschieht im wesentlichen im Zustand "Wait Beat", der als normaler Betriebszustand anzusehen ist, den alle Instanzen anstreben (formal: "Wait-Beat" ist ein akzeptierender Zustand des Protokollautomaten). In diesem Zustand wird ein Zeitgeber (im folgenden

auch Timer genannt) dazu verwendet, periodisch ($1/HB_{intervall}\ [Hz]$) ein "timeouth"-Signal zu generieren, das die Instanz veranlaßt, eine aktualisierte "HB"-Nachricht zu versenden.

Da die gesamte Entscheidungslogik, ob eine Instanz noch verfügbar ist oder als ausgefallen angesehen werden muß, direkt mit der Intervallgröße $HB_{intervall}$ zusammenhängt, muß das verbleibende Timing in Abhängigkeit vom $HB_{intervall}$ (bzw. vom steuernden Zeitgeber) gewählt werden. Im folgenden wird diskutiert, welche Randbedingungen bei der Wahl von $HB_{intervall}$ beachtet werden müssen.

Das Timing muß für alle Instanzen gleich sein. Da hierfür auf systemspezifische Implementierungen von Zeitgebern zurückgegriffen wird, die aufgrund verschiedener Realisierungen in Hardware oder Software unter einer Zeitdrift leiden, muß das Timing robust ausgelegt werden. Dies kann folgendermaßen erreicht werden:

- Die maximale Auflösung der Zeitgeber eines Rechners/Prozessors (Anzahl der unterscheidbaren Uhrveränderungen pro Sekunde) sei $1/\sigma$ (Einheit ist Hz, also s^{-1}). Aktualisiert der Zeitgeber alle σ Sekunden die interne Uhr, dann sollte für zwei vom Protokoll zu unterscheidende, aufeinanderfolgende Timer-Ereignisse zu den Zeitpunkten (t_1, t_2) gelten:

  $$\forall t_1, t_2,\ \text{wobei}\ t_2 > t_1 \geq 0 : t_2 - t_1 > \sigma\ [sec] \tag{7.5}$$

  Hiermit wird sichergestellt, daß interne, Timer-gesteuerte Ereignisse vom Protokollautomaten unterscheidbar sind.

- Der minimale Zeitabstand zwischen zwei Ereignissen muß jedoch größer sein, so daß implementationsspezifische Zeitabweichungen der lokalen Uhren verschiedener Rechner kompensiert werden. Typische Endgeräte verfügen heutzutage über hochauflösende Zeitgeber, die in der Lage sind, im Bereich von $\sigma \approx 100$ Nanosekunden die Rechneruhren zu aktualisieren. Gleichzeitig tritt aber eine Zeitdrift zwischen zwei verschiedenen Rechnern innerhalb dieser Zeit auf. Ähnlich wie in [59] muß man daher eine zusätzliche Zeitdrift τ mit der Einheit $[\mu s/s]$ annehmen, um die die Zeitangabe jeder Rechneruhr pro Sekunde von der physikalischen Zeit abweichen kann.

  Ist die maximale Zeitdrift der verschiedenen Uhren bekannt und nach oben begrenzt durch τ, dann lassen sich zwei Ereignisse zur physikalischen Zeit (t_1, t_2) auf jedem beteiligten Prozessor unterscheiden, falls gilt:

  $$\forall t_1, t_2,\ \text{wobei}\ t_2 > t_1 \geq 0 : t_2 - t_1 > \sigma + \tau * \sigma\ [sec] \tag{7.6}$$

  In [59] wurden beispielsweise für τ bei Messungen Werte von maximal $200\mu s/s$ gemessen. Dort wird durch ein zusätzliches Protokoll zur Uhrensynchronisation diese Zeitdrift auf wenige Mikrosekunden reduziert. Hier soll jedoch auf zusätzlichen Overhead durch ein weiteres Protokoll verzichtet werden.

- Für eine leichtgewichtige Implementation ist zusätzlich zu beachten, daß häufige Timer-Ereignisse erheblichen "Overhead" produzieren können. Je weiter also zu unterscheidende Timer-Ereignisse auseinanderliegen, desto leichtgewichtiger wird das Protokoll. Dies spricht für ein großes k in Formel 7.7.

  Im normalen Betriebszustand ("WB") des Protokollautomaten müssen zwei aufeinanderfolgende "timeouth"-Zeitgeberereignisse unterschieden werden, so daß laut Formel 7.7 gelten muß:

$$HB_{intervall} > k(\sigma + \tau * \sigma)\ [sec], k \geq 1 \tag{7.7}$$

- Da alle $HB_{intervall}$-Sekunden eine HB-Nachricht von einer Instanz an alle anderen Instanzen gesendet wird, muß die maximale Nachrichtenlaufzeit für derartige Nachrichten beachtet werden. Es macht keinen Sinn, zwei HB-Nachrichten so schnell aufeinanderfolgend zu senden, daß aufgrund der Varianz in den Zwischenankunftzeiten beim Empfänger nicht unterschieden werden kann, in welchem Intervall die Nachricht versendet wurde. In einer Messung der Nachrichtenlaufzeiten[7] in einem "Broadcast"-Netz (vgl. [59]) wurde für verschiedene Nachrichtenlängen ermittelt, daß 90% aller Nachrichten nach 10 Millisekunden den Empfänger erreichen. Dies gilt insbesondere auch noch für eine hohe Auslastung des Netzes, da in den 10 Millisekunden (ms) bereits Verzögerungen durch Kollisionen berücksichtigt sind. Typischerweise sind die Nachrichtenlaufzeiten bei vergleichbaren LAN-Szenarien noch deutlich geringer (< 1 ms). Eine großzügige Wahl der maximalen Nachrichtenlaufzeiten erhöht allerdings die Robustheit, da so auch untypische Verzögerungen in den Endsystemen oder Netzkomponenten[8] kompensiert werden. Es folgt somit:

$$HB_{intervall} > k(\sigma + \tau * \sigma)\ [sec] > 10\ \text{ms}, k \geq 1 \tag{7.8}$$

  Es fällt auf, daß die Abschätzung für die maximale Nachrichtenlaufzeit die Formel 7.8 deutlich dominiert. Damit ist eine untere Schranke für die Intervalldauer hergeleitet. Aufgrund der großzügigen Abschätzung von $HB_{intervall} > 10$ ms als "Ereignisabstand", kann auf einen zusätzlichen, verteilten Uhrendienst verzichtet werden. Durch die Auflösung dieser Abhängigkeit vom Uhrendienst (vgl. "Zeit" in Abbildung 6.5) wird das Gruppenmanagement-Protokoll noch leichtgewichtiger, als vergleichbare Protokolle, die einen Uhrendienst benötigen (beispielsweise [169]). Durch

[7] Die Messung betrachtet die Nachrichtenlaufzeiten zwischen lose gekoppelten Systemen. Das verwendete Übertragungsmedium war ein 10Base2-Ethernet. Die dort gemachten Beobachtungen sind auf die Systeme in Abbildung 4.8 übertragbar.

[8] So ist beispielsweise die Verzögerung einer Dateneinheit in einem Ethernet-LAN, das mit Switches ("multiport bridges") realisiert ist, u.a. davon abhängig, ob der Switch eine lokale Zwischenspeicherung der Dateneinheiten vornimmt ("store and forward") oder sofort die Weiterleitung nach Empfang der Empfängeradresse initiiert ("cut through"). Weitere Verzögerungen können sich durch Verklemmungen am Ausgang einer Switch-Schnittstelle ergeben, falls Dateneinheiten aus mehr als einem Segment quasigleichzeitig auf das selbe Segment weitergeleitet werden müssen ("output blocking").

diese pragmatische Abschätzung, die auf empirisch ermittelten Nachrichtenlaufzeiten in LAN-Umgebungen beruht, ist auch sichergestellt, daß es zu keiner Überschätzung der Leistungsfähigkeit des Protokolls kommt. Beispielsweise wird auf Grundlage einer a priori angenommenen, oberen Schranke für die Zeitabweichung der Uhren voneinander, im Zuge des Protokollentwurfs in [85], eine sehr optimistische Echtzeiterkennung eines Ausfalls im Bereich von 200 ms abgeleitet. Das Protokoll lieferte dementsprechend auch falsche "Prognosen" zur Verfügbarkeit einzelner Gruppenmitglieder.

- Für die Aufnahme in eine Gruppe muß jedes Gruppenmitglied dem Antrag des Bewerbers zustimmen. Dafür muß ausreichend Zeit vorgesehen werden, da nicht davon ausgegangen werden kann, daß jede Instanz sofort die "WantJoin"-Nachricht des Antragstellers empfängt. Auch diese Nachricht könnte verloren gehen. Der Bewerber muß also ggf. mehrfach seinen Antrag versenden und die periodisch eintreffenden "HB"-Nachrichten der Gruppenmitglieder daraufhin untersuchen, ob sie Bestätigungen für den Antrag beinhalten. Es kann jedoch auch passieren, daß eine Instanz während der Aufnahmephase ausfällt. In diesem Fall soll das potentielle Gruppenmitglied sich passiv verhalten, damit die verbleibenden Gruppenmitglieder zuerst wieder Konsens herstellen können (vgl. Diskussion zu Fluktuationen, Abschnitt 7.2.4.4). Für die Latenzzeit bis zur Wiederherstellung des Konsens nach einem Ausfall wurde $HB_{intervall} * (HB_{maxlost} + 3)$ $[sec]$ hergeleitet (s.o., Abschnitt 7.2.5.1). Innerhalb dieser Zeitspanne sollte es jeder Instanz möglich sein, dem Bewerber entweder eine Bestätigung zuzusenden, oder aber wieder Konsens nach einem Ausfall herzustellen und die neue Gruppenzusammensetzung mit einer HB-Nachricht zu propagieren. Innerhalb von $HB_{intervall} * (HB_{maxlost} + 3)$ $[sec]$ ist somit entweder der Neubewerber von allen Gruppenmitgliedern bestätigt, oder die Bewerbung wurde durch den Ausfall unterbrochen. Die Zeit (fortan "Abtastzeit"), die ein Bewerber für das "Einsammeln der Bestätigungen" (vgl. Schritte (2) und (4), Abschnitt 7.2.4.3) aufwenden muß, muß somit mindestens so groß sein wie $HB_{intervall} * (HB_{maxlost} + 3)$ $[sec]$.

$$\text{Abtastzeit} > HB_{intervall} * (HB_{maxlost} + 3)\ [sec] \tag{7.9}$$

  Sollte es einem Bewerber p nicht gelingen, während der Abtastzeit der Gruppe beizutreten, dann liegt entweder ein Dissens vor (Ausfall eines Gruppenmitglieds während der Bewerbung), oder der Bewerber ist selbst ausgefallen. Die Gruppenmitglieder können somit nach Verstreichen der Abtastzeit auch einem anderen Bewerber q eine Bestätigung zusenden, obwohl gilt: $p \lessdot q$ (p hat Priorität vor q), da p möglicherweise ausgefallen ist.

- Treten gleichzeitig mehrere Bewerber auf, sollen nicht priorisierte Bewerber in einen Wartezustand übergehen, so daß der am höchsten priorisierte Bewerber zuerst der Gruppe beitreten kann. Diese Wartezeit sollte größer als die Abtastzeit sein, da danach die priorisierte Instanz mit hoher Wahrscheinlichkeit von der Gruppe auf-

genommen wurde und nicht mehr mit den wartenden Instanzen um die Aufnahme konkurriert. Es ist also für die Wartezeit zu fordern:

$$\text{Wartezeit} > \text{Abtastzeit} > HB_{intervall} * (HB_{maxlost} + 3)\ [sec] \tag{7.10}$$

Die hergeleiteten Ungleichungen sind die Grundlage für ein vernünftiges Timing bei der Kommunikation zwischen den Protokollinstanzen. Im Zuge einer Implementation können zur Optimierung verschiedener weiterer Ziele die Freiheitsgrade bei der Wahl des genauen Timings ausgeschöpft werden, d.h. die Wahl von k, $HB_{intervall}$ und $HB_{maxlost}$.

7.2.5.3 "Protocol Data Unit"-Format

Das "Protocol Data Unit"-Format (PDU-Format) muß eine möglichst effiziente Umsetzung der oben besprochenen Mechanismen unterstützen. Es bietet sich an, Informationen mit fester Länge in einen Kopfteil zu plazieren. Darin enthalten sind Hinweise auf die nachfolgenden, variablen Teile der PDUs. Zur Unterstützung einer einfachen Interpretation der eintreffenden PDUs sollte das Format sehr einfach strukturiert sein. Beispielsweise kann man eine auf heutigen Prozessoren übliche Wortbreite (32-Bit) als PDU-Typ-Kennung und für andere Zählwerte verwenden. Dies erleichtert den Zugriff auf die Informationen im Vergleich zu Protokollen, die eine Platzoptimierung im Protokollkopf vornehmen. Beispielsweise wird beim "Internet Protocol" (IP, [190]) die Versionsangabe mit vier Bit kodiert. Beim "Transmission Control Protocol" (TCP, [191]) werden wichtige semantische Informationen für die Kennzeichnung von Verbindungsauf/abbau in einzelnen Bits kodiert. Dies erfordert auf modernen Prozessoren vergleichsweise aufwendige Bit-Manipulationen. Der Zugriff auf die Informationen innerhalb der PDU kann weiter vereinfacht werden, indem das PDU-Format auch bei unterschiedlichen PDU-Typen weitgehend gleich bleibt. Auf der Empfangsseite braucht dann bei Zugriff auf Typ-neutrale Informationen keine Fallunterscheidung in Abhängigkeit vom PDU-Typ vorgenommen werden.

Neben der Typ-Kennung ("WJ", "JM", "HB") muß eine Instanz mit der "HB"-Nachricht die lokale Gruppenliste mitführen. Es muß somit ein Zählwert für die Anzahl der nachfolgenden Informationen über Gruppenmitglieder folgen. Außerdem sollte die Instanz immer mitteilen, welche Position sie selbst ihrer Meinung nach in dieser Gruppe hat. Zur Erhöhung der Robustheit sollte jede Instanz eine monoton steigende Sequenznummer in jede PDU einfügen. Die anderen Instanzen können so zufällige Paketverdoppelungen von tatsächlich neuen Informationen unterscheiden. Dies ist wichtig, da der Entscheidungsalgorithmus darauf beruht, daß eine Instanz lokal prüfen kann, wann sie zuletzt *neue* Informationen von einer anderen Instanz empfangen hat.

Für jede Instanz, die in der lokalen Gruppenliste geführt wird, sollte in der HB-Nachricht der Name der Instanz (pid) und die zuletzt von dieser Instanz empfangene Sequenznummer (seqnr) eingetragen werden. Hiermit kann eine Instanz entscheiden, ob andere Instanzen die gleiche Sicht auf die Prozessorgruppe haben. Fällt eine Instanz aus, gibt es bereits vor Verstreichen von $(HB_{maxlost} + 1) * HB_{intervall}$-Sekunden deutliche Hinweise auf dieses

Problem, da für diese Instanz die Sequenznummern in den versendeten Gruppenlisten der anderen Instanzen konstant bleiben.

Da die Anzahl der Instanzen variabel ist, muß im PDU-Kopf ein weiterer Zählwert für diese Anzahl (hbtotal) eingefügt werden. Weiterhin muß eine optionale Kennung in die "HB"-Nachricht eingefügt werden können, falls eine Instanz die "WJ"-Nachricht eines Bewerbers bestätigen möchte. Hierfür kann wiederum der Name des Bewerbers herangezogen werden und im PDU-Kopf wird eine entsprechende Markierung gesetzt (hbothers), die das Vorhandensein dieser Bestätigung hinter der Gruppenliste anzeigt.

Das PDU-Format ist in Abbildung 7.4 dargestellt. Die einzelnen Felder haben eine Größe von 32-Bit oder 64-Bit. Der PDU-Typ wird im ersten Feld "hbtype" angegeben und kennzeichnet die PDU als "WJ", "JM" oder "HB". Im Feld "hbtotal" wird bei HB-Nachrichten die Anzahl der Gruppenmitglieder aus Sicht des Senders angegeben. "hbmyself" gibt die Position des Senders in der angefügten Gruppenliste an. "hbothers" gibt zusätzliche Einträge am Ende der Gruppenliste an (s.u.). Im 64-Bit breiten Feld "hbseqnr" trägt die Instanz eine monoton steigende Sequenznummer ein. Geht man von einer $HB_{intervall}$-Länge von einer Millisekunde aus, dann erlaubt diese Feldgröße eine ununterbrochene Laufzeit des Protokolls von etwa 585 Millionen Jahren. Angesichts dieser hohen potentiellen Laufzeit erscheint die Feldgröße überdimensioniert. Eine Feldgröße von 32-Bit würde jedoch bei gleicher $HB_{intervall}$-Länge nur eine Laufzeit von ca. 50 Tagen ermöglichen. Dieser Wert wäre deutlich zu gering. Ein Zwischenwert von beispielsweise 48-Bit bringt keinen Gewinn, da ein Zugriff auf Nutzinformationen in den verbleibenden 16-Bit wiederum aufwendige Bit-Manipulationen erfordern würde.

7.2.5.4 SDL-Spezifikation

Die Spezifikation mit Hilfe von endlichen Automaten ist relativ grob. Es sind zwar die wesentlichen Zustände und Zustandsübergänge dargestellt, eine hinreichende Hilfe für eine Implementierung ist dies jedoch noch nicht. Außerdem ist die Beschreibung der oben dargestellten Abläufe noch sehr ungenau, das genaue Verhalten einer Instanz ist noch nicht erkennbar.

Eine wesentlich feinere Spezifikation erlaubt die Modellierung des Protokolls mit Hilfe von SDL. Die SDL-Spezifikation des Protokolls (vgl. Anhang B) ist bereits so detailliert, daß ein hieraus automatisch generiertes Modell die Simulation des HB-Protokolls (für ein oder mehrere Instanzen) und des erforderlichen "Broadcast"-Mediums erlaubt. Somit ist nicht nur eine genauere Modellierung (beispielsweise die Verwaltung der Gruppenliste auf den einzelnen Instanzen) möglich, sondern eine Validierung der Mechanismen am Modell.

An dieser Stelle würde eine genaue Diskussion der einzelnen Module zu weit gehen. Dies wird im Anhang nachgeholt. Zwei Aspekte sind jedoch bermerkenswert: Erstens ist es notwendig, ein "Broadcast"-Medium als eigenständigen Prozeß zu spezifizieren, da SDL keine vordefinierten Kanäle für Punkt-zu-Mehrpunkt-Kommunikation bzw. für Gruppenkommunikation bereitstellt. Offensichtlich ist dieses Werkzeug primär für die Modellierung

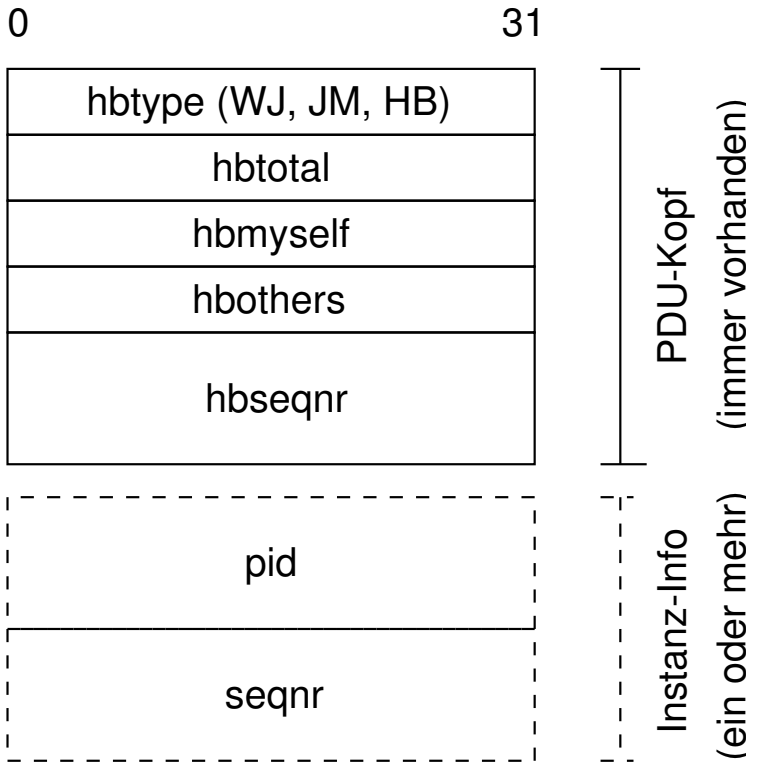


Abbildung 7.4: Syntax der GMP-PDUs

bzw. Spezifikation von "Peer-to-Peer"-Kommunikation ausgelegt und zwingt somit den Entwickler zu einigen zusätzlichen Anstrengungen. Aus der Sicht des Sicherheitsmanagements ist dies ein Nachteil, da zu befürchten ist, daß eher auf formale Methoden verzichtet wird, wenn sie nicht die nötige Ausdruckskraft für eine intuitive Anwendung bereitstellen.

Andererseits ermöglicht die Spezifikation des "Broadcast"-Mediums als eigenständigen Prozeß auch die Simulation von Paketverlusten und anderen Netzfehlern. Dies ist wiederum aus Sicht des Fehlermanagements sinnvoll, da validiert werden kann, ob die simulierten Fehler vom Protokoll kompensiert werden können.

Zweitens mußte bereits bei der Spezifikation eine Implementierung der $\lessdot$-Relation angegeben werden, damit das Modell tatsächlich für eine Simulation herangezogen werden kann. Am einfachsten wäre die Verwendung der eindeutigen Prozeßidentifikatoren vom SDL-Datentyp "PID". Diese sind garantiert verschieden für zwei verschiedene Instanzen und wären somit als Prozessorname geeignet (vgl. Definition 7.1). Leider unterstützt SDL für diesen Datentyp nur den Identitätsoperator ("=") und nicht die Vergleiche für "kleiner" oder "größer". Auch in diesem Fall mußte über einen Umweg eine eindeutige und vergleichbare Namensvergabe erreicht werden. Der "Admin"-Prozeß übernimmt hierfür die Erschaffung der HB-Protokollinstanzen und übergibt ihnen als Namen eine monoton steigende Zahl. Dadurch ist eine eindeutige und vergleichbare Identifizierung der "Prozessoren"

gegeben. Diese Festlegung ist jedoch nur für die Simulation erforderlich. Eine Implementierung muß einen anderen, verteilten Mechanismus benutzen, der ohne den "Admin"-Prozeß auskommt. Der "Admin"-Prozeß ist somit nicht als Teil der Spezifikation, sondern nur als Teil der Simulation anzusehen.

In folgenden sind zwei Abläufe dargestellt, die bei Simulationsläufen mit dem Modell aufgezeichnet wurden. Die MSCs zeigen einerseits das Verhalten einer einzigen Instanz, die alleine eine neue Gruppe gründet (vgl. Abbildung 7.5).

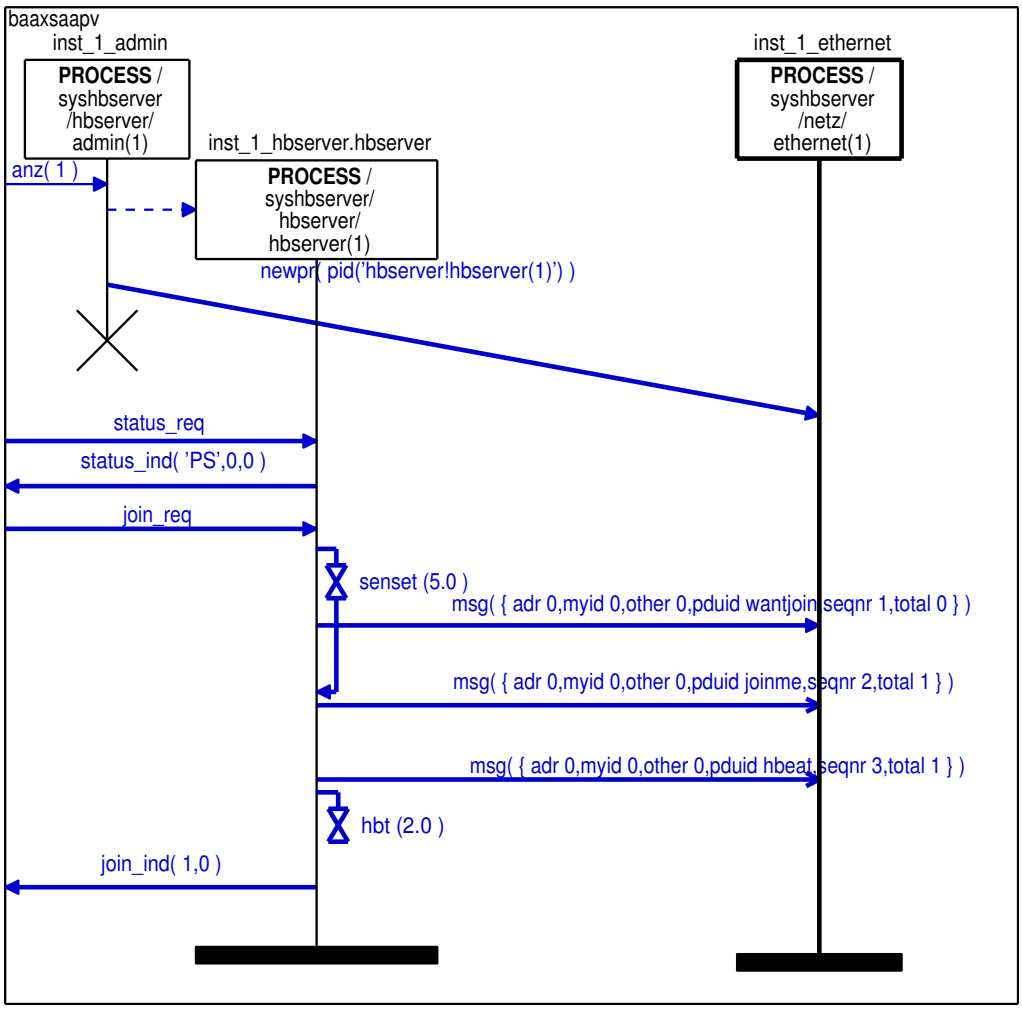


Abbildung 7.5: Ablauf I: Einzelne Instanz gründet neue Gruppe

Die zweite Abbildung zeigt zwei Instanzen, die gleichzeitig versuchen, in eine (noch nicht existente) Gruppe aufgenommen zu werden. Die Instanz "inst_1_" hat höhere Priorität und zwingt die zweite Instanz in den Zustand "BACKOFF". Nach Verstreichen der "Backoff"-Zeit wird die zweite Instanz in die inzwischen von der ersten Instanz gebildete Gruppe aufgenommen. Man beachte, daß die erste Instanz ihren Benutzer hierüber mit einer

"STATUS_IND"-Nachricht informiert. Die Abbildung ist aufgrund der Größe zweigeteilt (vgl. Abbildung 7.6 und 7.7).

Das SDL-Modell zeigt, daß die grundlegenden Mechanismen des Protokolls funktionieren: Eine Gruppe wird gebildet und Zu- bzw. Abgänge werden wie erwartet erkannt und behandelt. Die Gruppe verwaltet sich dezentral. Das SDL-Modell simuliert jedoch noch nicht alle Robustheitsmechanismen, beispielsweise wird keine Gruppenliste in den "HB"-Nachrichten versendet.

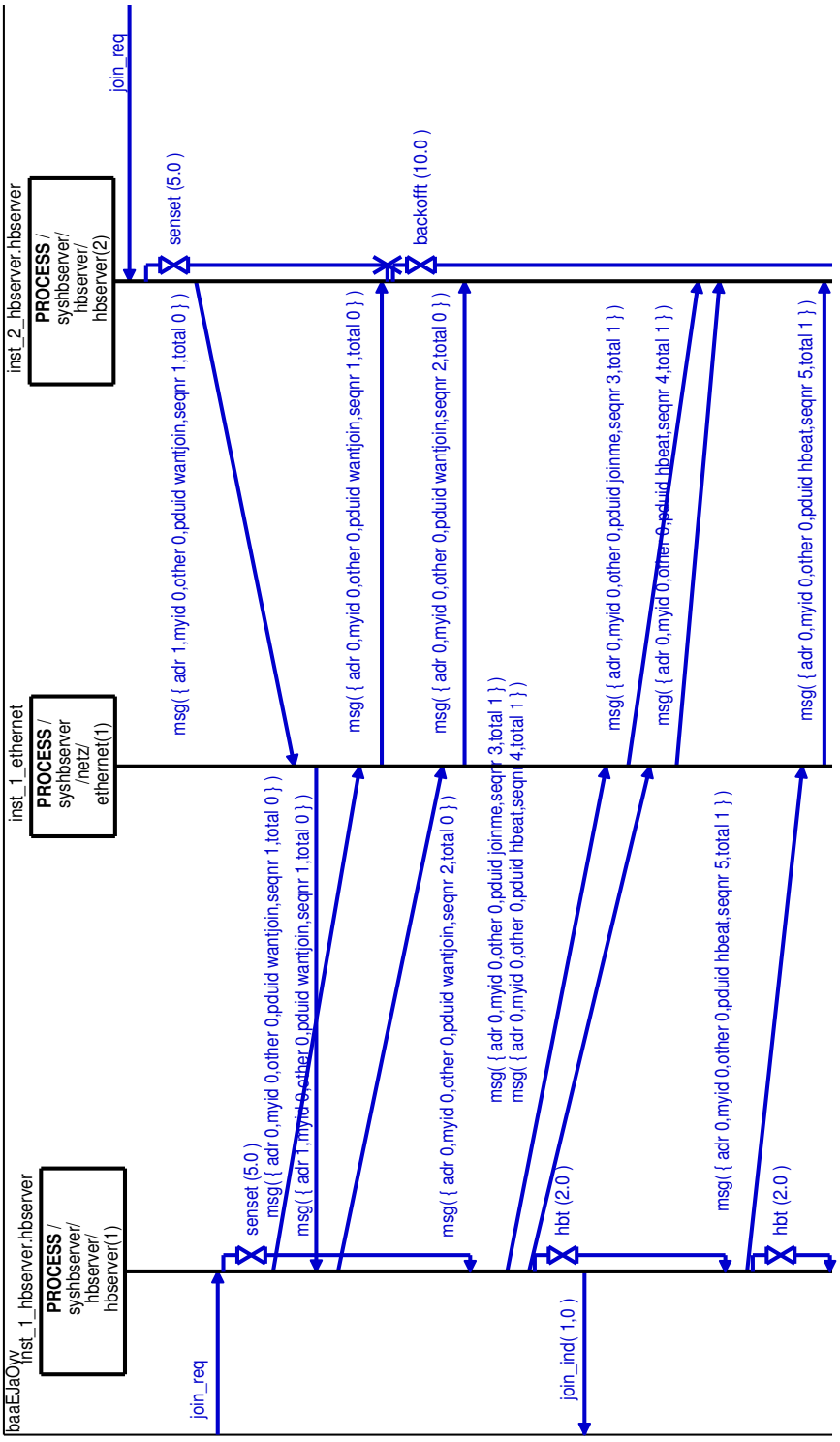


Abbildung 7.6: Ablauf II: Zwei Instanzen, zweite Instanz wird in Gruppe aufgenommen (1/2)

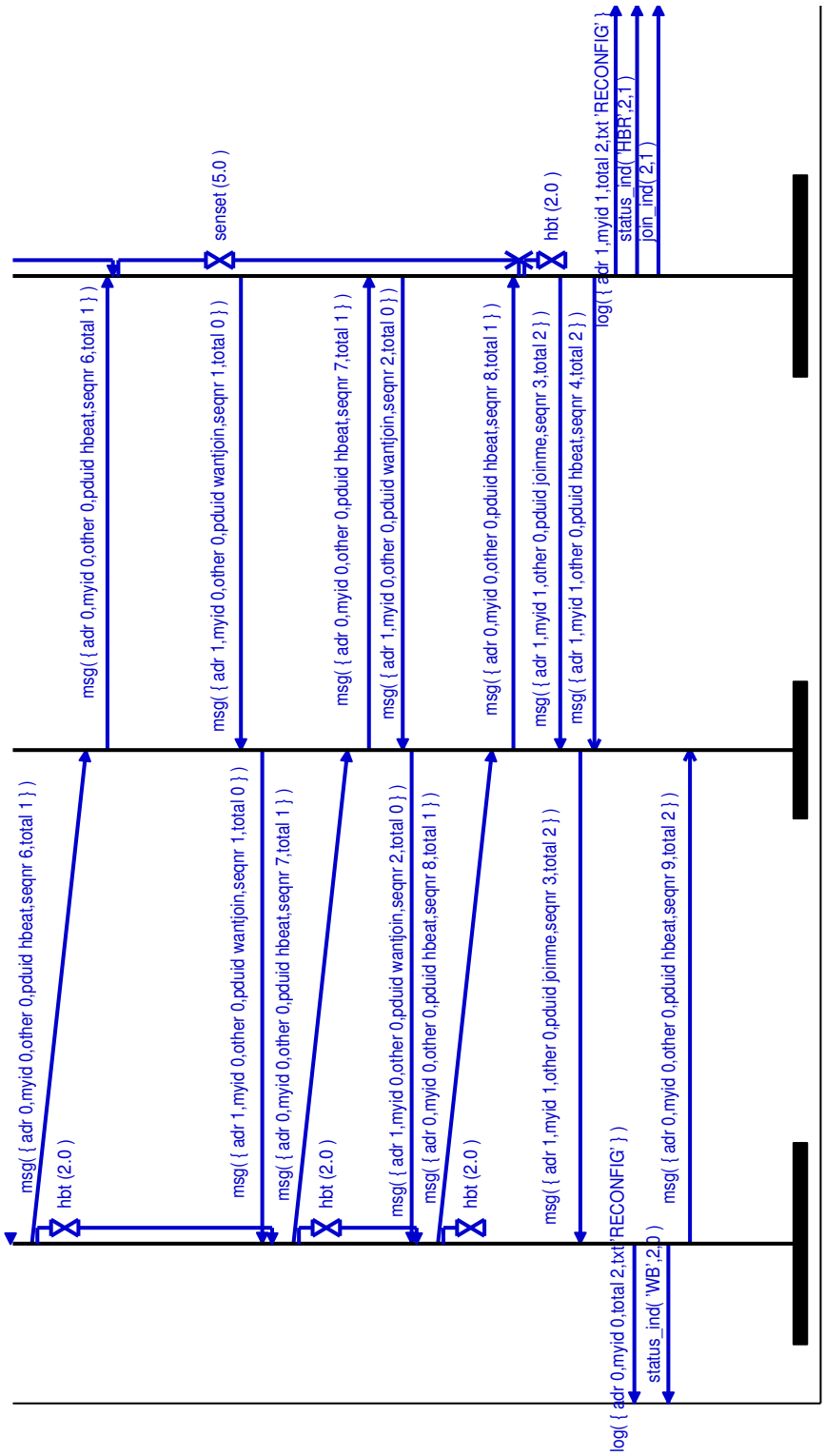


Abbildung 7.7: Ablauf II: Zwei Instanzen, zweite Instanz wird in Gruppe aufgenommen (2/2)

7.3 Validation vs. Sicherheitsanalyse

Eine formale Protokollspezifikation ist nicht nur Grundlage für eine möglichst exakte Implementation, sondern bietet darüber hinaus die Möglichkeit, manuell oder automatisch mit entsprechenden Tools eine Validierung des Protokolls durchzuführen.

In diesem Kapitel wird verdeutlicht werden, daß neben den "klassischen" Validationszielen (s.u.) noch weitere, für die Überlebensfähigkeit essentielle Validierungsschritte erforderlich sind. Es muß nämlich zusätzlich analysiert werden, ob das Protokoll nicht nur robust gegen technische Störungen ist, sondern insbesondere auch bei gezielten Angriffen noch seinen Dienst erbringt. Diese zusätzliche Überprüfung der Protokollspezifikation ist eine Sicherheitsanalyse. Diese Sicherheitsanalyse ist für überlebensfähige Systeme extrem wichtig, daher wird ihr ein eigenes Kapitel gewidmet (vgl. Kapitel 8). Die nun folgende "klassische" Validation wird sehr knapp gehalten, da es sich hierbei um bereits bekannte, etablierte Techniken handelt.

7.3.1 Validation des leichtgewichtigen Gruppenmanagement-Protokolls

Ziel der "klassischen" Validation einer Protokollspezifikation ist es, die logische Plausibilität dieser Spezifikation zu überprüfen. Die Validation soll zeigen, daß die Spezifikation des Protokolls festzulegenden Korrektheitskriterien genügt.

Die Validation bezieht sich somit immer auf die Spezifikation und nicht auf eine Implementation. Ist eine Validation erfolgreich, d.h. es wurde gezeigt, daß die Spezifikation den Anforderungen genügt, dann ist jede spezifikationskonforme Implementation geeignet, die Dienste zu erbringen. Die Konformität mit der Spezifikation muß selbstverständlich auch gezeigt werden. Dies erfolgt für jede Implementation im Zuge der Protokollverifikation, die für den Prototyp auszugsweise in Abschnitt 9 durchgeführt wird.

Typische Korrektheitskriterien für Kommunikationsprotokolle sind *Vollständigkeit* und *Konsistenz*. Eine Protokollspezifikation ist unvollständig, falls es vorkommen kann, daß in einem oder mehreren Zuständen unerwartete Nachrichten eintreffen. Eine Protokollspezifikation ist demnach vollständig, wenn in jedem Zustand auf alle möglichen eintreffenden Nachrichten reagiert werden kann. Ein Protokoll ist darüber hinaus konsistent, wenn es nachweislich frei von *Verklemmungen* ("Deadlocks"), unproduktiven Verhalten ("Livelocks") und unerwarteten Abbrüchen ("improper termination") ist (vgl. [120]).

7.3.2 Vollständigkeit

Die Vollständigkeit einer Protokollspezifikation hat zwei Aspekte. Zum einem muß validiert werden, ob das Protokoll "den Erwartungen an die Funktionalität" entspricht (Semantische Vollständigkeit). Auf der anderen Seite validiert man, ob das Protokoll die möglichen Eingaben (Nachrichten) verarbeiten kann (Syntaktische Vollständigkeit).

7.3.2.1 Syntaktische Vollständigkeit

Die Vollständigkeit eines Protokolls erfordert, daß in jedem Zustand auf alle möglicherweise eintreffenden Nachrichten reagiert werden kann. Das Protokoll muß so spezifiziert werden, daß es zu keinem "unerwarteten Empfang" einer Nachricht kommen kann.

Die Vollständigkeit wird in der vorliegenden Spezifikation dadurch erreicht, daß Nachrichten, die in einem Zustand eintreffen und nicht verarbeitet werden können, entweder für eine spätere Verarbeitung aufbewahrt werden (beispielsweise im Zustand "CC" durch das SDL-Save-Konstrukt) oder explizit vernichtet (konsumiert) werden, ohne eine beobachtbare Aktion nach sich zu ziehen (vgl. Diskussion in Abschnitt 117).

7.3.2.2 Semantische Vollständigkeit

Das Protokoll entspricht den Erwartungen, wenn die im Zuge der Dienstspezifikation definierten Dienstprimitiven vom Protokoll verarbeitet werden. In der vorliegenden Spezifikation wird die Vollständigkeit dadurch erreicht, daß für alle Dienstprimitiven, die der Benutzer an das Protokoll senden kann ("XYZ_REQ"), entsprechende Prozeduren im Protokoll vorgesehen sind. Kann eine Dienstprimitive nicht unmittelbar (innerhalb desselben Zustandes) verarbeitet werden, wird dies vom Protokoll vermerkt, um nach der Diensterbringung mit einer Antwort den Benutzer zu informieren. Die hierfür eingesetzten Mechanismen werden im Detail in Abschnitt 117 diskutiert.

7.3.3 Keine Verklemmungen ("Deadlocks")

Eine Verklemmung liegt vor, wenn der Protokollautomat in einen Zustand gelangt, in dem keine weitere Protokollverarbeitung mehr möglich (obwohl erforderlich) ist. Das vorliegende Protokoll ist verklemmungsfrei:

Der Startzustand wird zwingend verlassen durch einen Übergang in den Zustand "PS". In diesem Zustand ist keine Protokollverarbeitung erforderlich, der Protokollautomat reagiert lediglich auf Benutzerinteraktionen. Aus der Spezifikation ist ersichtlich, daß mit Ausnahme des Startzustandes und des Zustands "PS" in allen Zuständen durch das Setzen von Zeitgebern eine Verklemmung verhindert wird. In der SDL-Spezifikation sind die drei Zeitgeber zum Generieren der internen Signale ("timeoutb", "timeouts" und "timeouth" in Abbildung 7.3) benannt: "BACKOFFT", "SENSET" und "HBT". Treten keine erforderlichen externen Ereignisse auf, dann tritt schließlich immer ein Zeitgeberereignis ein, das einen Zustandsübergang herleitet. Der Zustand "BO" kann immer wieder durch Ablaufen des "BACKOFFT"-Timers verlassen werden. Die Zustände "WJS" und "HBR" können immer durch den Ablauf des "SENSET"-Timers verlassen werden. Der Zustand "WB" wird periodisch durch den Zeitgeber "HBT" verlassen. Der Zustand "CC" konsumiert keine Signale, er wird durchlaufen, und es erfolgt zwingend ein Übergang in den Zustand "WB".

7.3.4 Keine "Livelocks"

Ein "Livelock" liegt vor, wenn der Protokollautomat ständig die gleiche Abfolge von Zuständen durchläuft, ohne wirklichen Fortschritt zu erzielen. In diesem Fall ist es erforderlich zu definieren, was "Fortschritt erzielen" bedeutet. Typischerweise wird dies durch Kennzeichnen bestimmter Zustände als *akzeptierende* Zustände erreicht. Das Protokoll ist frei von "Livelocks", wenn nachweislich keine unendlichen Zustandsfolgen generiert werden, die keinen akzeptierenden Zustand enthalten.

Im vorliegenden Protokoll sind die Zustände "PS" und "WB" akzeptierende Zustände.[9] Das Protokoll verharrt initial und nach einem "LEAVE_Req" im Zustand "PS". Nach einem "JOIN_Req" strebt der Protokollautomat hingegen den Zustand "WB" an. Eine endliche Anzahl von Partnerinstanzen vorausgesetzt, werden diese Zustände schließlich auch immer erreicht. Auch in diesem Fall sind es die Zeitgeberereignisse, die den Automaten schließlich in den Zustand "WB" zwingen. Der Zustand "PS" kann durch Benutzeranforderungen (durch ein "LEAVE_Req", ggf. mit einem Umweg durch den Zustand "BO") immer wieder erreicht werden.

7.3.5 Kein ungewollter Abbruch

Es ist in der Spezifikation keine Termination der Protokollautomaten vorgesehen. Es kann daher auch nicht zu einem ungewollten Abbruch des Protokollautomaten kommen, da kein (übersehenes) Ereignis eine Termination herbeiführt.

7.4 Erreichte Zuverlässigkeit

Die Spezifikation des eben vorgestellten Protokolls soll der Erhöhung der Zuverlässigkeit dienen, da in Abschnitt 6.1 gezeigt wurde, daß durch die Parallelisierung zur Erhöhung der Leistungsfähigkeit die Zuverlässigkeit sinkt. Es wurde dargestellt, daß durch den Ausfall einer Instanz das Gesamtsystem nicht mehr richtig funktioniert, da es zu systematischen Paketverlusten kommt. Das System wurde daher als "n aus n"-System klassifiziert, bei dem alle n Instanzen verfügbar sein müssen (vgl. die Formeln 6.3 – 6.6).

In Abschnitt 7.4.1 wird nachgewiesen, daß durch das GMP die Verfügbarkeit des Gesamtsystems deutlich zunimmt und eine beliebig hohe Verfügbarkeit erreicht werden kann. In Abschnitt 7.4.2 wird gezeigt, wie diese Erkenntnisse für die Dimensionierung einer parallelen Packet Screen verwendet werden könnten.

[9] In Abbildung 7.3 ist dies, wie in der Automatentheorie üblich, durch den doppelten Rand des Zustandes angedeutet. Modellierungsvorgaben von SDL zwingen zu der Unterscheidung von "S" und "PS". Im Automatenmodell entsprechen diese beiden Zustände dem akzeptierenden Startzustand.

7.4.1 Verfügbarkeit der parallelen Packet Screen

Durch den Einsatz des eben vorgestellten, leichtgewichtigen GMP-Protokolls konfigurieren sich die Komponenten neu, so daß das System mit verminderter Leistung weiterarbeiten kann. Der Ausfall einer oder mehrerer Instanzen führt nicht mehr zu systematischen Paketverlusten, da nun jede Instanz als aktive Reserve für jede andere Instanz fungiert. Ein solches System wird auch als "1 aus n"-System bezeichnet, da es intakt bleibt, so lange mindestens eine der n Instanzen verfügbar ist. Dadurch ist die Verfügbarkeit des Systems deutlich erhöht, denn nun müssen alle n Instanzen gleichzeitig ausfallen, bevor es zu systematischen Paketverlusten kommt.

Analog zu der in Abschnitt 6.1.2 eingeführten Notation gilt (U = "Unverfügbarkeit"):

$$U_{1\ of\ n}(n,t) = P\left(\bigcap_{i=1}^{n} b_i(t)\right) \tag{7.11}$$

Damit das Gesamtsystem nicht mehr verfügbar ist, müssen zum Zeitpunkt t alle n Instanzen versagt haben. Und es gilt aufgrund der Unabhängigkeit der Ausfallwahrscheinlichkeiten auf den Instanzen:

$$U_{1\ of\ n}(n,t) = \prod_{i=1}^{n} P(b_i(t)) \tag{7.12}$$

Wählt man wie im Beispiel aus Abschnitt 6.1.2 $P(a_i(t)) = 0,95$, dann folgt $P(b_i(t)) = 1 - P(a_i(t)) = 0,05$. Und man erhält durch Einsetzen in Formel (7.12) die Ausfallwahrscheinlichkeit für das Gesamtsystem zu:

$$U_{1\ of\ n}(n,t) = 0,05^n \tag{7.13}$$

Die Wahrscheinlichkeit für einen Totalausfall nimmt erwahrtungsgemäß mit jeder weiteren Instanz ab bzw. die Verfügbarkeit nimmt mit jeder Instanz wegen $A_{1\ of\ n}(n,t) = 1 - U_{1\ of\ n}(n,t)$ zu. Die Verfügbarkeit kann somit beliebig durch Zufügen weiterer Instanzen erhöht werden.

7.4.2 Verfügbarkeit bei Leistungsvorgaben

Aufgrund des analytischen Modells aus Abschnitt 4.6 ist es nun auch möglich, die Verfügbarkeit in Abhängigkeit von Leistungsvorgaben zu optimieren.

Beispiel: Wieviele Instanzen werden benötigt, damit bei 50 Filterregeln ein Durchsatz von mindestens 20.000 Paketen pro Sekunde über einen Zeitraum von t Tagen mit 95% Wahrscheinlichkeit garantiert werden kann, falls keine Reparatur ausgefallener Komponenten vorgenommen werden kann?

Nimmt man beispielsweise an, daß zu den in Kapitel 4 beschriebenen Instanzen gleichwertige Komponenten verwendet werden sollen, deren individuelle Verfügbarkeit nach t Tagen bei $P(a(t)) = 0,85$ liegt, ergibt sich folgende Vorgehensweise für die Berechnung der benötigten Instanzen:

1. Die geforderte Leistung von 20.000 DG/s wird in die nach n aufgelöste Formel (4.22) eingesetzt, um die Anzahl der Instanzen zu berechnen, die für den geforderten Paketdurchsatz erforderlich sind:

$$Pmax_n(n, FR) = \frac{n}{nB_{sel} + B_{fil*}}, \quad B_{fil*} = B_{fil} + c * FR \tag{7.14}$$

   wird nach n aufgelöst zu

$$n = \frac{Pmax_n(n, FR) * B_{fil*}}{1 - Pmax_n(n, FR) * B_{sel}} \tag{7.15}$$

   Durch Einsetzen der Werte für $Pmax_n(n, 50) = 20.000\ DG/s$, sowie der in Abschnitt 4.6.3 errechneten Werte für B_{fil}, c und B_{sel} erhält man als Ergebnis für obige Gleichung, daß $n \approx 3,098$ ist. Man benötigt somit $\lceil n \rceil = 4$ Instanzen, um bei 50 Filterregeln den Mindestdurchsatz von 20.000 DG/s im ausfallfreien Fall nicht zu unterschreiten.

2. Die erforderliche Redundanz muß so berechnet werden, daß diese vier Instanzen nach t Tagen mit einer (Überlebens-)Wahrscheinlichkeit von 95% noch aktiv sind. Da leider gilt $P(a(t)) = 0,85$ müssen offensichtlich weitere Instanzen zur Erhöhung der Redundanz hinzugefügt werden. Es wird also ein "4 aus n"-System gesucht, für das gilt: $A_{4\ of\ n}(n, t) \geq 0,95$. Hilfreich ist bei dieser Suche die Formel für die Unverfügbarkeit von parallelen "m aus n"-Komponenten (vgl. [204]):

$$U_{m\ of\ n}(t) = \sum_{k=n-m+1}^{n} \binom{n}{k} P\Big(b(t)\Big)^k P\Big(a(t)\Big)^{n-k} \tag{7.16}$$

   Diese Formel gibt die Wahrscheinlichkeit dafür an, daß *mindestens eine mehr als die geforderte Anzahl der Komponenten* $(n - m + 1)$ ausfällt. Gesucht ist demnach ein n, so daß gilt:

$$A_{4\ of\ n}(n, t) = 1 - U_{4\ of\ n} \geq 0,95 \tag{7.17}$$

3. Das Berechnen der Verfügbarkeit in Abhängigkeit von n ergibt für $n = 5$:

$$\begin{aligned} A_{4\ of\ 5}(5, t) &= 1 - \left(\binom{5}{2} 0,15^2 * 0,85^3 + \binom{5}{3} 0,15^3 * 0,85^2 + \binom{5}{4} 0,15^4 * 0,85 + 0,15^5 \right) \\ &\approx 0,835 \end{aligned} \tag{7.18}$$

Die Verfügbarkeit bei $n = 5$ ist somit nicht ausreichend, da kleiner als $0,95$. Hingegen ergibt die Berechnung der Verfügbarkeit für $n = 6$:

$$A_{4\ of\ 6}(6,t) = 1 - \left(\binom{6}{3} 0,15^3 * 0,85^3 + \binom{6}{4} 0,15^4 * 0,85^2 + \binom{6}{5} 0,15^5 * 0,85 + 0,15^6 \right) \approx 0,953 \tag{7.19}$$

Es werden demnach aufgrund der Anforderungen an die Verfügbarkeit mindestens 6 Instanzen benötigt, so daß nach t Tagen mit einer Wahrscheinlichkeit von über 95% noch vier Instanzen verfügbar sind. Damit ist, unter Verwendung von Formel (7.14), während der gesamten Laufzeit ein Durchsatz von mindestens $Pmax_n(4,50) \approx 24.250\ DG/s$ gewährleistet.

Durch die Parallelverarbeitung ist es nunmehr möglich, die Leistungsfähigkeit zu beschleunigen und gleichzeitig die Verfügbarkeit zu erhöhen.

7.5 Zusammenfassung

In diesem Kapitel wurde argumentiert, daß vorhandene GMPs ungeeignet sind, um gleichzeitig allen Anforderungen an überlebensfähige Systeme nachzukommen. Es gibt zwar sichere Gruppenmanagement-Protokolle, doch diese sind entweder viel zu aufwendig oder aber für ein ganz anderes Fehlermodell konzipiert. Will man ein leichtgewichtiges Gruppenmanagement erreichen, so sind die einzusetzenden Mechanismen genau an das Systemmodell und an das Fehlermodell anzupassen. Genau dies wurde beim Entwurf des leichtgewichtigen Gruppenmanagement-Protokolls getan. Die im Fehlermodell benannten Fehler (Instanzausfälle, Kabelbrüche und Ausfälle von Netzkomponenten) werden behandelt. Auf die Betrachtung anderer Fehler, beispielsweise die ungewollte Netzpartitionierung, kann verzichtet werden.

Leichtgewichtigkeit wird allerdings auch aktiv erreicht: Die konsequente Nutzung von "Multicast"-Nachrichten reduziert das erforderliche Nachrichtenaufkommen. Die Beschränkung auf wenige Instanzen und wenige Nachrichtentypen ermöglicht ein einfaches, leicht interpretierbares PDU-Format.

Die Nutzung von formalen Spezifikationstechniken ermöglichte schließlich die Simulation und Validation von wesentlichen Protokollmechanismen. Unter der Vielzahl in Frage kommender, formaler Spezifikationstechniken wurden zwei ausgewählt. Die Beschreibung mit einem Automatenmodell ermöglichte die Darstellung der Grobstruktur des Protokolls und die Diskussion der wesentlichen Mechanismen. Die im Anhang A dargestellte Protokollspezifikation mittels SDL verfeinerte diese Grobdarstellung bis zu einem Grad, der Simulation und Validation ermöglichte, und gleichzeitig durch die standardisierte, graphisch formale Darstellung die Voraussetzung für eine manuelle Analyse des spezifizierten Verhaltens schafft.

Diese exakte (da formale) Beschreibung zusammen mit der Eignung für eine manuelle Analyse (da graphisch nachvollziehbar) ist auch die Voraussetzung für die folgende Sicherheitsanalyse.

Durch den Einsatz dieses Protokolls kann die Verfügbarkeit der parallelen Packet Screen drastisch erhöht werden. Die Instanzen fungieren nunmehr als aktive Redundanz füreinander, so daß aus dem “n aus n”-System ein “1 aus n”-System entsteht, das eine deutlich höhere Verfügbarkeit aufweist, wie in Abschnitt 7.4 nachgewiesen wurde. Es wurde am Beispiel gezeigt, wie die Dimensionierung eines Systems berechnet werden kann, wenn gleichzeitig Anforderungen an Leistungsfähigkeit und Verfügbarkeit gestellt werden.

Kapitel 8

Sicherheitsanalyse für das leichtgewichtige GM-Protokoll

Die Kernfragen bei der Sicherheitsanalyse sind, welche protokollspezifischen Angriffe möglich sind und welche zusätzlichen Mechanismen zur Erkennung, Abmilderung oder Abwehr dieser Bedrohungen erforderlich sind. Es stellt sich die Frage, warum entsprechende Überlegungen nicht bereits im Zuge der Spezifikation unternommen werden. Hierfür gibt es mehrere Gründe:

- Das primäre Ziel der Spezifikation ist es, eine Schnittstelle (Dienstspezifikation) und ein Protokoll (Protokollspezifikation) zu beschreiben, die den funktionalen Anforderungen genügen, die an anderer Stelle formuliert wurden. Die funktionalen Anforderungen sind in Abschnitt 6.3.2 genannt; diese Anforderungen zielen in erster Linie darauf ab, daß eine Gruppe von Prozessen/Prozessoren so verwaltet wird, daß Ausfälle zeitnah erkannt werden, um zügig Konsens über die Gruppenmitgliedschaft herzustellen.

- Neben diesen funktionalen Anforderungen kann es weitere Anforderungen geben. In diesem Fall handelt es sich primär um Anforderungen an eine hohe Leistungsfähigkeit und Anforderungen an die Sicherheit, die sich aus den generischen Anforderungen an überlebensfähige Systeme ergeben. Anforderungen an die Leistungsfähigkeit können bei der Spezifikation bereits beachtet werden; das obige Protokoll stellt beispielsweise nur einen schwachen Konsens her, weil dies algorithmisch erheblich einfacher und somit effizienter ist.

- Erst nach der (ersten) Protokollspezifikation ist eine systematische Sicherheitsanalyse möglich, die nicht nur generische Schwachstellen aufzeigen kann, sondern darüber hinaus *protokollspezifische Aussagen* ermöglicht. Dies ist ein wesentlicher Aspekt in Hinblick auf die Überlebensfähigkeit, da sich in der Vergangenheit gezeigt hat, daß viele Sicherheitsprobleme in verteilten Systemen gerade durch Ausnutzen von protokollspezifischen Schwachstellen entstehen (vgl. beispielsweise [103, 107, 207, 247,

248]). Anders formuliert: erst durch die (vorläufige) Festlegung des Protokolls ist eine eingehende Sicherheitsanalyse der Protokollspezifikation möglich.

Im folgenden werden die Annahmen über die Umgebung ergänzt (vgl. Abschnitt 8.1), "generische" Sicherheitsanforderungen an Gruppenmanagement-Protokolle diskutiert (vgl. Abschnitt 8.2) und schließlich protokollspezifische Sicherheitsanalysen durchgeführt (vgl. Abschnitt 8.5). Dafür ist es erforderlich, zusätzliche Entwurfsschritte zu diskutieren, die beachtet werden müssen, wenn gleichzeitig Leistungs,- Verfügbarkeits- und Sicherheitsanforderungen an einen verteilten Dienst gestellt werden (vgl. Abschnitt 8.3). Vor der Sicherheitsanalyse am Protokoll müssen die hierfür gängigen Techniken motiviert werden (vgl. Abschnitt 8.4).

Das Ziel dieses Kapitels ist es, das leichtgewichtige GM-Protokoll in ein sicheres GM-Protokoll zu überführen, das auch in einer maliziösen Umgebung eingesetzt werden kann. Damit ist dieses Kapitel vergleichbar mit Abschnitt 6.2. Ein Fehlermodell führt zur Definition einer Fehlersemantik, die das Verhalten eines Systems bei Auftreten von Fehlern definiert. Die Diskussion der Angriffe (sozusagen das "Angriffsmodell") führt zur Spezifikation von zusätzlich erforderlichem Systemverhalten für den Fall eines Angriffs.

8.1 Sicherheitsrelevante Annahmen über die Umgebung

In Abschnitt 7.2.2 wurden *technische* Annahmen über die Umgebung (Environment) formuliert. Diese technischen Annahmen über die Umgebung ergeben sich aus anderen Protokollspezifikationen (beispielsweise für Ethernet IEEE 802.3) und sind somit typisch für Kommunikationsprotokolle, die aufgrund ihrer Anordnung in einer Schichtenstruktur in einem Dienstnehmer- (Schicht (n+1)) zu Diensterbringer- (Schicht (n)) -Verhältnis stehen.

Implizit wird bei diesen Annahmen über die Umgebung meist unterstellt, daß sich die Umgebung "gutmütig" verhält (vgl. [116]). Das Protokoll muß nur auf bestimmte, a priori bekannte Ereignisse aus der Umgebung reagieren, so als würde die Umgebung einer spezifizierten Schnittstelle genügen.

In einem überlebensfähigen System kann man solche Annahmen in der Regel jedoch nicht machen. Vielmehr muß davon ausgegangen werden, daß die Umgebung möglicherweise maliziös ist. Das wohldefinierte Verhalten der Umgebung gegenüber dem Protokollautomaten ist somit ein *erwünschtes Verhalten*, unter dem der Protokollautomat die funktionalen Anforderungen erfüllen kann (vgl. [1]). Zusätzlich muß jedoch mit unerwünschtem Verhalten (vgl. "adverse conditions" in Definition 2.2) gerechnet werden, das erkannt werden muß, um hierauf adäquat reagieren zu können. Die "klassische Protokollspezifikation", die oben durchgeführt wurde, spezifiziert also immer nur einen Teil des möglichen Verhaltens der Umgebung. Diese "Unterspezifikation" kann zu Sicherheitsproblemen führen (vgl. [117]).

Betrachtet man das Protokoll als Regelwerk bezüglich Syntax (PDU-Typen, PDU-Format), Semantik (Interpretation und Reaktion auf PDUs, Ausführen von Protokollfunktionen) und Timing (zeitliche Restriktionen bezüglich des Nachrichtenablaufs), dann muß man für überlebensfähige Systeme annehmen, daß die Umgebung *beliebig gegen Syntax-, Semantik- und Timing-Regeln verstoßen* kann. Vollständige Kontrolle hat ein Protokollautomat somit lediglich über die eigene Interpretation eintreffender PDUs.[1] Um so wichtiger ist es daher, Verstöße gegen Syntax und Timing zu erkennen, um mit zusätzlichen Regeln hierauf zu reagieren (vgl. "Resistance" und "Recognition" in Abschnitt 2.2.3).

Im folgenden wird von einer solchen "feindlichen" bzw. maliziösen Umgebung ausgegangen, die zumindest partiell gegen die Syntax-, Semantik- und Timing-Regeln des Protokolls verstößt.

Es müssen in Anlehnung an [68] die folgenden Fähigkeiten eines Angreifers ("active eavesdropper") betrachtet werden:

- Der Angreifer kann alle Nachrichten abhören (vgl. [42]).
- Als legitimer Benutzer des Netzes kann er aktiv an der Kommunikation teilnehmen, d.h. er kann zu jedem anderen Benutzer des Netzes Dateneinheiten senden und ggf. die Herkunft der Daten verschleiern (vgl. [43]).

Insbesondere hat der Angreifer die Möglichkeit, zuvor abgehörte Nachrichten zu einem beliebigen späteren Zeitpunkt erneut zu senden ("replay").

Die Fähigkeit, Dateneinheiten anderer Netzbenutzer unbemerkt zu vernichten, können jedoch ausgeschlossen werden. Es wird somit ein hinreichender physikalischer Schutz der Netzkomponenten und Instanzen aus Abbildung 4.8 vorausgesetzt, so daß es einem Angreifer nicht gelingen kann, die Kommunikation der anderen Benutzer durch Manipulieren von Übertragungsmedien oder aktiven Netzkomponenten zu unterbrechen. Darüber hinaus gilt weiterhin die Grundannahme, daß die Sicherheit jeder einzelnen Instanz hinreichend hoch ist. Ein Einbruch auf einer Instanz ("root compromise") und daraus folgende Möglichkeiten des Angreifers werden nicht betrachtet.

Es wird ebenfalls nicht der Fall betrachtet, daß eine Packet Screen durch einen "Denial of Service" (DoS)-Angriff außer Betrieb genommen werden kann (vgl. [71]). In diesem Fall hilft auch der Einsatz eines GMPs nicht, das aus dem Angriff resultierende Verfügbarkeitsproblem zu lösen, da der Angreifer nach einer Rekonfiguration die nächste Instanz angreifen kann. In diesem Abschnitt werden lediglich die zusätzlichen Risiken behandelt, die erst beim Einsatz des GMPs entstehen.

[1] Folgendes Zitat aus [1, S.8] spricht für sich: "The specification of a system Π asserts that Π guarantees a property M under the assumption that its environment satisfies some property E. [...] It is important to realize that E is an assumption about the environment, not a constraint placed on it. The environment cannot be constrained or controlled by the system."

8.2 Sicherheitsanforderungen an das Gruppenmanagement-Protokoll

Neben den funktionalen Anforderungen, die bereits in Abschnitt 6.3.2 formuliert wurden und zur obigen Spezifikation führten, müssen weitere (Hilfs-) Anforderungen an das GM-Protokoll gestellt werden. Hierbei handelt es sich um Sicherheitsanforderungen, deren Umsetzung eine ordentliche Diensterbringung auch unter "adverse conditions" ermöglichen sollen.

Es gibt eine "high level" Anforderung aus der Sicht des Sicherheitsmanagements, unter der sich die noch zu formulierenden Sicherheitsanforderungen subsumieren lassen:

Hauptsicherheitsanforderung an GM-Protokolle:
Die Integrität der Gruppe (Gruppenzusammensetzung) muß gewährleistet sein.

Diese Anforderung läßt sich folgendermaßen konkretisieren:

SA1: Nur autorisierte Prozessoren dürfen in die Gruppe aufgenommen werden.

SA2: Autorisierten Prozessoren darf die Aufnahme in die Gruppe nicht verweigert werden.

SA3: Die Integrität des Gruppenkonsens[2] muß gewährleistet sein.

Im allgemeinen Fall können weitere Sicherheitsanforderungen hinzukommen. Beispielsweise könnte man Vertraulichkeit bezüglich der Gruppenzusammensetzung fordern. Dies könnte bedeuten, daß es Prozessoren nicht möglich sein soll, Anzahl und/oder Identität der Gruppenmitglieder[3] zu erkennen. Die oben genannten Sicherheitsanforderungen sind nach Auffassung des Autors jedoch maßgeblich.

Zu beachten ist allerdings, daß diese Anforderungen in Konflikt mit anderen Anforderungen an ein überlebensfähiges System stehen können. Darüber hinaus sind die Anforderungen nicht gleichwertig zu gewichten. Es hängt vielmehr vom konkreten Einsatzgebiet des GMPs ab, welche Anforderungen Vorrang haben.

Im konkreten Fall der parallelen Packet Screen haben die Leistungsanforderungen einen hohen Stellenwert. Schließlich sind diese Leistungsanforderungen ausschlaggebend für die Parallelisierung gewesen. Die Parallelisierung erfordert wiederum eine Gruppenverwaltung. Diese Gruppenverwaltung darf die Leistungsgewinne durch die Parallelisierung nicht

[2] Dieser Punkt ist erklärungsbedürftig: Auch wenn nur autorisierte Prozessoren operational in der Gruppe vorhanden sind, kann es dennoch durch einen geschickten Angriff vorkommen, daß diese Prozessoren eine unterschiedliche oder auch falsche Sicht auf die Gruppenzusammensetzung haben.

[3] Es sind sogar Szenarien denkbar, in denen nicht einmal die einzelnen Gruppenmitglieder die Identität der anderen Gruppenmitglieder kennen dürfen. Eine etwas liberalere Form dieser Anforderung ist, daß die Identität der Gruppenmitglieder gegenüber nicht-Gruppenmitgliedern vertraulich sein soll.

zu stark beeinträchtigen. Daher wurde versucht, ein leichtgewichtiges Protokoll zu spezifizieren. Die im folgenden zu diskutierenden Angriffsszenarien auf dieses Protokoll erfordern schließlich neue Sicherungsmechanismen. Es ist daher offensichtlich, daß nur solche Mechanismen benutzt werden dürfen, die möglichst aufwandsarm während des Protokollablaufs sind. Dies wiederum bedeutet aber auch, daß die oben dargestellten Sicherheitsanforderungen nicht "um jeden Preis" durchgesetzt werden können. Auf diesen Aspekt wird unten noch genauer eingegangen.

8.3 Zusätzliche Entwurfsschritte für Protokolle in überlebensfähigen Systemen

In den folgenden Abschnitten werden Angriffsmöglichkeiten auf das HB-Protokoll dargestellt. Diese Angriffsmöglichkeiten stehen im Widerspruch zu den gerade formulierten Sicherheitsanforderungen. Um den Sicherheitsanforderungen dennoch zu genügen, müssen Sicherheitsmechanismen aus den Angriffsszenarien abgeleitet werden. Diese Sicherheitsmechanismen erhöhen die Komplexität des Protokolls. Die Sicherheitsanforderungen konkurrieren somit direkt mit den Leistungsanforderungen. Außerdem muß evaluiert werden, ob die zusätzlichen Sicherheitsmechanismen nicht selbst wieder für einen Angriff ausgenutzt werden können. Die Darstellung in den folgenden Abschnitten ist somit Ergebnis eines iterativen Prozesses:

1. Durchführen einer Sicherheitsanalyse mit dem Ziel, Sicherheitsprobleme aufzudecken.
2. Erarbeiten von protokollspezifischen Sicherheitsmechanismen zum Abwenden, Abmildern und Erkennen von Angriffen auf das Protokoll und den Dienst.
3. Abschätzen der Auswirkungen auf die Leistungsfähigkeit, ggf. Austauschen der aufwendigen Sicherheitsmechanismen gegen weniger aufwendige. Überprüfen, ob aufwendige Sicherheitsmechanismen aus dem Protokoll herausgenommen und in die Umgebung verlagert werden können.
4. Abschätzen der Auswirkungen auf die Verfügbarkeit, ggf. Austauschen zu restriktiver Sicherheitsmechanismen gegen weniger restriktive.
5. Wiederholen der Analyse in Schritt (1).

Dieser iterative Prozeß kann je nach Priorisierung der Anforderungen an verschiedenen Stellen abgebrochen werden. Er kann aus Sicht des Sicherheitsmanagements in Schritt (1) abgebrochen werden, falls keine Sicherheitsmängel mehr erkannt werden oder die verbleibenden Restrisiken als tragbar eingestuft werden. Aus Sicht des Leistungsmanagements muß in Schritt (3) abgebrochen werden, falls zu aufwendige Sicherheitsmechanismen den Einsatz des Protokolls gefährden.

Weiterhin muß die Sicht der Netzbenutzer beachtet werden. Zwar sind die Benutzer auch an Sicherheitsmechanismen interessiert bzw. zur Einhaltung bestimmter Sicherheitsrichtlinien verpflichtet, jedoch ist für sie die Verfügbarkeit des Netzes bzw. der Zugriff auf verteilte Betriebsmittel von primärer Bedeutung. Konkret bedeutet dies, daß in Schritt (4) solche Sicherheitsmechanismen ausgeschlossen werden müssen, die zu erheblichen Störungen der Arbeitsabläufe führen könnten. Beispielsweise darf bei Erkennen eines Angriffs nicht sofort sämtliche Kommunikation in das Internet am Firewall unterbrochen werden. Diese Reaktion wäre zu drastisch. Eventuell könnte die Kommunikation mit der angegriffenen Komponente unterbrochen werden. Bei der parallelen Packet Screen ist ein ähnlicher "Tradeoff" zwischen Sicherheit und Verfügbarkeit zu erkennen. Zur Risikominimierung wäre es bei Erkennen eines direkten Angriffs auf das einzusetzende GMP sinnvoll, die Ausführung des Protokolls zu beenden. Bei einem langanhaltenden Angriff würde während dieser Zeit allerdings die Verfügbarkeit des Netzes reduziert werden, falls es zu einem Ausfall einer Instanz kommt. Durch das Abschalten des GMPs käme es während des Angriffs bei Ausfall einer Instanz zu den in Abschnitt 6.1.3 diskutierten, systematischen Paketverlusten. Die aus Sicherheitgründen sinnvolle Abschaltung des GMPs bei Erkennen eines Angriffs ist somit zu restriktiv.

8.4 Sicherheitsanalyse für Protokolle

Bevor mit der eigentlichen Sicherheitsanalyse begonnen wird, werden die heute üblichen formalen Verfahren und damit einhergehende ungelöste Probleme kurz vorgestellt. Es handelt sich hierbei im wesentlichen um zwei Ansätze: Sicherheitsanalysen basierend auf Logiken und Automatenmodell-basierte Ansätze.

Die Durchführung einer Sicherheitsanalyse sollte nicht nach irgendeiner "ad hoc"-Methode erfolgen. Vielmehr muß eine Methode gefunden werden, die sowohl adäquat als auch verständlich ist.

Adäquat ist eine Methode nach Auffassung des Autors, wenn sie es ermöglicht, Aussagen über die Sicherungsziele des Protokolls abzuleiten. Für den allgemeinen Fall sind das Aussagen zur Integrität, Authentizität und Vertraulichkeit von Nachrichten, Kommunikationspartnern und Kommunikationsbeziehungen und Aussagen zur Verfügbarkeit (eines Dienstes), sowie Aussagen zu den hieraus ableitbaren Sicherungszielen (beispielsweise die Nicht-Abstreitbarkeit, die wesentlich auf der Authentizität beruht).

Schwieriger ist es, allgemein verständliche Sicherheitsanalysen zu formulieren. Auch hier hat die Wahl der Methode erheblichen Einfluß darauf, für wen die damit verfaßte Sicherheitsanalyse schließlich geeignet ist. Möglich ist hier die gesamte Bandbreite von informalen Beschreibungen bis hin zu formalen Deduktionen. Weitere wichtige Kriterien sind der Bekanntheitsgrad der Methode und die Verfügbarkeit eventuell notwendiger Analysewerkzeuge.

8.4.1 BAN-Logik und deren Weiterentwicklungen

Eine etablierte, formale Methode zur Beschreibung der Authentifikation[4] in verteilten Systemen ist die sog. "BAN-Logic", benannt nach den drei Autoren [41]. Die Vorgehensweise bei einer Protokollanalyse umfaßt die folgenden Schritte:

1. Überführen der informalen Protokollbeschreibung in eine idealisierte, formale Notation. Die BAN-Logik stellt dafür eine Reihe von Konstrukten zur Verfügung, die typische kryptographische Primitiven, wie beispielsweise symmetrische und asymmetrische Schlüssel, "frische Werte" (nonces[5]) und verschlüsselten Text, darstellen. Hiermit wird der Inhalt der übertragenen Nachrichten dargestellt.

2. Aufschreiben der Annahmen vor dem ersten Protokollschritt. Hierbei wird festgelegt, was die wesentlichen Randbedingungen für das Ausführen des Nachrichtenaustausches mit dem jeweiligen Protokoll ist. Beispielsweise kann die Existenz eines gemeinsamen Schlüssels oder die Existenz von "Nonces" (frische Werte, s.u.) postuliert werden.

3. Durchführen der Protokollanalyse. Dabei werden die Annahmen vor einem Protokollschritt in gültige Aussagen (assertions) nach diesem Protokollschritt transformiert. Diese Behauptungen werden durch die folgenden Protokollschritte erweitert und führen schließlich nach Ablauf des Protokolls zu einer Menge von Schlußfolgerungen. Die Schlußfolgerungen können dabei durch einen Satz von Axiomen transformiert werden. Beispielsweise gibt es ein Axiom, das bei Empfang einer verschlüsselten Nachricht und Kenntnis des Schlüssels die Kenntnis der unverschlüsselten Nachricht ableitet.

Am Ende eines Protokollablaufes können somit auf Grundlage der Schlußfolgerungen gesicherte Aussagen über das erlangte Wissen der Kommunikationspartner abgeleitet werden. Beispielsweise die Aussage "A weiß, daß nur B Nachricht X gesendet haben kann".

Die größte Kritik an der BAN-Logik und an ihrer wesentlichen Erweiterung (GNY-Logik, vgl. [99]) ist, daß im Zuge der Überführung des Protokolls in eine formale Beschreibung Interpretationsfehler, Streichungen und Ersetzungen vorgenommen werden, die nachweislich zu Fehlern führen können (vgl. [225, 33, 161]). Darüber hinaus wird kritisiert, daß

[4] Der Nachweis der Identität von Benutzern/Prozessen ist von fundamentaler Bedeutung, da hierdurch viele höherwertige Dienste erst ermöglicht werden. Beispiele für höherwertige Dienste, die eine Authentifikation erfordern sind: Verläßliche Abrechnung ("accounting") und verläßliche Autorisation ("credentials", Zuweisung von Rollen "rollenbasierte Zugriffskontrolle"). Viele kryptographische Protokolle benötigen eine vertrauenswürdige, dritte Instanz: einen Authentifikations-Dienst. Auch für diese Dienste stellt sich in verteilten Systemen schließlich die Frage, wie eine Beschleunigung/Lastverteilung herbeigeführt werden kann, was schließlich zur Frage der "sicheren Replikation von Authentifikations-Diensten" führt [98].

[5] Häufig werden Zufallszahlen verwendet, um eine neue Zahl zu generieren. Diese neue Zahl ist noch nicht im Zuge des Protokollablaufs verwendet worden und stellt somit einen "frischen Wert" dar.

diese Überführung ohnehin informal (ohne formale Semantik, eher intuitiv) ist und somit die ganze BAN-Logik eigentlich nicht als formale Beschreibung angesehen werden kann (vgl. [161, 225]). In [212] wird darüber hinaus gezeigt, daß die BAN-Logik nicht geeignet ist, die Korrektheit einer Authentifikation nachzuweisen. Eine weitere, wesentliche Kritik wird in [225] zusammengefaßt: Die BAN-Logik macht eigentlich keine Aussagen zur Sicherheit eines kryptographischen Protokolls[6], sondern lediglich zum Vertrauen der legitimen Kommunikationspartner in den Authentifikationsvorgang. Es folgte eine Reihe von Erweiterungen und Verbesserungsvorschlägen (beispielsweise [99, 108, 161, 160, 49]). Die Diskussion um die Eignung der BAN-Logik und derer Weiterentwicklungen zur formalen Beschreibung von sicheren Kommunikationsprotokollen hält weiter an.

8.4.1.1 Schlußregeln der BAN-Logik

Für die Sicherheitsanalyse des GMPs wird u.a. auf die BAN-Logik zurückgegriffen. Die folgende Zusammenfassung zeigt die wesentlichen Symbole der BAN-Logik, deren Semantik und die allgemeine Schlußregel der BAN-Logik anhand von Beispielen. Weitere Schlußregeln bzw. Axiome werden im Zuge der Sicherheitsanalyse bei Bedarf eingeführt.

$P \models X$**:** P glaubt X bzw. P hat gute Gründe, X zu glauben.

$P \lhd X$**:** P hat die Nachricht X empfangen. Jemand hat die Nachricht X oder die verschlüsselte Nachricht $\{X\}_K$ (s.u.) an P gesendet und P konnte die Nachricht empfangen und ggf. entschlüsseln.

$P \mid\sim X$**:** P hat in der Vergangenheit eine Nachricht X gesendet bzw. X ist Teil einer Nachricht, die P irgendwann einmal gesendet hat.

$\#(X)$**:** Der Ausdruck X ist bisher noch nicht im Zuge des Protokollablaufs verwendet worden. Die BAN-Logik bezeichnet dies als "freshness", jedoch sagt $\#(X)$ eigentlich nichts über das Alter von X aus, sondern lediglich, daß X im Zuge der bisherigen Protokollverarbeitung als "noch nicht verwendet" angesehen werden kann.

$P \Rightarrow X$**:** P hat Autorität über X und ihm kann daher (uneingeschränkt) geglaubt werden, wenn er X äußert.

$P \overset{K}{\longleftrightarrow} Q$**:** Der symmetrische Schlüssel K ist zwischen P und Q für die vertrauliche Nachrichtenübertragung geeignet. P und Q werden diesen Schlüssel nicht mißbrauchen oder kompromittieren.

$\{X\}_K$**:** Die Nachricht (der unverschlüsselte Text) X ist mit dem symmetrischen Schlüssel K verschlüsselt.

[6] Die BAN-Logik kennt das Konzept eines "Angreifers" nicht. Sie befaßt sich lediglich mit Nachrichten von "legitimen" Kommunikationspartnern.

Dies sind die wichtigsten Symbole der BAN-Logik. Es gibt einige weitere Symbole, die für die folgende Argumentation jedoch nicht erforderlich sind.

Neben diesen elementaren Symbolen gibt es eine Schlußregel, mit der neues Wissen auf Basis von a priori vorhandenem Wissen und empfangenen Nachrichten abgeleitet werden kann. Die allgemeine Form dieser Schlußregel lautet:

$$\frac{A_1, A_2, \ldots, A_n}{A_{n+1}, A_{n+2}, \ldots, A_{n+m}} \tag{8.1}$$

Hierbei stellen die n Ausdrücke über dem Balken vorhandenes oder durch vorangegangene Schlußfolgerungen erlangtes Wissen dar. Die Ausdrücke unter dem Balken stellen neues Wissen dar, das aus dem vorhandenen abgeleitet werden kann, gdw. alle A_i mit $1 \leq i \leq n$ erfüllt (wahr) sind. Die folgenden Axiome sind Beispiele für diese allgemeine Schlußregel:

Authentifikation des Senders ("message-meaning")
$A_1 := P|\equiv Q \overset{K}{\longleftrightarrow} P$ "P glaubt, einen geheimen, symmetrischen Schlüssel K für die Kommunikation mit Q zu kennen". $A_2 := P \lhd \{X\}_K$ "P hat eine mit K verschlüsselte Nachricht empfangen". Aus A_1, A_2 wird $A_3 := P|\equiv Q|\sim X$ "P glaubt, Q hat irgendwann einmal X gesagt (gesendet)" gefolgert:

$$\frac{P|\equiv Q \overset{K}{\longleftrightarrow} P, \quad P \lhd \{X\}_K}{P|\equiv Q|\sim X} \tag{8.2}$$

A_3 steht als neues Wissen für weitere Schlußfolgerungen zur Verfügung.

Sichtbarkeit von Teilen einer Nachricht
Ist eine zusammengesetzte Nachricht empfangen worden ($P \lhd (X, Y)$), dann sind auch deren Teile sichtbar:

$$\frac{P \lhd (X, Y)}{P \lhd X, \ P \lhd Y} \tag{8.3}$$

Das gleiche gilt auch für erlangtes Wissen. Wenn man annehmen kann, daß ein Kommunikationspartner eine zusammengesetzte Nachricht akzeptiert, dann kann man auch annehmen, daß er einen Teil davon akzeptiert:

$$\frac{P|\equiv (Q|\equiv (X, Y))}{P|\equiv Q|\equiv X, \ P|\equiv Q|\equiv Y} \tag{8.4}$$

Anfrage/Antwort basierend auf "frischen" Nachrichten ("nonce-verification")
Wenn P glaubt, daß X "frisch" ist, und P außerdem glaubt, daß X von Q irgendwann einmal gesendet wurde, dann glaubt P, daß auch Q den Text X als "frisch" anerkennt:

$$\frac{P|\equiv \#(X), \ P|\equiv \ Q|\sim X}{P|\equiv (Q|\equiv X)} \tag{8.5}$$

Dieses Axiom ist insofern wichtig, als es das einzige Axiom ist, bei dem von $|\sim$ auf $|\equiv$ geschlossen wird. Es handelt sich hierbei um ein abstraktes[7] "Challenge-Response"-Verfahren. P generiert einen frischen Wert X, der zu Q gelangt. Q sendet X verschlüsselt wieder zurück, so daß P zur Überzeugung gelangen kann, daß die Nachricht tatsächlich von Q zurückgesendet wurde ($P|\equiv\ Q|\sim X$). Erst dann kann geschlossen werden, daß Q ebenfalls X gesehen hat und dies im Zuge der Protokollverarbeitung als "Challenge" akzeptiert hat und somit "an X glaubt".

8.4.1.2 Wichtige Ergänzungen der BAN-Logik

Eine der wesentlichen Erweiterungen der GNY-Logik (vgl. [99]) ist, daß zwischen "Glauben" ($P|\equiv X$, "P glaubt, daß X") und vorhandenem Wissen ($P \ni X$, "P kennt X") unterschieden wird. Neu Erlerntes muß explizit in das vorhandene Wissen aufgenommen werden, um für weitere Deduktionen zur Verfügung zu stehen. Neues Wissen kann so auch aus vorhandenem Wissen generiert werden, ohne daß zuvor eine Nachricht empfangen werden muß, beispielsweise:

$$\frac{P \lhd X}{P \ni X} \quad \text{und} \quad \frac{P \ni X,\ P \ni Y}{P \ni (X,Y),\ P \ni F(X,Y)} \tag{8.6}$$

wobei $F()$ eine beliebige Funktion ist.

Mit einem zusätzlichen Symbol ($*$) "not originated here" (nicht von hier stammend), kann man darüber hinaus in der GNY-Logik ausdrücken, daß eine Nachricht empfangen wurde, die definitiv nicht lokal erzeugt wurde. Dies ist eine Spezialform des "recognisable"-Symbols ($\phi(X)$), das besagt, daß eine Instanz die Nachricht X als solche erkennen würde. Die Nachricht ist noch nicht empfangen worden, könnte aber aufgrund von Informationen (unverwechselbar) bei Empfang als X klassifiziert werden.[8] Die BAN-Logik kann dies nicht ausdrücken. Allerdings wird dort grundsätzlich angenommen, daß eine Instanz lokale Nachrichten von fremdgenerierten Nachrichten unterscheiden kann und Teile einer zusammengesetzten Nachricht unterschieden werden können (falls die Einzelteile eigenständige Bedeutung haben).

"message meaning" der GNY-Logik
Auch bei der GNY-Logik gibt es ein "message-meaning"-Axiom, das im Vergleich zum Axi-

[7] Das "Challenge-Response"-Verfahren abstrahiert von der notwendigen Verschlüsselung der Nachrichten. Beispielsweise sind für $A_2 :=\ P|\equiv\ Q|\sim X$ mehrere Zwischenschritte und die Anwendung des Axioms (8.2) erforderlich.

[8] Der Abstraktionsprozeß bei der Protokollbeschreibung mit Hilfe der hier vorgestellten Logiken postuliert bestimmte Eigenschaften (beispielsweise $\phi(X)$), deren Einhaltung im Zuge der Protokollspezifikation umgesetzt werden muß. Hierfür können ebenfalls formale Techniken eingesetzt werden. Die Erkennung bestimmter Daten kann man beispielsweise mit formalen Methoden zur Typprüfung belegen [31].

om der BAN-Logik (vgl. Formel 8.2) deutlich komplexer ist, da hier alle Voraussetzungen und sämtliches neues Wissen explizit aufgeführt sind:

$$\frac{P \lhd *\{X\}_K,\ P \ni K,\ P|\equiv P \overset{K}{\longleftrightarrow} Q,\ P|\equiv \phi(X),\ P|\equiv \#(X,K)}{P|\equiv Q|\sim X,\ P|\equiv Q|\sim \{X\}_K,\ P|\equiv Q \ni K} \tag{8.7}$$

Aus $A_1 :=$ "P sieht eine mit K verschlüsselte Nachricht, die nicht von P stammt", $A_2 :=$ "P kennt K", $A_3 :=$ "P glaubt, K ist ein symmetrischer Schlüssel für die vertrauliche Kommunikation mit Q", $A_4 :=$ "P glaubt, daß er X erkennen würde" (ggf. erst nach einer Entschlüsselung), $A_5 :=$ "P glaubt, daß X und K 'frisch´ sind" wird gefolgert, daß: $A_6 :=$ P glaubt "Q ist im Besitz des Schlüssels K, und hat irgendwann einmal $\{X\}_K$ und X gesendet (conveyed)."

Dieses sind die aus der Logik unmittelbar ableitbaren Erkenntnisse. Bei der Analyse eines Protokolls müssen diese Aussagen nach Auffassung des Autors wieder im Kontext der Diensterbringung durch das Protokoll interpretiert werden. Die oben genannte Schlußfolgerung (A_6) könnte im *Kontext eines solchen Protokollablaufs* beispielsweise bedeuten: *"Ich (P) habe eine Nachricht X empfangen, die nicht von mir generiert wurde. Ich erkenne diese Nachricht als mit dem Schlüssel K verschlüsselt, den außer mir nur noch mein Kommunikationspartner Q kennt. Da weder er noch ich den Schlüssel oder die Nachricht veröffentlicht haben (A_5), muß Q die Nachricht verschlüsselt haben. Q wollte mir somit vertraulich X mitteilen (bzw. Q hat $\{X\}_K$ gesendet und X gemeint)."*

8.4.2 Automatenmodell-basierte Ansätze

Direkter auf Sicherheitsaspekte gehen die Automatenmodell-basierten Ansätze ein. Bei diesen Ansätzen wird neben dem zu untersuchenden Protokoll bzw. den Protokollinstanzen ein Angreifer ("active saboteur", [68]) modelliert. Es werden dem Angreifer bestimmte Fähigkeiten zugeordnet, wie beispielsweise die Möglichkeit zum Abhören, Modifizieren und Wiedereinspielen ("replay") und Vernichten von abgehörten Nachrichten.

Die Protokollinstanzen und der Angreifer können dann beispielsweise als endliche Automaten modelliert werden, um eine automatische Analyse des möglichen Verhaltens bei verschiedenen Protokollabläufen ("sessions" oder "runs") mit Hilfe von Simulationswerkzeugen zu gewinnen. Typischerweise wird der Angreifer als Teil des Kommunikationsmediums modelliert. Das nunmehr "maliziöse" Kommunikationsmedium empfängt nicht nur die Nachrichten von den Instanzen, um sie an die Kommunikationspartner weiterzuleiten, sondern kompromittiert ggf. die Vertraulichkeit (Abhören) und die Integrität (Modifizieren) der Nachrichten. Darüber hinaus verallgemeinert dieses "maliziöse" Übertragungsmedium typische technische Eigenschaften eines Übertragungsmediums: zufällige Paketverluste werden ggf. aktiv herbeigeführt (Vernichten von Dateneinheiten) und fehlerhafte Paketverdoppelungen können ebenfalls systematisch provoziert werden (Wiedereinspielen) (vgl. Abschnitt 8.1). Bei diesen "model checkern" (vgl. [165,156]) werden unsichere Zustände oder auch unsichere Zustandsfolgen festgelegt und mit dem bei der Automaten-basierten

Protokollanalyse üblichen Verfahren (Durchsuchen des Zustandsraumes, [120, S.218ff]) untersucht. Wird ein als unsicher klassifizierter Zustand aus dem Startzustand erreicht, dann ist ein Sicherheitsproblem gefunden. Das Hauptproblem bei der Analyse ist, daß bei nichttrivialen Systemen die Zustandsräume sehr groß werden, ein vollständiges Absuchen daher nicht immer gelingt.

Der Hauptnachteil dieser Modell-basierten Methoden ist somit, daß ihr Erfolg sehr stark von der Genauigkeit des Modells abhängt. Fehler in Protokollmechanismen, von denen bewußt oder unbewußt im Zuge der Modellbildung abstrahiert wird (beispielsweise um den Zustandsraum für die Analyse überschaubar zu halten), können später bei der Simulation nicht gefunden werden.

8.4.3 Hybride Methoden

In [35] wird eine hybride Methode beschrieben, die logische Ausdrücke basierend auf der Notation aus [99] und somit basierend auf [41] benutzt, um den Zustandsraum für den "model checker" zu verringern. In [202] wird ein formales System beschrieben, das auf Basis eines logischen Deduktionssystems Protokollbeschreibungen in CSP[9] analysiert.

Eine weitere hybride Methode wird in [91] vorgestellt. Auch hier wird angenommen, daß es einen Angreifer ("active saboteur", [68]) mit bestimmten Fähigkeiten gibt. Anhand von lokal, bei den beteiligten Instanzen beobachtbaren (Protokoll-)Ereignissen, den sog. "Strands", können dann Aussagen über die Korrektheit des Protokolls in Abhängigkeit von den Fähigkeiten des Angreifers abgeleitet werden. Diese Methode kommt ohne ein Simulationswerkzeug ("model checker") aus, erfordert jedoch analog zur BAN-Logik einen ganz erheblichen (fehleranfälligen) Abstraktionsprozeß.

Eine Kombination aus "model checker" und Deduktionssystem wird in [113] vorgeschlagen. Der Autor verweist auf den hohen Aufwand, um eine korrekte formale Beschreibung für geforderte Sicherheitsaspekte zu formulieren. Die formale Beschreibung der Sicherheitsanforderungen erforderte mehr Zeit als die anschließende automatische Validation.

8.4.4 Offene Probleme und Schlußfolgerungen

Die beiden vorgestellten Techniken zur Analyse von Protokollen mit Sicherheitsfunktionalität lassen eine Reihe von Problemen offen. Vordringlich fällt auf, daß keine der Logikbasierten Methoden eine Möglichkeit vorsieht, den Aufwand für die Protokollschritte zu beschreiben. Dasselbe trifft auch auf die Modell-basierten Ansätze zu, die sich aber leichter um derartige Parameter ergänzen ließen. In der jüngeren Vergangenheit (vgl. [209]) häufen sich allerdings die sog. "Denial of Service"-Angriffe (DoS, vgl. [48, 150, 153, 215, 89, 163, 96, 207, 209, 89]), obwohl die Gefahren schon seit langem bekannt sind (beispielsweise [96]).

[9] Communicating Sequential Processes

Hierbei handelt es sich um Angriffe mit dem Ziel, autorisierten Benutzern den Zugriff auf ihre Dienste zu erschweren oder diesen Zugriff sogar vollständig zu unterbinden. Es gibt zwei typische Angriffsstrategien, die bei DoS-Angriffen immer wieder zu beobachten sind:

1. Es werden Fehler der Implementation ausgenutzt, um den Prozeß, der den Dienst erbringt, zu terminieren. Es handelt sich somit um echte "faults" im Sinne des Fehlermanagements (vgl. [147]), die vom Angreifer systematisch ausgebeutet werden, um gezielt einen Ausfall des Dienstes zu bewirken. Dieser Art von DoS-Angriffen kann daher auch mit den Methoden des klassischen Fehlermanagements begegnet werden.

2. Die für die Diensterbringung erforderlichen Ressourcen werden durch den Angreifer gezielt aufgebraucht. Der Dienst ist dann für andere Benutzer (vorübergehend) nicht erreichbar (temporärer Ausfall) oder die Auftragsbearbeitung leidet unter erheblichen Funktions- bzw. Leistungseinbußen ("degradation").

Es ist der zweite Angriffstyp, der im Zuge des Entwurfs eines überlebensfähigen Systems genauer betrachtet werden muß, da sich hier erneut ein "Tradeoff" zwischen Sicherheit und Verfügbarkeit[10] ergibt.

Der Angreifer versucht beim zweiten Angriffstyp in der Regel, den Dienst mit maliziösen Aufträgen zu überlasten, so daß wenige oder gar keine Ressourcen für die Diensterbringung von Aufträgen autorisierter Benutzer übrig bleiben. Gerade für diese Abschätzung, des mit der Protokollverarbeitung einhergehenden Aufwands, wären entsprechende formale Beschreibungen erforderlich. Am Beispiel der BAN-Logik läßt sich das Problem verdeutlichen: Das folgende Axiom "message meaning" aus [41] beschreibt die Bedeutung einer eintreffenden, verschlüsselten Nachricht:

$$\frac{P \mid\equiv Q \overset{K}{\leftrightarrow} P, \quad P \lhd \{X\}_K}{P \mid\equiv Q \mid\sim X} \tag{8.8}$$

Unabhängig davon, für wie realistisch man diese Axiome einschätzt und wie gut oder schlecht sie geeignet sind, um Sicherheitsprobleme bei der Authentifikation nachzuweisen, erkennt man hieran sofort zwei Probleme:

- Es wird nichts über den Verschlüsselungsalgorithmus und die Schlüssellänge ausgesagt. Die Frage der Sicherheits-Dienstgüte (SQoS, vgl. [7, 70]) wird nicht weiter betrachtet und muß gesondert analysiert werden.

- Das Axiom abstrahiert vom Aufwand für die Entschlüsselung von X auf der Seite des Empfängers (hier P). In der formalen Darstellung ist zwar ersichtlich, daß es sich um den verschlüsselten Text X handelt ($\{X\}_K$), der Klartext der Nachricht (X), auf

[10] Die Verfügbarkeit wird im Zuge des Sicherheitsmanagements ebenfalls betrachtet. Man könnte daher auch sagen, daß es einen "Tradeoff" zwischen verschiedenen Sicherungszielen gibt.

den im Axiom auch geschlossen wird, ist in realen Systemen jedoch erst nach einer Entschlüsselung mit dem symmetrischen Schlüssel K bekannt. Der Aufwand für Ver- und Entschlüsselung ist im Vergleich zum Aufwand für den Empfang der Nachricht erheblich größer (vgl. [24, S.84ff]).

Dieser Aufwand kann von einem Angreifer aber für einen der oben genannten DoS-Angriffe verwendet werden, beispielsweise indem er durch häufiges Wiedereinspielen von $\{X\}_K$ den Empfänger P zum mehrfachen Entschlüsseln der Nachricht veranlaßt.

Der "Tradeoff" zwischen Sicherheit und Verfügbarkeit kann mit Hilfe der BAN-Logik nicht ausgedrückt werden. Typischerweise sind aufwendige kryptographische Mechanismen schwerer durch Kryptoanalyse zu brechen und insofern sicherer als weniger aufwendige kryptographische Mechanismen. Andererseits sind es gerade die aufwendigen kryptographischen Mechanismen, die besonders leicht von einem Angreifer für einen DoS-Angriff ausgenutzt werden können. Der Angreifer kann mit vergleichsweise wenigen Aufträgen eine hohe Last und/oder einen hohen Betriebsmittelbedarf für die Auftragsbearbeitung auf dem angegriffenen System erzeugen.

Es gibt erste formale Ansätze, diesen Aufwand beim Einsatz von sicheren Protokollen zu beschreiben (vgl. [163,164]) und bei Implementationen aufwandsarme Mechanismen einzusetzen [13]. Die typische Vorgehensweise ist, bei einem Protokollablauf[11] den anfallenden Aufwand zu notieren und bei zu großem Aufwand den Protokollablauf (mit einem Fehler) zu beenden (vgl. [163,100]). Vollständig verstanden ist diese Vorgehensweise jedoch nicht. Es gibt hierbei nach Auffassung des Autors mindestens drei Probleme:

- Nach einem Abbruch des Protokollablaufs durch einen der beiden Kommunikationspartner hindert nichts den zweiten Kommunikationspartner (Angreifer) einen weiteren Protokollablauf zu initiieren. Es kann durch den "Fail-Stop"-Ansatz somit lediglich der Aufwand für *einen* Protokollablauf begrenzt werden. Der Dienst ist immer noch durch Wiederholung der Abläufe oder durch mehrere gleichzeitige Abläufe angreifbar. Die vollständige Einstellung der Protokollverarbeitung wäre sicherlich keine Alternative. Der Angreifer wäre dann noch erfolgreicher: Anstelle einer Dienstverminderung ("degradation") hätte er einen Dienstausfall (aus Benutzersicht) erreicht.

- Das "Fail-Stop"-Prinzip ist ungeeignet für multilaterale Kommunikationsbeziehungen. Wenn beispielsweise für eine Videokonferenz viel Aufwand betrieben wurde, um n-Kommunikationspartner miteinander zu synchronisieren, dann kann der Protokollablauf nicht einfach abgebrochen werden, falls der $(n+1)$-te Kommunikationspartner ein Angreifer ist.

[11] Es muß die Protokollverarbeitung von einem Protokollablauf unterschieden werden. Ein Protokollablauf ist die Ausführung des Protokolls durch eine Instanz auf einem System. Es kann mehrere, voneinander unabhängige Protokollabläufe pro Protokoll zeitgleich auf einem System geben (beispielsweise mehrere TCP-Verbindungen entsprechen verschiedenen TCP-Protokollabläufen). Die Protokollverarbeitung ist die Menge aller Protokollabläufe eines Protokolls (beispielsweise TCP-Protokollverarbeitung).

- Das "Fail-Stop"-Prinzip ist ungeeignet für ständig erforderliche Kommunikationsbeziehungen. Beispielsweise kann der Protokollablauf zur Fehlererkennung beim Gruppenmanagement nicht einfach abgebrochen werden, nur weil es durch einen Angreifer zu aufwendig für einzelne oder alle Instanzen wird. Der Dienst ist essentiell und muß in jedem Fall erbracht werden.

Zusammenfassend läßt sich feststellen, daß die verschiedenen formalen Analysemethoden viele wichtige, sicherheitsrelevante Aspekte ausklammern. Sie sind nicht in der Lage, die Sicherheit eines Protokolls zu beweisen [100], sondern nur bestimmte Fehlerarten aufzufinden.

Sie erfordern in der Mehrzahl einen fehleranfälligen Abstraktionsprozeß bei der Überführung in eine logische Darstellung des Protokolls oder die Modellbildung für proprietäre (teilweise nicht verfügbare) "modell checker". Weiterhin ist die Möglichkeit, die Auswirkungen von DoS-Angriffen formal zu beschreiben, heute noch nicht richtig verstanden.

In dieser Arbeit wird daher ein anderer Weg untersucht. Im Zuge der Protokollspezifikation werden ohnehin "Message Sequence Charts" (MSCs) als standardisierte Beschreibung von Nachrichtenabläufen eingesetzt, um einen ersten Entwurf des Kommunikationsablaufes darzustellen. Mit dieser "halbformalen" Methode lassen sich auch kryptographische Protokolle und die dabei gemachten Annahmen darstellen (vgl. beispielsweise [24, 23]). Diese Szenarien müssen zwar manuell generiert werden, dafür lassen sich so aber alle möglichen Freiheitsgrade eines Angreifers nachbilden. Zusätzlich kann bei Bedarf in den Beschriftungen eine BAN-Logik basierte Annotation angebracht werden (vgl. Anhang A). Es können auch Aktionen eingezeichnet werden, die den Betriebsmittelverbrauch für eingehende oder abgehende Nachrichten beschreiben, ohne hierfür einen neuen Formalismus wie in [163] einzuführen.

8.5 Sicherheitsprobleme, Bedrohungspotential und erforderliche Sicherheitsmechanismen

In diesem Abschnitt werden die konkreten Sicherheitsprobleme in Hinblick auf die Sicherheitsanforderungen vorgestellt. Die einzelnen Sicherheitsprobleme können von Angreifern ausgenutzt werden. Diese konkreten Bedrohungen des ungesicherten GM-Protokolls werden mit Hilfe von MSCs dargestellt. MCSs als halbformale, graphische Darstellung der Kommunikationsabläufe, bieten die Möglichkeit, die kausalen Zusammenhänge zwischen Angriff und Protokollabläufen darzustellen. Ihre Eignung für die protokollspezifische Sicherheitsanalyse wird in [23] gezeigt. Eine Einführung in die Protokollanalyse mit MSCs ist in Anhang A dargestellt. Dort wird auch beschrieben, wie die eben diskutierten, formalen Beschreibungsmethoden (BAN-Logik und Kostenannotationen) in Standard-MSCs integriert werden können, um so die Vorteile der Beschreibungstechniken miteinander zu kombinieren.

In den folgenden Unterabschnitten werden die Bedrohungen in Hinblick auf die in Abschnitt 8.2 vorgestellten Sicherheitsanforderungen (SAs) an das Gruppenmanagement-Protokoll diskutiert.

8.5.1 Zu SA1: Nur autorisierte Prozessoren dürfen in die Gruppe aufgenommen werden

Das generische Sicherungsziel hinter dieser Anforderung ist, daß die Identität der Gruppenmitglieder zweifelsfrei festgestellt werden muß. Die Aufgabe der Gruppe ist die Erfüllung eines bestimmten Dienstes (hier die Zugriffskontrolle auf Dateneinheiten). Diese Aufgabe kann nur von dafür geeigneten Prozessoren durchgeführt werden. Eine Autorisierung zur Partizipation an der Diensterbringung darf demnach erst nach einer Authentifikation (sichere Identifikation) erfolgen. "SA1" erfordert daher Mechanismen zur Authentifikation.

Erfolgt eine derartige Authentifikation nicht, könnte ein Angreifer Gruppenmitglied werden und anschließend Dateneinheiten durch die parallele Packet Screen tunneln. In Abbildung 8.1 ist dieser Verstoß gegen "SA1" graphisch dargestellt.

Als Gegenmaßnahme bieten sich kryptographische Verfahren an, die eine Authentifikation der (potentiellen) Gruppenmitglieder untereinander ermöglichen. Geht man von den bekannten kryptographischen Verfahren aus, so hat man die Möglichkeit, symmetrische oder asymmetrische Verfahren für die Authentifikation einzusetzen (vgl. [206]).

Setzt man asymmetrische Kryptographie ein, erfordert dies für jedes Gruppenmitglied ein individuelles Schlüsselpaar. Eine Einweg-Streufunktion (vgl. [192]) könnte zur Berechnung eines "Message Authentication Codes, MAC" verwendet werden, der anschließend mit dem privaten Schlüssel der Instanz verschlüsselt wird. Über den öffentlichen Schlüssel dieser Instanz könnten die anderen Instanzen bei Empfang dieser Dateneinheit sowohl die Integrität der Nachricht als auch die Authentizität überprüfen. Der Hauptnachteil beim Einsatz asymmetrischer Kryptographie ist der im direkten Vergleich mit symmetrischer Kryptographie deutlich höhere Rechenaufwand, um eine vergleichbare Sicherheit zu erreichen (vgl. [24]).

Die Entscheidung für asymmetrische oder symmetrische Verschlüsselung zieht auch deutliche Unterschiede im Schlüsselmanagement nach sich. Während die Verwendung von asymmetrischer Kryptographie i.d.R. Zugriff auf einen oder mehrere Zertifikatsdienste erfordert, über die dynamisch auf die öffentlichen Schlüssel der Kommunikationspartner zugegriffen werden kann, werden symmetrische Schlüssel entweder a priori vor der Benutzung zwischen zwei Kommunikationspartnern etabliert oder durch einen Schlüsseldienst (dem alle beteiligten Kommunikationspartner trauen müssen) dynamisch bereitgestellt. Ein leichtgewichtiges GMP sollte ohne zusätzliche Abhängigkeiten von Schlüsseldiensten auskommen. Der Zugriff auf die Schlüssel muß schnell erfolgen können und sollte daher nicht dynamisch erfolgen. Außerdem kann so eine neue Fehler- und Gefahrenquelle vermieden werden. Der Ausfall des Schlüsseldienstes kann die Verfügbarkeit des GMPs nicht beeinträchtigen. Die

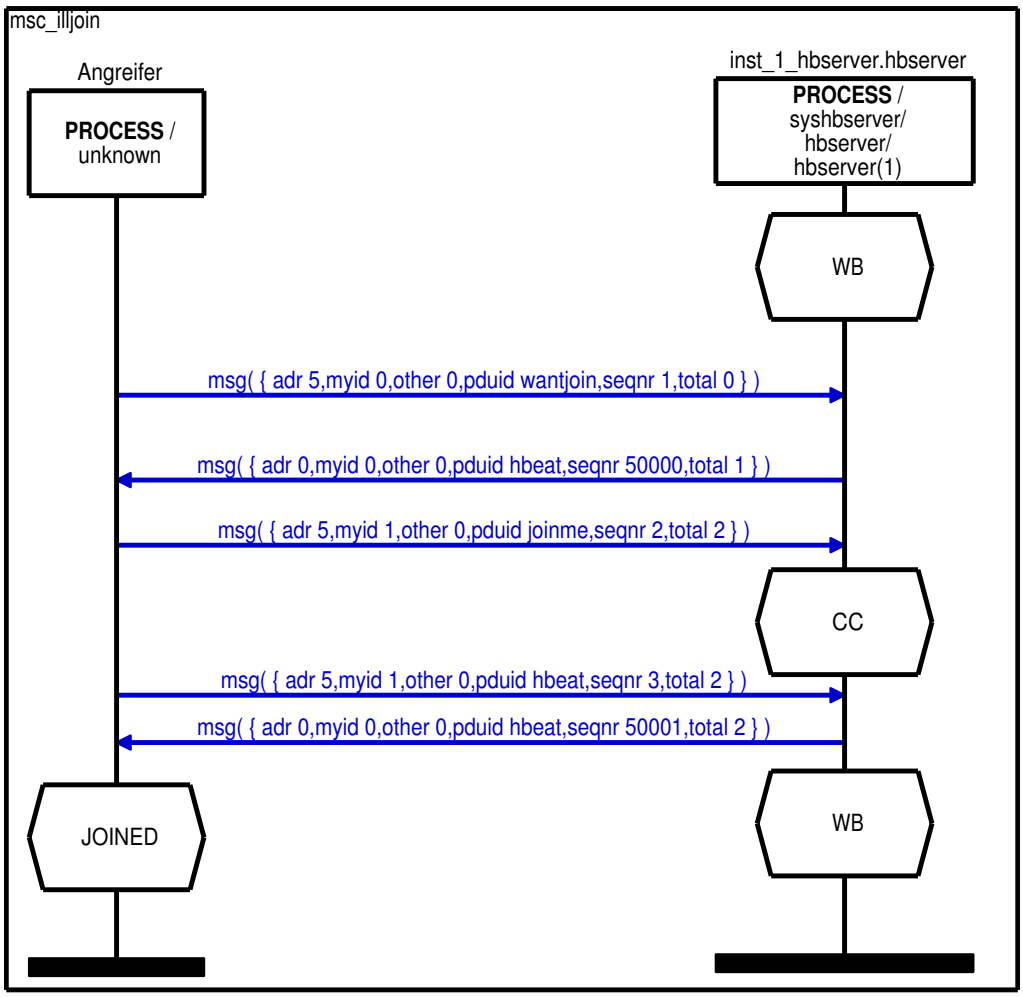


Abbildung 8.1: Nicht autorisierte Gruppenmitgliedschaft

Sicherheit des GMPs ist ebenfalls nicht von der Sicherheit des Schlüsseldienstes bei der Verwendung von symmetrischen Schlüsseln abhängig.

Es liegt demnach nahe, symmetrische Kryptographie für die Integritätsprüfung der Nachricht und die Authentifikation der Gruppenmitglieder zu verwenden. In einem ersten Ansatz würde man $(n * (n-1))/2$ symmetrische Schlüssel generieren, so daß sich jeweils zwei Instanzen untereinander authentifizieren könnten. Außerdem könnte der Schlüssel mit in die Berechnung des MAC-Wertes einbezogen werden, so daß auch die Integrität der Nachricht überprüft werden könnte (vgl. [198, 140]). Über den verwendeten symmetrischen Schlüssel könnte jede Instanz verifizieren, von welcher anderen Instanz die jeweilige Nachricht stammt. Der Nachteil dieser Methode liegt darin, daß nun keine "Multicast"-Nachrichten mehr versendet werden könnten, da in diesem Fall jeweils ein individueller "Keyed-MAC" für jede andere Instanz in die Nachricht kopiert werden müßte. Der Auf-

wand wäre abhängig von der Anzahl der Instanzen (vgl. [32]) und nicht konstant wie beim Einsatz der asymmetrischen Kryptographie.

Will man aus Gründen der Leistungsfähigkeit dennoch symmetrische Kryptographie bei konstantem Rechenaufwand pro Nachricht einsetzen, bleibt nur noch die Alternative, einen gemeinsamen Schlüssel für alle Gruppenmitglieder zu verwenden. Es kann dann über den Ansatz des "Keyed-MAC' (vgl. [140]) die Integrität der Nachrichten von allen Instanzen überprüft werden. Eine eindeutige Authentifikation ist allerdings nicht mehr möglich. Jedoch kann für jede eintreffende Nachricht validiert werden, ob sie von einem (potentiellen) Gruppenmitglied stammt. Diese Eigenschaft ist in vielen Fällen (und auch in diesem Fall) ausreichend.

Wie bei den meisten kryptographischen Protokollen ist auch in diesem Fall die Grundannahme, daß keine GMP-Instanz den geheimen Schlüssel kompromittiert, d.h. diesen Schlüssel an Dritte weitergibt oder mit diesem Schlüssel Nachrichten von Dritten verschlüsselt/signiert. Außerdem gilt auch bei dem GMP, daß alle Instanzen ein gemeinsames Interesse an der Protokollverarbeitung bzw. Diensterbringung haben. Da auf allen Instanzen die gleiche Policy gilt (dieselbe Sicherheitsdomäne, vgl. Definition (2.4)) kann der Fall ausgeschlossen werden, daß eine Instanz versucht, sich durch Täuschung (mittels maliziöser Protokollabläufe) Vorteile zu erschleichen. Die impliziten Annahmen für eine Sicherheitsanalyse mit der BAN-Logik (s.u.) sind daher im Fall dieses GMPs gültig.

In der folgenden Abbildung ist die Authentifikation eines Bewerbers bei einem Gruppenmitglied mit Hilfe des "Keyed-MAC' abgebildet. $H(K, \ldots)$ gibt die Berechnung des Hash-Wertes an, in den der Schlüssel K mit einbezogen ist. Die Schlußfolgerung auf der Seite des Gruppenmitglieds ist anhand der BAN-Logik notiert. Es handelt sich hierbei im wesentlichen um das in [99, S.247] vorgestellte "Message Interpretation Axiom, No. 3". Die Abbildung verdeutlicht somit auch, wie die Annahmen und Schlußfolgerungen der BAN-Logik mit Hilfe von Kommentaren als Annotationen in die MSC-Notation eingetragen werden können. Nicht Bestandteil der BAN-Logik, aber Motivation für die Einführung des "Keyed-MAC' (vgl. [140]) ist, daß für die *Authentifikation* einer Nachricht msg gilt:

$$\{msg\}_K \equiv msg, H(K, msg) \tag{8.9}$$

Die mit dem gemeinsamen Schlüssel verschlüsselte Nachricht ist Beweis[12] für die Herkunft der Nachricht. Die Nachricht, zusammen mit dem Hash-Wert dieser Nachricht, in dessen Berechnung der gemeinsame Schlüssel mit einbezogen wurde, sind ebenfalls Beleg für die Herkunft der Nachricht. Es ist übrigens nicht erforderlich, daß msg und $H(K, msg)$ in derselben Nachricht versendet werden. Es muß nur ersichtlich sein, daß nach Empfang von msg und $H(K, msg)$ letzteres der Hash-Wert des anderen ist und dieser Hash-Wert tatsächlich von einer Partnerinstanz und nicht von der lokalen Instanz berechnet wurde. Erweiterungen der BAN-Logik ermöglichen dies mit dem "not originated here"- und

[12] Für symmetrische Schlüssel gilt: Der Empfänger weiß, daß die Nachricht vom Kommunikationspartner stammt oder von jemandem, der ebenfalls in Besitz des gemeinsamen Schlüssels K ist.

"recognisable"-Operatoren explizit zu formulieren. Die Annahme aus Formel 8.9 wird implizit in Abbildung 8.2 benutzt.

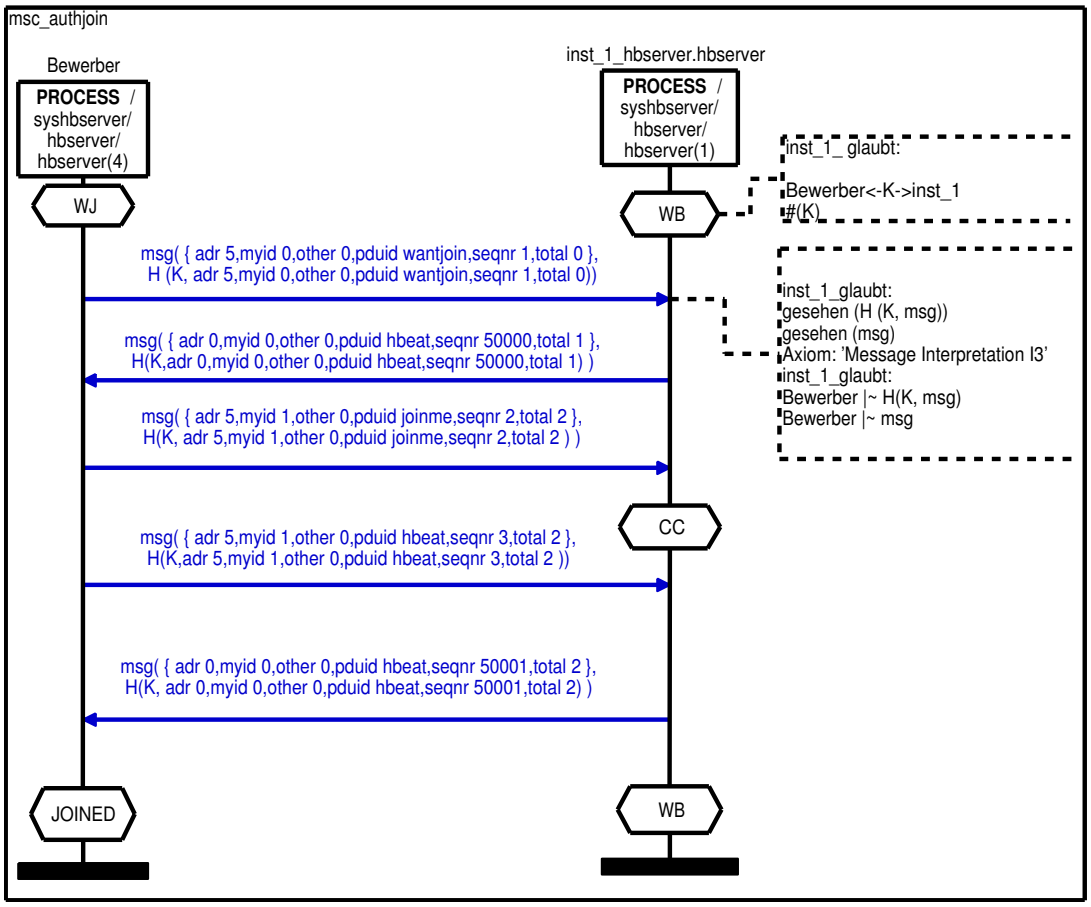


Abbildung 8.2: Authentifikation der Nachrichten durch "Keyed-MAC", Annotation gemäß der BAN/GNY-Logik

8.5.2 Zu SA2: Autorisierten Prozessoren darf die Aufnahme in die Gruppe nicht verweigert werden

Das generische Sicherungsziel hinter dieser Anforderung ist die Wahrung der Verfügbarkeit. Hindert ein Angreifer Gruppenmitglieder am Beitritt zur Gruppe, ist dies ein Angriff auf die Verfügbarkeit des Dienstes, der durch die Gruppe erbracht werden soll. Im schlimmsten Fall hindert der Angreifer bereits den ersten Bewerber an der Gruppenbildung. Der Dienst, den die Gruppe erbringen soll, kann dann überhaupt nicht angeboten werden.

In anderen Fällen leidet die Leistungsfähigkeit und ggf. die Verfügbarkeit des Dienstes, da jeder nicht-aufgenommene, autorisierte Bewerber nicht an der Diensterbringung partizi-

piert und bei Ausfall eines Gruppenmitglieds nicht als aktive Redundanz zur Verfügung steht. In Abbildung 8.3 ist dieser Verstoß gegen "SA2" graphisch dargestellt.

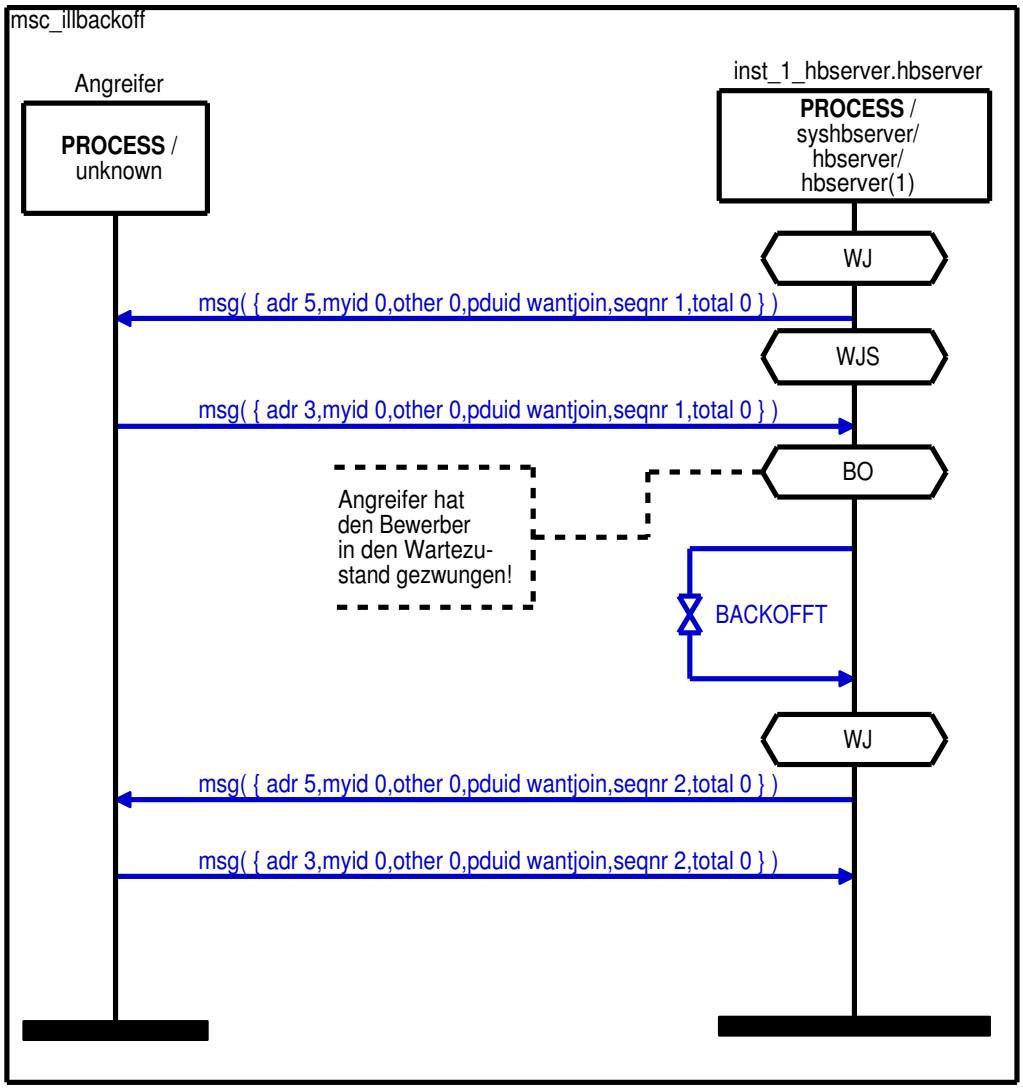


Abbildung 8.3: Angreifer verhindert die Aufnahme eines Bewerbers in die Gruppe

Der Angreifer wartet bis ein "WantJoin" eines Bewerbers eintrifft und sendet seinerseits ein hochpriorisiertes "WantJoin". Dies erreicht der Angreifer leicht durch die Wahl eines Prozessornamens, so daß nach Definition 7.1 gilt:

$$p_{Angreifer} \lessdot p_{Bewerber} \tag{8.10}$$

Der Bewerber würde dann in den Wartezustand "Backoff" wechseln. Der Angreifer kann diesen Vorgang beliebig häufig wiederholen und somit den Bewerber permanent von der Aufnahme in die Gruppe abhalten bzw. die Gruppenbildung verhindern.

Dieser Angriff ist sehr gefährlich, da er sowohl sehr schädlich ist und die Angriffsschwelle außerdem sehr gering ist. Der Angreifer braucht nicht wie in Abbildung 8.3 die hochpriorisierte "WantJoin"-Nachricht als Reaktion auf die Nachricht eines Bewerbers zu schicken. Er kann vielmehr diese Nachricht periodisch (z.B. ein mal pro $HB_{intervall}$) senden und somit aufwandsarm[13] den Angriff durchführen.

In einem ersten Ansatz führt dieser Angriff wiederum direkt zum Einsatz von kryptographischen Mechanismen. Falls ein Bewerber nicht in der Lage ist, den richtigen "Keyed-MAC" seiner Nachricht anzufügen, erkennen die anderen Instanzen diesen sofort als Angreifer und ein gleichzeitiger autorisierter Bewerber würde sich nicht verdrängen lassen, da auch er den falschen "Keyed-MAC" des Mitbewerbers erkennen kann.

Je nach den Fähigkeiten des Angreifers erfordert dieses Problem jedoch noch zusätzliche Mechanismen. Konnte ein Angreifer beispielsweise die "WantJoin"-Nachricht einer anderen Instanz (I_a) abhören, so verfügt er über eine gültige Nachricht (inklusive MAC-Wert). Bewirbt sich zu einem späteren Zeitpunkt eine weitere Instanz (I_b), und es gilt $I_a \lessdot I_b$, so kann der Angreifer Instanz I_b durch Wiedereinspielen der aufgezeichneten "WantJoin"-Nachricht von I_a wiederum verdrängen.

Um diesen Angriff abzuwehren sind drei weitere Mechanismen erforderlich:

- Von aktiven Gruppenmitgliedern werden nur "HB"-Nachrichten akzeptiert. Dieser Mechanismus schließt den Fall aus, daß eine wiedereingespielte Nachricht eines aktiven Gruppenmitglieds einen anderen Bewerber verdrängt (s.o.).

- Es wird von aktiven Mitgliedern eine einfache Authentifikation des Bewerbers verlangt, indem sie den Bewerber auffordern, eine bestimmte initiale Sequenznummer zu verwenden. In dem Bestätigungspaket für einen Bewerber wird dessen Sequenznummer von einem oder mehreren Gruppenmitgliedern auf einen Wert gesetzt, der in jedem Fall "frisch" ist. Der Bewerbung wird erst dann zugestimmt, wenn der Bewerber eine gültige "WantJoin"-Nachricht mit der angeforderten Sequenznummer sendet. Durch dieses einfache "Challenge-Response"-Verfahren kann sich ein Bewerber gegenüber den Gruppenmitgliedern authentifizieren. Außerdem wird ein noch im Zuge der Bewerbung befindlicher Mitbewerber nur dann durch eine priorisierte Bewerbung verdrängt, wenn der neue Bewerber sowohl über höhere Priorität als auch über eine (im obigen Sinne) authentisierte Bewerbung mit "frischer" Sequenznummer verfügt.

- Ein Bewerber tritt nur dann zurück, wenn die eigene Sequenznummer und die Priorität kleiner ist als die eines Mitbewerbers. Vor dem Zurücktreten merkt sich der Bewerber die höchste empfangene Sequenznummer (entweder aus der "WJ"-Nachricht

[13] Der Angreifer benötigt keine Protokollfunktionen, die eine Interpretation der Nachrichten der Bewerber durchführen. Durch das periodische Senden der hochpriorisierten Nachrichten erübrigt sich diese gezielte Reaktion auf einzelne Nachrichten der Bewerber.

des Mitbewerbers oder idealerweise aus einer zwischenzeitlich eingetroffenen "HB"-Nachricht), um nach der Wartezeit aufgrund dieser Sequenznummer eine "frische"-Nummer zu berechnen (s.u.).

Mit diesen zusätzlichen Mechanismen wird erreicht, daß ein autorisierter Bewerber schließlich immer in die Gruppe aufgenommen wird. Leider ist die Bereitstellung bzw. Generierung einer "frischen" Sequenznummer für einen Bewerber durch Gruppenmitglieder nicht ganz trivial. Die Gruppenmitglieder brauchen hierfür Kenntnis über die zuletzt von diesem Bewerber als aktives Gruppenmitglied verwendete Sequenznummer (*oldseqnr*), so daß sie einen "frischen" Wert als initiale Sequenznummer beim Bewerber anfordern (*newseqnr*).

Jedes Gruppenmitglied muß daher eine "Ehemaligen-Liste" (Alumni-Liste) führen, in dem alle jemals als Gruppenmitglied aktiven Prozesse mit der zuletzt von ihnen verwendeten Sequenznummer gespeichert werden. Offensichtlich muß diese Liste bei Ausfall eines Prozessors aktualisiert und auf "stabilem Speicher" gesichert werden. Jedes Gruppenmitglied kann dann in der "Ehemaligen-Liste" bei einer eintreffenden Bewerbung nachsehen, ob die Sequenznummer größer ist als die letzte bekannte Sequenznummer. Ist dies nicht der Fall, generiert das Gruppenmitglied aufgrund der Kenntnis von *oldseqnr* aus der "Ehemaligen-Liste" eine frische Sequenznummer, so daß gilt: $oldseqnr < newseqnr$ und verlangt hierfür eine Authentifikation.

Zusätzlich ist es noch erforderlich, daß bei mehreren Anforderungen zur Authentifikation der Bewerbung erst dann zugestimmt wird, wenn die Authentifikation für die größte ("frischste") Sequenznummer erfolgt ist. Dies ist erforderlich, weil die in der "Ehemaligen-Liste" gespeicherte Sequenznummer nicht notwendigerweise die tatsächlich zuletzt verwendete Sequenznummer des ehemaligen Gruppenmitglieds ist. Die eine oder andere Instanz könnte u.U. ein oder mehrere der letzten "HB"-Nachrichten verloren haben (Paketverluste).

Um ganz sicher zu gehen, daß immer ein wirklich frischer Wert für die Authentifikation angefordert wird, ist folgender Algorithmus hinreichend:

1. Jedes Gruppenmitglied führt eine "Ehemaligen-Liste". Zusammen mit den Namen der Prozessoren wird hier die letzte erkannte (höchste) Sequenznummer gespeichert.

2. Bei jeder Änderung an der Gruppenliste, spätestens jedoch alle t-Sekunden wird:

   (a) Die höchste je verwendete Sequenznummer ermittelt.

   (b) Die Gruppenliste als Ehemaligen-Liste mit der ermittelten Sequenznummer auf stabilen Speicher geschrieben.

3. Liegt eine Bewerbung vor, wird eine Authentifikation basierend auf der frischsten Sequenznummer aus der aktuellen Gruppenliste (*oldseqnr*) für den Bewerber generiert, so daß gilt:

$$newseqnr = oldseqnr + \lceil t/HB_{intervall} \rceil, t > HB_{intervall} \tag{8.11}$$

Hierdurch ist sichergestellt, daß bei Ausfall einer Instanz, der mit Paketverlusten der letzten "HB"-Nachrichten einher geht, die anderen Instanzen in jedem Fall einen frischen Wert beim nächsten Beitrittsversuch anfordern; auch dann, falls sie unmittelbar danach selbst ausfallen und somit nicht die letzte Sequenznummer auf dem stabilen Speicher ausgelagert haben.

4. Es wird die höchste Sequenznummer für die Authentifikation propagiert, d.h. liegen mehrere Authentifikationsanforderungen von anderen Instanzen vor, dann übernehmen alle Gruppenmitglieder die höchste erkannte Sequenznummer aus einer dieser Authentifikationsanfragen für ihre eigene Anfrage (erhöht die Robustheit).

5. Eine Instanz, die sich bewerben möchte, sieht nach, ob sie eine Alumni-Liste auf dem lokalen stabilen Speicher findet. Ist das der Fall, ermittelt sie die größte alte Sequenznummer und berechnet nach Formel 8.11 eine neue Sequenznummer, die sie in die erste "WJ"-Nachricht aufnimmt.

Der obige Algorithmus ist hinreichend, sofern mindestens eine Instanz[14] über die gesamte Zeit verfügbar ist, um für neue Bewerber immer einen frischen Wert für die Authentifikation zu generieren.

Falls durch ein nicht-abwendbares Ereignis alle Instanzen ausfallen, ist es erforderlich, die Instanzen nach der Reparatur mit einem neuen, geheimen Schlüssel auszustatten[15], oder die zuletzt ausgefallene Instanz als erste neu in Betrieb zu nehmen. Wird ein neuer Schlüssel verwendet, darf die initiale Sequenznummer jeder Instanz auch wieder bei Null beginnen.

Die Authentifikation wird somit um einen garantiert "frischen" Wert in der Sequenznummer erweitert, um Angriffe durch Wiedereinspielen abzuwehren, die andernfalls autorisierten Bewerbern die Aufnahme in die Gruppe verweigern würden. Auch hierfür kann man wieder eine BAN-Logik-basierte Annotation in die MSCs aufnehmen. In Abbildung 8.4 ist dargestellt, wie sich ein Gruppenmitglied davon überzeugt, daß er eine frische, gültige Bewerbung ("WJ"-Nachricht) empfangen hat. Die Annahme, daß die Nachricht msg frisch ist (BAN-Logik: $\#(msg)$) beruht auf der in [41, S. 8] motivierten Schlußfolgerung, nach der ein ganzer Ausdruck als "frisch" angesehen werden kann, wenn man das für einen Teilausdruck glaubt:

$$\frac{P|\equiv \#(X)}{P|\equiv \#(X,Y)} \tag{8.12}$$

[14] Das muß nicht immer dieselbe Instanz sein, sondern vielmehr irgendeine Instanz.

[15] Es ist nicht sichergestellt, daß die Instanz mit der aktuellsten "Ehemaligen-Liste" auch die Gruppe neu gründet. Nur unter diesen Umständen wäre ein Beibehalten des alten Schlüssels unter Berücksichtigung der Gleichung (8.11) sicher.

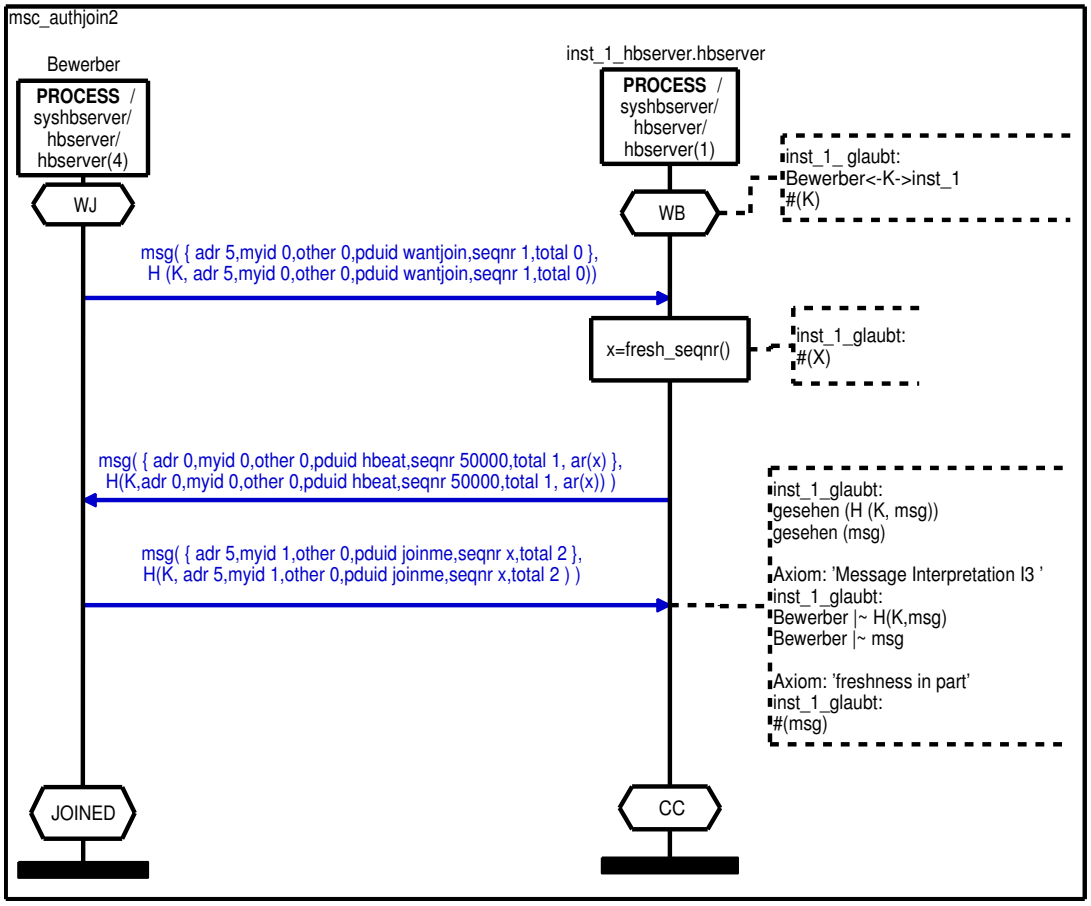


Abbildung 8.4: Authentifikation der Nachrichten durch "Keyed-MAC" und frische Sequenznummer, Annotation gemäß der BAN-Logik

8.5.3 Zu SA3: Die Integrität des Gruppenkonsens muß gewährleistet sein

Auch wenn es einem Angreifer nicht gelingen sollte, sich wie in Abschnitt 8.5.1 dargestellt, in die Gruppe "einzuschleichen", kann er dennoch die Protokollverarbeitung derart stören, daß hierunter der Gruppenkonsens leidet. Eine unterschiedliche Sicht der Gruppenmitglieder auf die Gruppenzusammensetzung führt immer dazu, daß Aufträge von der Gruppe entweder nicht verarbeitet werden oder aber von mehr als einer Instanz verarbeitet werden.

Der erste Fall tritt ein, wenn es dem Angreifer gelingt, durch Stören der Kommunikation den Eindruck zu erwecken, es seien mehr als die tatsächliche Anzahl an Gruppenmitgliedern vorhanden. Der zweite Fall tritt ein, falls es dem Angreifer gelingt, einem einzelnen Gruppenmitglied oder mehreren Gruppenmitgliedern den Ausfall eines andern Mitglieds vorzutäuschen.

Auch in diesem Fall handelt es sich um einen Angriff auf die Verfügbarkeit. Im ersten Fall werden einige Aufträge von der Gruppe gar nicht verarbeitet. Im zweiten Fall leidet die Leistungsfähigkeit der Gruppe, da einzelne Aufträge mehrfach verarbeitet werden. Außerdem steigt die Netzbelastung an, da die Gruppe zu Paketverdoppelungen beiträgt und der Aufwand für die Protokollverarbeitung auf den Rechnern, die mehrfach dieselben Dateneinheiten empfangen, steigt ebenfalls an. Die folgenden Abschnitte stellen die im Vergleich zu den obigen Betrachtungen (hinsichtlich SA1 und SA2) deutlich komplizierteren Angriffsszenarien und Gegenmaßnahmen vor. Abschnitt 8.5.4 stellt einen indirekten Angriff auf den Gruppenkonsens vor, indem der Angreifer versucht, mit IP-Dateneinheiten einen DoS auszulösen, um so den Gruppenkonsens zu stören. Im darauffolgenden Abschnitt 8.5.5 wird ein direkter Angriff vorgestellt, der spezifische Eigenschaften des HB-Protokolls ausnutzt, um einen DoS auszulösen.

Bereits an dieser Stelle sei darauf hingewiesen, daß die hierfür zu erarbeitenden Gegenmaßnahmen einen weiteren Iterationsschritt bei der empfohlenen Vorgehensweise aus Abschnitt 8.2 bedeuten. Der in Abschnitt 8.5.5 dargestellte Angriff ist nur aufgrund der erforderlichen Gegenmaßnahmen möglich, die in den beiden letzten Abschnitten diskutiert wurden. Der Angriff aus Abschnitt 8.5.4 ist aufgrund des Selektionsalgorithmus aus Abschnitt 42 möglich. Die Erhöhung der Leistungsfähigkeit ermöglicht somit neue Angriffsmöglichkeiten, die nun ebenfalls (iterativ) betrachtet werden müssen.

8.5.4 "Denial of Service"-Angriff durch IP-Dateneinheiten

Ein möglicher Angriff auf das zu entwickelnde GM-Protokoll besteht darin, durch längeranhaltende Belastung einer oder mehrerer Instanzen, diese daran zu hindern, die "Heart-Beat"-Nachrichten in den erforderlichen Zeitintervallen zu senden.

Der Angriff bewirkt, daß auf den betroffenen Instanzen mehrere aufeinanderfolgende "performance"- oder "omission"-Fehler auftreten, die zwei unmittelbar sicherheitsrelevante Konsequenzen nach sich ziehen können:

1. Durch die hohe Dauerbelastung bei der Protokollverarbeitung erleidet eine Instanz mehrere aufeinanderfolgende "performance"-Fehler. Sie ist nicht in der Lage, schnell genug eigene "Heart-Beat"-Nachrichten zu senden, so daß die anderen Instanzen fälschlicherweise von einem Ausfall dieser Instanz ausgehen.

2. Durch die hohe Dauerbelastung bei der Protokollverarbeitung erleidet eine Instanz mehrere aufeinanderfolgende "omission"-Fehler. Eintreffende HB-Nachrichten anderer Instanzen werden aufgrund überlaufender Nachrichtenpuffer verworfen. Die lokale Instanz geht folglich fälschlicherweise von einem Ausfall der anderen Instanzen aus.

Ein Angreifer mit Kenntnissen über den eingesetzten Verteilungsalgorithmus kann Dateneinheiten gerade so generieren, daß alle Dateneinheiten von einer Instanz selektiert

werden. Dadurch ließe sich gezielt eine Instanz angreifen, die dann ggf. die oben genannten "performance"-Fehler erleidet. Die übrigen Instanzen sind durch diesen Angriff kaum betroffen, da sie die Dateneinheiten des Angreifers nicht selektieren. Möglicherweise rekonfigurieren sie sich, da sie von einem Ausfall der anderen Instanz ausgehen. Der DoS-Angriff ist dann erfolgreich. Es wäre dem Angreifer gelungen, die Leistung des Systems deutlich zu beeinträchtigen. Durch Anpassen der Dateneinheiten können daraufhin weitere Instanzen angegriffen werden.

Der zweite Fall ist ebenfalls kritisch, wenn die lokale Instanz durch die "omission"-Fehler von einem Fehler einer oder mehrerer anderer Instanzen ausgeht und sich schließlich rekonfiguriert. Auch in diesem Fall kommt es zu Paketverlusten. Die Dateneinheiten, die von der Instanz nach dem ursprünglichen Verteilungsschlüssel selektiert worden wären, werden nach der Rekonfigurierung von keiner Instanz selektiert und folglich verworfen. Gleichzeitig kommt es aber auch zu Paketverdoppelungen, da die fälschlicherweise rekonfigurierte Instanz nun zusätzlich Dateneinheiten verarbeitet, die bereits von anderen Instanzen verarbeitet werden. Auch in diesem Fall hat der Angreifer Erfolg, da die Leistung des Systems beeinträchtigt wird.

8.5.4.1 "Denial of Service"-Angriff: Erfolgsaussichten und Auswirkungen

Im schlimmsten Fall ist ein Angreifer in der Lage, die verfügbare Übertragungskapazität vollständig durch Senden eigener Dateneinheiten aufzubrauchen. Der Angreifer kann also unabhängig von der Paketlänge immer genügend Dateneinheiten generieren, so daß eine Instanz durch die eintreffenden Dateneinheiten über einen Zeitraum von mindestens $(HB_{maxlost}+1)*HB_{intervall}$ Sekunden voll belastet ist (d.h. die Auslastung U in Formel 4.13 ist in diesem Zeitraum konstant bei 1). Der Angreifer generiert hierbei, wie oben dargestellt, aufgrund der Kenntnis des Selektionsalgorithmus die Dateneinheiten derart, daß sie alle von der angegriffenen Instanz selektiert werden müssen.

Da durch den Angriff eine der n Instanzen alle Dateneinheiten des Angreifers selektiert, wird die angegriffene Instanz nach Formel 4.13 $Pmax_i(1)$ Dateneinheiten selektieren und zwar unabhängig von der tatsächlich vorhandenen Anzahl (n) der Instanzen.

Sei $Pmax_M$ die maximale Anzahl an Dateneinheiten, die pro Sekunde auf dem jeweiligen Übertragungsmedium versendet werden können, dann kann die angegriffene Instanz während des DoS-Angriffs im Mittel

$$\frac{Pmax_i(1)}{Pmax_M} \tag{8.13}$$

Prozent der eintreffenden Dateneinheiten bearbeiten. Unabhängig vom Ursprung[16] der Nachricht ist somit während des Angriffs die Wahrscheinlichkeit, daß die Instanz eine beliebige von ihr zu selektierende Nachricht nicht bearbeitet:

$$\mathcal{P}(\text{Instanz verliert PDU}) = 1 - \frac{Pmax_i(1)}{Pmax_M} \tag{8.14}$$

Mit anderen Worten, Formel 8.14 gibt auch die Wahrscheinlichkeit an, daß eine HB-Nachricht einer anderen Instanz verloren wird.[17] Die Wahrscheinlichkeit, daß alle $HB_{maxlost}+1$ *aufeinanderfolgenden HB-Nachrichten* verloren werden, ist unter Voraussetzung von unabhängigen Paketverlusten, in Anlehnung an Fomel 8.14:

$$\mathcal{P}(\text{Verlieren von } HB_{maxlost}+1 \text{ HB-PDUs}) = \left(1 - \frac{Pmax_i(1)}{Pmax_M}\right)^{HB_{maxlost}+1} \tag{8.15}$$

Formel 8.15 gibt somit die Wahrscheinlichkeit an, daß ein DoS-Angriff erfolgreich ist und eine angegriffene Instanz nach $(HB_{maxlost}+1) * HB_{intervall})$ Sekunden die Rekonfiguration durchführt. Diese Wahrscheinlichkeit kann man somit auch als $\mathcal{P}(DoSsuccess)$ bezeichnen, die Erfolgswahrscheinlichkeit für einen DoS-Angriff.

Offensichtlich ist die Wahrscheinlichkeit für dieses Ereignis von mehreren Faktoren abhängig. Je kleiner die Anzahl der tolerierbaren Nachrichtenverluste an HB-PDUs ist und je geringer der maximale Durchsatz einer einzelnen Instanz ist, desto höher ist die Wahrscheinlichkeit, daß der Angriff erfolgreich ist. Erfreulich ist allerdings, daß man unabhängig von der Leistungsfähigkeit einer einzelnen Instanz und somit unabhängig von dem Quotienten in Formel 8.13 die Wahrscheinlichkeit für einen erfolgreichen Angriff beliebig klein $(< \epsilon)$ halten kann. Dies erreicht man durch Erhöhen der tolerierbaren HB-Verluste, so daß gilt:

$$\forall \epsilon > 0, \forall Pmax_M, \forall Pmax_i(1), \exists HB_{maxlost} : \mathcal{P}(DoSsuccess) < \epsilon \tag{8.16}$$

Die Interpretation von Formel 8.16 ist: Unabhängig von der Potenz des Angreifers $(Pmax_M)$ und der Leistungsfähigkeit einer Instanz $(Pmax_i(1))$ kann durch die Wahl der tolerierbaren aufeinanderfolgenden HB-Verluste die Wahrscheinlichkeit für einen *fälschlicherweise vermuteten Ausfall* einer anderen Instanz beliebig klein $(< \epsilon)$ gehalten werden.

Es gibt darüber hinaus einige beachtenswerte Aspekte zur Wahl der Länge des HB-Intervalls $(HB_{intervall})$:

- Ein großes $HB_{intervall}$ ist günstig, um den Angriff automatisch oder eventuell manuell erkennen zu können. Durch ein großes $HB_{intervall}$ wird der Angreifer gezwungen, lange anzugreifen, um erfolgreich sein zu können.

[16] Angreifer, normaler Benutzer oder Partnerinstanz

[17] Hierbei wird angenommen, daß die Bearbeitung einer HB-Nachricht in etwa genauso aufwendig ist, wie die Bearbeitung einer zu filternden Nachricht.

- Ein großes $HB_{intervall}$ ist auch günstig, um die Belastung der Instanzen durch die Ausfallerkennung gering zu halten, da seltener eigene HB-Nachrichten gesendet werden müssen und gleichzeitig auch weniger HB-Nachrichten anderer Instanzen zu verarbeiten sind als bei kleinerer Intervalllänge.
- Ein großes $HB_{intervall}$ ist andererseits ungünstig, da die Latenzzeit zwischen Auftreten und Erkennen eines tatsächlichen Ausfalls größer wird. Dies trifft insbesondere dann zu, wenn ϵ in Formel 8.16 klein gewählt wird und somit $HB_{maxlost}$ größer gewählt werden muß.

Es scheint, daß die Parameter $HB_{intervall}$ und $HB_{maxlost}$ nicht für alle Konfigurationen gleich gewählt werden können, sondern unter anderem auch abhängig von der Leistungsfähigkeit der einzelnen Instanzen gewählt werden müssen.

8.5.4.2 Worst-Case Szenario

Der schlimmste Fall tritt ein, wenn lediglich zwei Instanzen aktiv sind und eine von beiden, wie oben beschrieben, angegriffen wird. Werden alle $HB_{maxlost}$ aufeinanderfolgenden HB-Nachrichten von der angegriffenen Instanz verloren, dann sieht sich die angegriffene Instanz als alleinige Instanz. Sie rekonfiguriert sich und filtert zukünftig alle eintreffenden Dateneinheiten.

Die andere Instanz empfängt ggf. noch HB-Nachrichten von der zweiten Instanz und behält ihre Konfiguration bei. In diesem Fall hat der Angreifer erreicht, daß die Leistungsfähigkeit der parallelen Paket Screen von $Pmax_n(2)$ auf $Pmax_n(1)$ zurück geht (vgl. Formel 4.14) und zusätzlich bis zu 50% aller Dateneinheiten verdoppelt werden[18], da der Verteilungsschlüssel inkonsistent auf den zwei Instanzen ist.

Dieses Szenario verdeutlicht, wie gefährlich ein DoS-Angriff sein kann. Es zeigt aber auch, daß die einzelnen Instanzen in jedem Fall inkonsistente Verteilungsschlüssel erkennen müssen, um anschließend wieder einen stabilen Zustand herbeizuführen. Beispielsweise müßten die beiden Instanzen in diesem "Worst-Case"-Szenario in der Lage sein, aufgrund ihrer HB-Signale irgendwann einmal zu erkennen, daß mehr als eine Instanz aktiv ist. Durch eine anschließende, automatische eingeleitete Rekonfiguration könnte sich die angegriffene Instanz sozusagen wieder von dem Angriff erholen (vgl. "recovery", Abschnitt 2.2.3).

In den beiden folgenden Abbildungen wird die Wahrscheinlichkeit für den Erfolg eines solchen Angriffs einmal für den "worst case" dargestellt (vgl. Abbildung 8.5) und einmal für den etwas realistischeren Fall eines "nur starken" Angreifers (vgl. Abbildung 8.6).

[18] Bis zu 50% (bezogen auf den ausgangsseitigen Durchsatz der Packet Screen) der Pakete werden verdoppelt, so lange die tatsächliche Last in Paketen pro Sekunde kleiner oder gleich $Pmax_n(1)$ ist. Unter dieser Last sind beide Instanzen in der Lage, alle Dateneinheiten zu verarbeiten. Die angegriffene Instanz versucht dies auch aufgrund der Annahme, sie sei alleine. Die zweite Instanz selektiert nach wie vor jede zweite Dateneinheit und verarbeitet sie. Abhängig von den konfigurierten Filterregeln werden somit bis zu 50% aller Dateneinheiten verdoppelt. Rekonfiguriert sich die zweite Instanz ebenfalls, dann würden sogar alle Dateneinheiten (100%) verdoppelt.

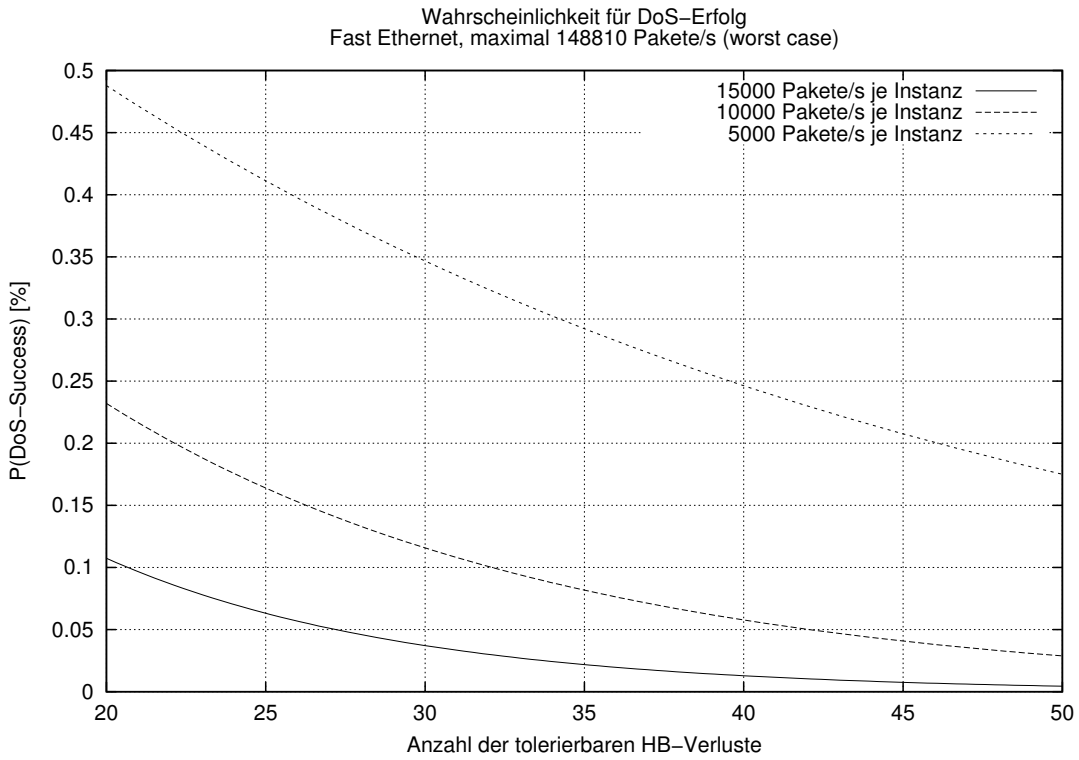


Abbildung 8.5: IP-basierter DoS-Angriff: Erfolgswahrscheinlichkeit ('worst case')

8.5.5 "Denial of Service"-Angriff auf das Gruppenmanagement-Protokoll

Ein weiterer DoS-Angriff mit denselben Auswirkungen, wie in Abschnitt 8.5.4 diskutiert, ergibt sich durch Angriff auf das Gruppenmanagement-Protokoll. Bei diesem Angriff versucht der Angreifer, durch Senden von Nachrichten an den SAP des GMPs die Instanzen zu überlasten.

Dieser Angriff unterscheidet sich qualitativ von dem Angriff mit IP-Dateneinheiten, da hier das GMP für den Angriff funktionalisiert wird:

- Der Angreifer kann durch Wiedereinspielen aufgezeichneter HB-Dateneinheiten versuchen, eine Rekonfiguration direkt zu erreichen. Das Wiedereinspielen von bereits gesendeten HB-Dateneinheiten darf daher nicht zu einer Störung der Prozessor-Gruppe führen.
- Der Angreifer kann durch Senden beliebiger Dateneinheiten an den SAP des GMPs versuchen, die Protokollverarbeitung in den Instanzen zu behindern. Die Protokollverarbeitung muß daher mit unbekannten Dateneinheiten, zu kurzen, zu langen und auch mit semantisch nicht angemessenen Dateneinheiten umgehen können.

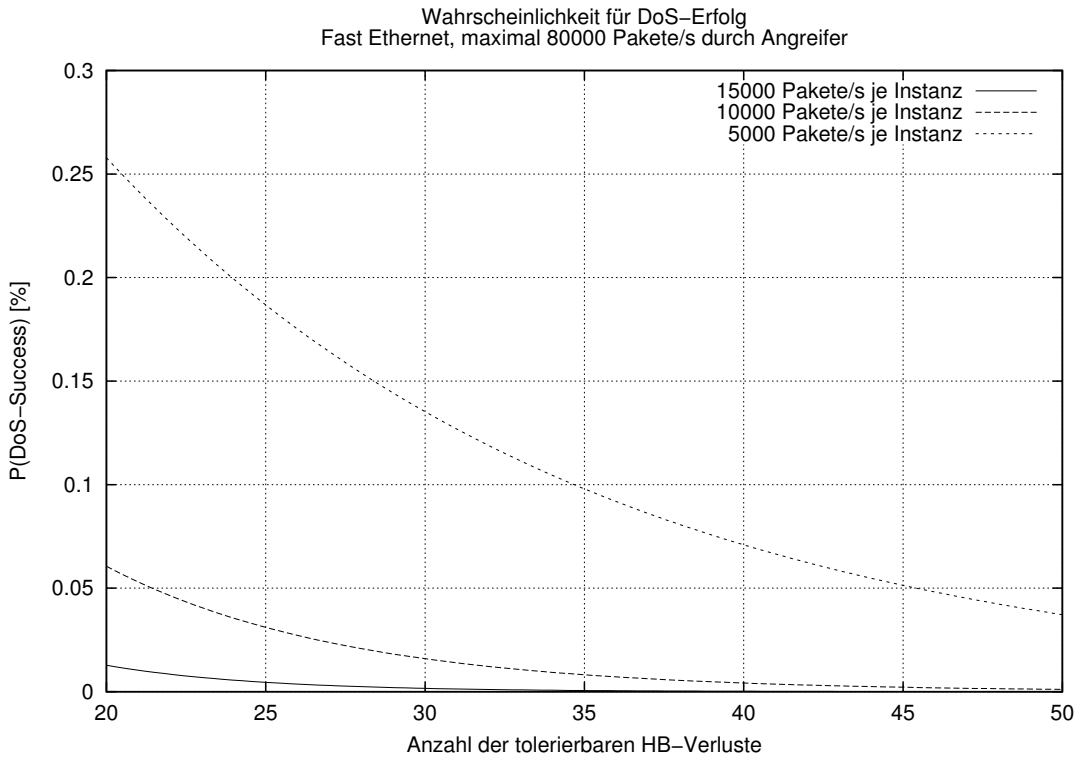


Abbildung 8.6: IP-basierter DoS-Angriff: Erfolgswahrscheinlichkeit (starker Angreifer)

- Ein Spezialfall der beiden oben genannten Angriffsformen besteht darin, empfangene Dateneinheiten so zu manipulieren, daß sie nicht offensichtlich als wiedereingespielte Dateneinheiten zu erkennen sind. Der Angreifer kann dies beispielsweise bei Kenntnis des GMPs durch Anpassen der PDUs versuchen. In diesem Fall müssen die Instanzen die Dateneinheiten auswerten, und das bedeutet, sie müssen die relativ aufwendigen kryptographischen Berechnungen zur Integritätskontrolle durchführen. Der Angreifer kann also durch massives Senden dieser speziell manipulierten Dateneinheiten versuchen, einen DoS durch die aufwendigen Hash-Verfahren zu erreichen.

Durch relativ einfache Mechanismen (s.o.) lassen sich "Replay"-Versuche aufdecken. Eine robuste Implementierung des GMPs ist ebenfalls relativ aufwandsarm, so daß einfache Angriffe basierend auf syntaktisch falsch gebildeten PDUs wirkungslos bleiben. Das letzte Beispiel ist das gefährlichste Angriffsszenario. Der kryptographische Hash-Wert wird benötigt, um sicherzustellen, daß der Angreifer nicht in der Lage ist, unerkannt Dateneinheiten zu fälschen und in die Kommunikation zwischen den Instanzen einzuspielen. Dieser Mechanismus kann aber vom Angreifer wiederum gegen das GMP ausgenutzt werden. Ein solcher Angriff kann nicht verhindert werden. Gegen diese Art von Angriffen sind alle Protokolle empfindlich, die mit Hilfe von aufwendigen (kryptographischen) Verfahren versuchen, die Kommunikationsbeziehungen zu sichern. Insbesondere sind hier zu nennen [12, 65, 106].

8.5.5.1 Erkennen des "Denial of Service"-Angriffs

Der Angriff kann genau wie der oben beschriebene Angriff nicht verhindert werden. Im Gegensatz zum in Anschnitt 8.5.4 dargestellten Angriff, kann dieser Angriff jedoch *dezentral* durch die betroffene GMP-Instanz selbst erkannt werden. Jede einzelne Instanz ist in der Lage, einen derartigen DoS-Angriff als solchen zu klassifizieren.

Diese Erkennungsmöglichkeit ist genau wie der zu erkennende Angriff protokollspezifisch. Aus der Kenntnis des Timings bei der HB-Kommunikation und der Syntax der HB-PDUs kann jede einzelne Instanz erkennen, ob sie gerade angegriffen wird.

Geht man von einem stabilen System mit n Instanzen aus, dann erwartet jede einzelne Instanz pro $HB_{intervall}$ im Mittel $n-1$ HB-Nachrichten von den anderen Instanzen. Wenn nun für einen bestimmten Zeitraum eine beliebige Instanz HB-Dateneinheiten empfängt, so daß gilt:

$$\sum_{t_0}^{t_1} \text{HB-PDU} \gg \frac{(n-1) * (t_1 - t_0)}{HB_{intervall}} \quad , \tag{8.17}$$

dann kann von einem Angriff ausgegangen werden. Da das System zuvor stabil lief, kann ein Konfigurationsfehler ausgeschlossen werden. Das GMP bzw. jedes Gruppenmitglied kennt zu jeder Zeit die Anzahl der Teilnehmer (hier n) und neu hinzukommende Gruppenmitglieder hätten sich sukzessive anmelden müssen. Es bleibt lediglich ein Angriff als mögliche Ursache.[19]

Offen in Formel 8.17 ist noch, was *deutlich größer (≫)* bedeutet und wie groß $\Delta t = t_1 - t_0$ sein muß. Der Angreifer will mit seinem DoS-Angriff im Idealfall[20] eine Rekonfiguration erreichen, so daß $\Delta t > (HB_{maxlost} + 1) * HB_{intervall}[sec]$ sein müßte. Die Erkennung des Angriffs wäre allerdings schon innerhalb von $HB_{intervall}$ möglich.

Der Schwellwert, ab dem eine Instanz ein vermehrtes Auftreten von HB-Dateneinheiten als DoS-Angriff deuten sollte, kann sehr niedrig gewählt werden, da die genaue *Anzahl an erwarteten HB-PDUs* zu jeder Zeit bekannt ist. Als einfache Heuristik könnte man

$$\sum_{t_0}^{t_1} \text{HB-PDU} \geq \frac{2(n-1) * (t_1 - t_0)}{HB_{intervall}} \quad , \quad \forall n > 1 \tag{8.18}$$

wählen. Die Wahl ist eigentlich nur von der relativen Zeitverschiebung ("timedrift") der einzelnen Uhren auf den Instanzen abhängig. Bei der doppelten Anzahl an eintreffenden

[19] Während der Entwicklung sind Tests natürlich ebenfalls mögliche Ursache.

[20] DoS-Angriffe sind in jedem Fall störend, da die von ihnen verbrauchten Ressourcen der eigentlichen Auftragsverarbeitung entzogen werden. Der *Idealfall* aus Sicht des Angreifers ist eine Stilllegung des angegriffenen Dienstes. Bei allen Betrachtungen wird jedoch hier davon ausgegangen, daß die "Host"-Sicherheit der Packet Screen und somit die Sicherheit jeder einzelnen Instanz adäquat ist, um direkte Angriffe mit dem Ziel eines "root compromise" abzuwehren. Die Techniken hierfür sind seit langem bekannt [93]. Was dem Angreifer also bleibt, ist spezifische Eigenschaften der parallelen Packet Screen auszunutzen, beispielsweise die Fähigkeit zur Rekonfiguration.

Dateneinheiten aufgrund einer massiven Zeitverschiebung der lokalen Uhren sollte über ein größeres $HB_{intervall}$ nachgedacht werden (vgl. Formel 7.8).

Die eben durchgeführte Abschätzung ist ebenfalls als "Worst-Case" für die Erkennung eines Angriffs zu verstehen. Wenn ein stabiles System vorliegt, dann kann bereits eine einzige HB-PDU mit falscher Integritäts-Prüfsumme[21] als Angriff angenommen werden. Der obige "Worst-Case" setzt demnach ein Wiedereinspielen von abgehörten Dateneinheiten mit richtiger Integritäts-Prüfsumme voraus. Die eben vorgestellte Heuristik ist jedoch unabhängig von der Gültigkeit der Integritäts-Prüfsumme. Mit anderen Worten, der Angriff und auch das Ende des Angriffs kann aufwandsarm (s.o.) erkannt werden. Der Angreifer muß hingegen deutlichen Aufwand (Abhören und Wiedereinspielen) betreiben, um obigen Angriff durchzuführen. Dies ist ein enormer Vorteil für den Angegriffenen. Mit ganz ähnlichen Mechanismen wird auch in anderen Protokollen versucht, erst dann Aufwand in die Kommunikation zu investieren, wenn man sich sicher ist, daß es sich um einen legitimen Kommunikationspartner handelt.

8.5.5.2 Abwehren des Angriffs

Wird ein DoS-Angriff auf Grundlage der obigen Abschätzung erkannt, bietet sich eine heuristische Vorgehensweise an. In jedem Fall muß dem Administrator der Angriffsversuch gemeldet werden, um weitere Maßnahmen einleiten zu können. Darüber hinaus könnte die lokale Instanz der Prozessor-Gruppenverwaltung die folgende Heuristik anwenden, um dem DoS-Angriff entgegenzuwirken:

- Ignorieren unerwarteter Nachrichten: Die Instanz weiß, wer in der Prozessor-Gruppe ist. Nachrichten von anderen potentiellen Instanzen können einfach ignoriert werden. In einer lokalen Datei kann eine Liste der gültigen Gruppennamen gespeichert werden. Nachrichten können unmittelbar verworfen werden, wenn sie nicht von einem Prozessor mit gültigem Namen versendet wurden.[22]

  Für den Angreifer bedeutet dies, er muß einen richtigen Namen wählen, um diese erste Hürde zu überwinden. Es ist implementationsspezifisch, wie leicht ein solcher Name feststellbar ist. Gültige Namen werden von den Gruppenmitgliedern in den jeweiligen Nachrichten versendet und können je nach verwendetem Übertragungsmedium mit geringem oder aber nur mit beträchtlichem Aufwand abgehört werden.

[21] Bei dem hier zu entwickelnden leicht-gewichtigen Protokoll für lokale Workstation-Cluster ist der Fall, daß eine Integritäts-Prüfsumme aufgrund eines Übertragungsfehlers falsch ist, vernachlässigbar. Bei allen bekannten Schicht-2 Protokollen wird über eine Paketprüfsumme sicher gestellt, daß Dateneinheiten mit Übertragungsfehlern die Anwendung gar nicht erst erreichen.

[22] Diese Vorgehensweise hat den Nachteil, daß neue Instanzen nicht einfach dynamisch zur Gruppe hinzugefügt werden können. Es müßten zuvor erst ihre Adressen den anderen Instanzen bekanntgegeben werden, damit diese die Nachrichten neuer Herkunft nicht als Angriff deuten.

- Verbessertes Ignorieren unerwarteter Nachrichten: Die Instanz weiß darüber hinaus, welchen Inhalt die HB-Nachrichten der Partnerinstanzen haben sollten. Nachrichten, die von den Erwartungen abweichende Informationen (Anzahl und Adressen der aktiven Gruppenmitglieder) beinhalten, können einfach ignoriert werden, so lange Nachrichten mit erwarteten Informationen im selben $HB_{intervall}$ eintreffen.

  Dieser Mechanismus erfordert eine Änderung der Interpretationsreihenfolge eintreffender HB-Nachrichten. Anstatt zuerst die aufwendige Integritätskontrolle durchzuführen, wird mit einer *deutlich aufwandsärmeren Plausibilitätskontrolle* der Aufwand für das Behandeln plumper DoS-Angriffe minimiert.

  Sollte der Angreifer geschickt genug sein, um Dateneinheiten mit der tatsächlichen Anzahl von Gruppenmitgliedern und deren Identität zu generieren, dann würde die Plausibilitätskontrolle keine Vorteile bringen. In diesem Fall kann das Gruppenmanagement-Protokoll mit einer einfachen Heuristik aufwandsarm weiterarbeiten. HB-Dateneinheiten mit plausiblem Inhalt werden ohne Validierung der Integritätsprüfsumme bearbeitet. Die Instanz überprüft erst nach Ende des DoS-Angriffs oder sporadisch zwischendurch, wenn keine plausiblen HB-Nachrichten mehr empfangen werden, die Integritätsprüfsumme.

- Agressives Ignorieren aller Nachrichten: Mit dieser Methode kann einem DoS am effektivsten entgegengewirkt werden. Die Instanz vernichtet sofort alle eintreffenden Dateneinheiten und aktualisiert lediglich einen Zähler, der die Summe der pro $HB_{intervall}$ eingetroffenen Dateneinheiten angibt. Ist dieser Wert größer als ein (von der aktuellen Anzahl der Instanzen abhängiger) Schwellwert, vernichtet die Instanz für die Dauer eines weiteren $HB_{intervall}$ alle Dateneinheiten. Ist die Summe kleiner oder gleich diesem Schwellwert, dann wird der Angriff als beendet angesehen und die normale Diensterbringung wird fortgesetzt.

Die oben genannte Heuristik ist nur deshalb anwendbar, da die Praxis zeigt, daß DoS-Angriffe sehr häufig, dafür aber relativ kurzzeitig, auftreten. DoS-Angriffsversuche wurden auf Firewall-Installationen des Autors wöchentlich, zeitweise sogar täglich beobachtet. Die Wahrscheinlichkeit für einen DoS-Angriff ist also sehr groß im Vergleich zur Ausfallswahrscheinlichkeit einer Instanz. Die Interpretation der obigen Heuristik ist demnach folgendermaßen:

So lange ein DoS-Angriff abläuft, darf die Ausfallerkennung annehmen, daß keine Instanz ausfällt.

Bei dieser Heuristik ist zu beachten, daß die auftretenden DoS-Angriffe in der Regel im Bereich von Minuten bis wenigen Stunden liegen. Ein über Stunden geführter DoS-Angriff ist erkennbar und durch andere Gegenmaßnahmen abzustellen (beispielsweise vorübergehende Netzabtrennung oder Filterung von Dateneinheiten bereits auf "Backbone"-Routern). Bei einem nur über wenige Minuten geführten Angriff ist die Wahrscheinlichkeit für einen tat-

sächlichen, zeitgleichen Ausfall einer oder gar mehrerer Instanzen vernachlässigbar klein.[23] Die Heuristik beruht letztendlich auf der Annahme, daß während eines Angriffs keine Instanz ausfällt.

In jedem Fall handelt es sich hierbei um eine Einschränkung des Dienstes (degradation). Die Einschränkung besteht darin, daß während eines DoS-Angriffs die Gruppenverwaltung vorübergehend ausgesetzt wird. Dadurch wird es dem Angreifer erschwert, das eigentliche Angriffsziel zu erreichen, nämlich die Gruppenverwaltung in einen inkonsistenten Zustand zu überführen.

Es bleibt ein Restrisiko, nämlich der Ausfall einer Komponente während eines Angriffs. In diesem Fall wird die Herstellung der neuen Gruppenzusammensetzung verzögert, bis der Angriff vorüber ist. Danach erholt sich das System innerhalb einer Zeitspanne von $(HB_{maxlost}+3)*HB_{intervall}[sec]$ (vgl. Abschnitt 7.2.5.1), indem wieder Konsens hergestellt wird.

Darüber hinaus muß auch betrachtet werden, inwieweit sich die Sicherheit der parallelen Installation mit eingesetztem, sicheren Gruppenmanagement von der ursprünglichen Sicherheit einer sequentiellen Installation unterscheidet:

- Durch das einzusetzende Protokoll ist eine neue Angriffsmöglichkeit gegeben. Die eingesetzten Sicherheitsmechanismen reichen aus, um einfache Angriffe leicht abzuwehren. Es besteht ein Restrisiko, daß durch einen geschickten Angreifer (möglich nur durch Insider) der in Abschnitt 8.5.5 diskutierte Angriff glückt und vorübergehend die Gruppenverwaltung ausgesetzt wird. Da dieser Angriff aber dezentral erkannt und gemeldet wird, ist der mögliche Schaden gering. Der Erkenntnisgewinn (recognition) ist hingegen groß: es muß einen Angreifer mit Zugang zum lokalen Schicht-2 Segment geben.

- Der in Abschnitt 8.5.4 diskutierte Angriff ist über IP möglich, kann also von überall her erfolgen. Durch die entwickelten Gegenmaßnahmen kann die Erfolgswahrscheinlichkeit für diesen Angriff jedoch beliebig klein gehalten werden. Darüber hinaus kann der Angriff bei mehr als einer Instanz nie so verheerend sein, wie bei einer sequentiellen Lösung. Im schlimmsten Fall gelingt es dem Angreifer, kurzzeitig den Gruppenkonsens zu stören, die Diensterbringung durch die anderen Instanzen bleibt jedoch weiter möglich. Eine einzelne Instanz ohne Gruppenmanagement wäre hingegen in jedem Fall dem DoS voll ausgesetzt. Die Verfügbarkeit ist somit durch das GMP trotz der neuen Risiken erhöht, insbesondere weil nach dem DoS eine automatische, schnelle Wiederherstellung einsetzt (“recovery”).

[23] Der Ausfall der vom Autor im Produktionsnetz betriebenen Firewall-Hardware konnte während der vierjährigen Tätigkeit nur einmal beobachtet werden. Ein als Packet Screen konfigurierter Rechner konnte nach einer vorübergehenden Stromabschaltung nicht wieder gestartet werden. Diese Beobachtung ist zwar nicht repräsentativ, dennoch kann durchaus behauptet werden, daß der Ausfall einer Instanz im Vergleich zu den tatsächlich auftretenden DoS-Versuchen extrem selten ist.

- Die Sicherheit kann allerdings durch Gefahren jenseits des eingesetzten GMPs gefährdet werden. Immerhin muß für den Einsatz ein gemeinsamer Schlüssel konfiguriert werden. Der unvorsichtige Umgang mit diesen Geheimnissen kann möglicherweise das gesamte Gruppenmanagement unterlaufen [9].

Zusammenfassend ergibt sich, daß die neuen Risiken durch das Gruppenmanagement durchaus tragbar sind, wenn man den Gewinn dagegenrechnet. Eine deutlich höhere Verfügbarkeit und Leistungsfähigkeit bei vergleichsweise geringen Restrisiken. Die höhere Verfügbarkeit ermöglicht auch ein besseres Audit ("recognition") und eine automatische Erholung ("recovery").

8.6 Zusammenfassung der zusätzlichen Mechanismen für ein sicheres GM-Protokoll

Im vorigen Abschnitt wurden potentielle Angriffe auf das GMP diskutiert und möglichst leichtgewichtige Gegenmaßnahmen vorgestellt. Hier sollen kurz die zusätzlich erforderlichen Mechanismen zur Absicherung des GMPs zusammengefaßt werden.

8.6.1 "Keyed Message Authentication Code" für Integrität und Authentizität

Es wird von allen Instanzen eine kryptographisch sichere Prüfsumme basierend auf dem MD5-Algorithmus (vgl. [198, 140]) und einem geheimen, gemeinsamen Schlüssel berechnet und als Teil der Dateneinheiten versendet. Dadurch ist die Authentizität der Nachrichten sichergestellt: Da der geheime Schlüssel mit in die Berechnung des "Message Authentication Codes" einbezogen wird[24], kann die Nachricht nur von einem autorisierten (potentiellen) Gruppenmitglied generiert worden sein. Darüber hinaus ist die Integrität der Nachricht vom Empfänger zweifelsfrei verifizierbar, so daß es keine Möglichkeit zum unerkannten Fälschen von Dateneinheiten durch Dritte gibt, die nicht im Besitz des geheimen, gemeinsamen Schlüssels sind.

Es wird eine kryptographisch sichere Prüfsumme anstelle einer symmetrischen Verschlüsselung gewählt, da letztere nur mit deutlich höherem Aufwand möglich wäre. Dies stünde im Widerspruch zur angestrebten Leistungsfähigkeit (vgl. auch Abschnitt 8.3).

8.6.2 Streng monoton steigende Sequenznummer

Die bereits in Abschnitt 7.2 zum Erkennen von Paketverdoppelungen durch Übertragungsfehlern eingeführte, streng monoton steigende Sequenznummer in den Dateneinheiten ist

[24] Man spricht daher auch von einem "Keyed Message Authentication Code" ("Keyed-MAC").

auch aus Sicherheitsgründen hoch interessant. Zusammen mit dem "Keyed-MAC" und der zusätzlichen Authentifikationsanfrage an Bewerber (s.u.) werden so Sicherheitsprobleme durch das Wiedereinspielen von Dateneinheiten beseitigt.

8.6.3 Authentifikationsanfrage an Bewerber

Ein Bewerber muß vor Aufnahme in die Gruppe, eine oder mehrere Authentifikationsanfrage(n) durch die Gruppenmitglieder beantworten. Die Gruppenmitglieder generieren hierfür einen frischen Wert, den der Bewerber in seiner nächsten PDU als Sequenznummer zu verwenden hat. Stimmt der berechnete "Keyed-MAC" dieser erneuten Bewerbung, zeigt der Bewerber, daß er im Besitz des gemeinsamen Geheimnisses ist und somit autorisiert ist, der Gruppe beizutreten.

8.6.4 Alumni-Liste und stabiler Speicher

Für die Generierung von frischen Sequenznummern wird eine Ehemaligen-Liste (Alumni-Liste) regelmäßig auf stabilen Speicher (beispielsweise Datei auf der Festplatte) gespeichert, um die Einhaltung einer streng monoton steigenden Sequenznummer trotz potentieller Ausfälle einzelner Gruppenmitglieder durchzusetzen.

8.6.5 Zusätzlicher Zustand für DoS-Angriffe

Wird ein DoS-Angriff erkannt, meldet die betroffene Instanz dies sowohl in ihren regelmäßigen HB-Nachrichten als auch durch Schreiben eines Audit-Trails. Darüber hinaus wechselt die Instanz in einen zusätzlichen Zustand ("ATTACK-DETECTED"), in dem nur noch eingeschränkt an der Gruppenverwaltung teilgenommen wird, um die Last durch den DoS zu beschränken. Dem Prozessor bleibt dadurch mehr Zeit für die Abarbeitung seiner eigentlichen Aufträge (hier: Filtern von Dateneinheiten). In diesem Zustand versendete HB-Signale beinhalten zusätzliche Informationen, die den Angriff auch den anderen Instanzen mitteilen ("HB*"). Durch periodisches Validieren des Nachrichtenaufkommens wird das Ende des DoS erkannt, um in den normalen Betriebszustand zu wechseln, aus dem heraus wieder Konsens hergestellt wird. In Abbildung 8.7 ist der neue Protokollautomat mit dem zusätzlichen Zustand noch einmal graphisch dargestellt.[25] Es gibt die beiden zusätzlichen, internen Signale "attd" und "eofattd" (Angriff und Ende des Angriffs erkannt), die den Protokollautomaten in den neuen Zustand "ATTACK-DETECTED" bzw. von diesem Zustand zurück in den akzeptierenden Zustand "WB" überführen. Je nach Implementation wird im Zustand "ATTACK-DETECTED" eine der oben diskutierten Heuristiken angewendet, um nicht zu viele Betriebsmittel zu verbrauchen. Die beim Prototypen implementierte Heuristik wird im folgenden Kapitel vorgestellt.

[25] In der Abbildung sind die Beschriftungen für Zustandsübergänge durch Benutzerinteraktionen aus Gründen der Übersichtlichkeit nicht eingetragen.

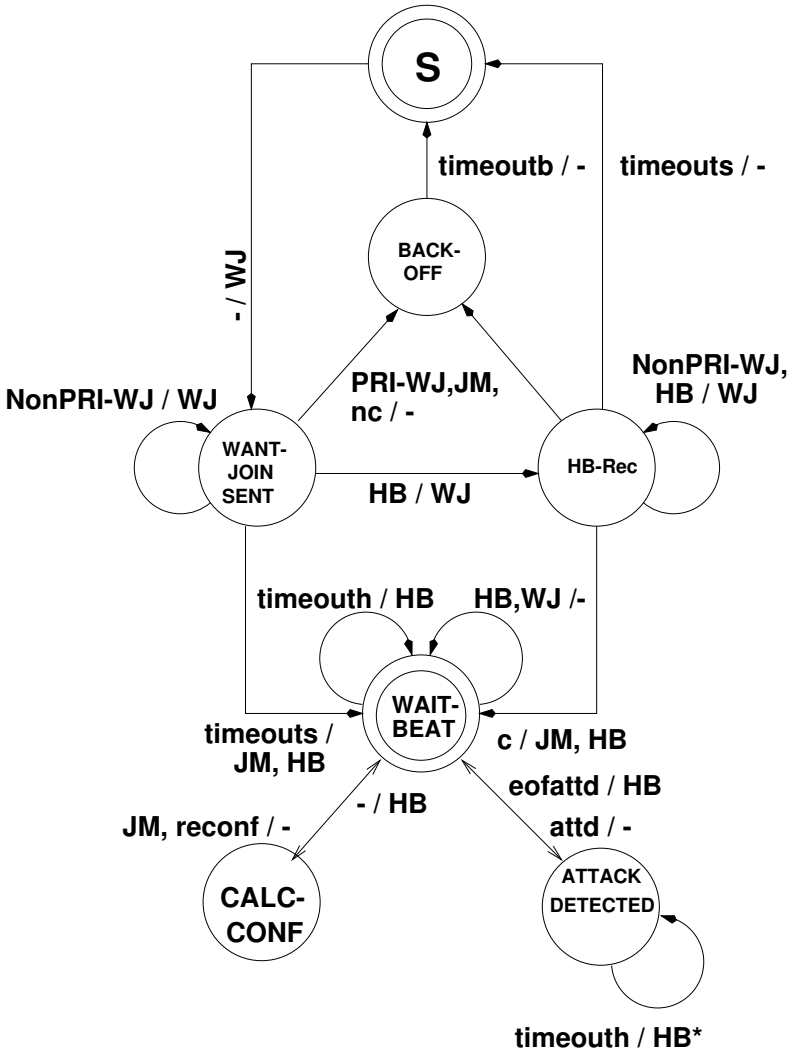


Abbildung 8.7: Zustandsautomat für ein sicheres, leichtgewichtiges GMP

8.7 Zusammenfassung

Es wurde eine Sicherheitsanalyse für das leichtgewichtige GMP durchgeführt, die zu zusätzlichen Sicherungsmechanismen führte. Dadurch wird endlich allen drei Optimierungszielen genügt: Die Leistungsfähigkeit wurde durch Parallelverabeitung gesteigert. Die hierdurch implizierten, potentiellen Verfügbarkeitsprobleme wurden durch ein Gruppenmanagement aufgefangen, das eine höhere Verfügbarkeit der parallelen Packet Screen gegenüber dem Ausgangs-System ermöglicht. Die zusätzlichen Sicherungsmechanismen schützen schließlich vor dem Mißbrauch dieses GMPs durch Dritte.

Hierbei wurden wichtige Teilergebnisse erzielt. Es wurde motiviert, daß durch zusätzliche Entwurfs- und Iterationsschritte eine sukzessive Absicherung der vom Protokoll benutzten

Gruppenmanagement-Mechanismen erfolgen muß. Voraussetzung hierfür ist eine detaillierte, formale Protokollspezifikation und die Formulierung von Sicherungszielen, die eine detaillierte Sicherheitsanalyse erst ermöglichen. Nur durch die genaue Spezifikation wurde es möglich, protokollspezifische Angriffe auf die Sicherungsziele zu diskutieren. Die daraus entwickelten Sicherungsmechanismen wurden in weiteren Iterationen ihrerseits analysiert.

So stellte sich heraus, daß die für die Authentifikation wichtigen, kryptographisch sicheren, "Message Digest"-Prüfsummen als Ausgangspunkt für "Denial of Service"-Angriffe benutzt werden können. Auf dieser Grundlage wurden weitere Mechanismen entwickelt, die den DoS-Angriff abmildern. Das GMP wird für die Dauer des Angriffs suspendiert, so daß die Betriebmittel dem eigentlichen Dienst (dem Screening) zur Verfügung stehen. Dies entspricht der geforderten Vorgehensweise einer Angriffserkennung mit einer anschließenden, gezielten "degradation" der Dienste eines überlebensfähigen Systems.

Als weiteres, wichtiges Teilergebnis konnte gezeigt werden, daß viele bekannte Methoden zur Analyse der Sicherheit in verteilten Systemen allein genommen unvollständig sind: Es sind immer nur Aussagen über einen Teil der generischen Sicherungsziele (Authentizität, Integrität, Vertraulichkeit und Verfügbarkeit) möglich. Daher wurde gezeigt, wie diese Verfahren miteinander kombiniert und in den normalen Systementwurf mit SDL/MSC integriert werden können (vgl. Anhang A). Diese Integration ist nicht nur wichtig, um Aussagen zu allen Schutzzielen machen zu können, sondern verhindert auch einen Bruch in der Darstellung zwischen Systementwurf und Sicherheitsanalyse. Durch die Integration in standardisierte, formale Beschreibungstechniken wie SDL/MSC, werden die Sicherheitsanalysen und die zusätzlichen Sicherungsmechanismen bei der Protokollsynthese in einer akzeptierten und gut verständlichen Darstellung den Entwicklern zugänglich gemacht.

Kapitel 9

Implementationsaspekte und Test des Gruppenmanagement-Protokolls

Nach der Spezifikation wird nun eine prototypische Implementation vorgestellt. Ziel der Implementation ist es, ein Gruppenmanagement-Protokoll zur Verfügung zu stellen, das wiederum auf seine Leistungsfähigkeit hin beurteilt werden kann. Die Leistungsfähigkeit hat hier mehrere Ausprägungen. Die in der Spezifikation gestellten Anforderungen müssen umgesetzt werden und die Hauptfunktionalität, nämlich die Ausfallerkennung muß gewährleistet werden. Anschließend muß aber auch die Gesamtleistungsfähigkeit des verteilten Systems, der nun ausfallsicheren parallelen Packet Screen, bewertet werden. Die entscheidenden Fragen sind: Welchen Einfluß hat das zusätzliche Fehlermanagement auf den Durchsatz? und: Ist die parallele Packet Screen noch genauso sicher wie eine herkömmliche Packet Screen?

Bei der Beschreibung der Implementation muß auf sehr systemspezifische Aspekte eingegangen werden. Häufig wird dies nicht gemacht und als "Implementationsdetail" vernachlässigt. Während der Diskussion wird gezeigt werden, daß eine sichere Implementation gerade diese Aspekte nicht vernachlässigen darf, da andernfalls neue Sicherheitslücken entstehen können, die zuvor weder bei der abstrakten Spezifikation, der modellhaften Validation oder der manuellen (Theorem-basierten) deduktiven Validation auffallen konnten.

Aber nicht nur aus Gründen der Sicherheit, sondern insbesondere auch aus Gründen der Effizienz müssen diverse systemspezifische Aspekte beachtet werden. Die sorgfältige Implementation trägt auch maßgeblich zum Erreichen der Sicherheit und somit der Überlebensfähigkeit bei.

In Abschnitt 9.1 werden die verwendeten Komponenten (Hardware und Software) beschrieben. Es folgt eine — für die Überlebensfähigkeit von langlaufenden Diensten sehr wichtige — Diskussion der Maßnahmen für einen geringen Speicherverbrauch in Abschnitt 9.2. Die bislang aufgeschobene Diskussion, wie einzelne Instanzen eindeutig zu identifizieren sind, wird in Abschnitt 9.3 als Implementationsaspekt dargestellt, gefolgt von der Diskussion, wie eintreffende PDUs analysiert werden können (Abschnitt 9.4). Die Implementierung

der bei zeitgesteuerten, asynchronen Systemen essentiellen Zeitgeber wird in Abschnitt 9.5 dargestellt. Ein Verfahren zum Testen des Timings wird in Abschnitt 9.6 beschrieben und schließlich wird auf einige Aspekte der Rechnersicherheit eingegangen, die bei der Dienstimplementation allgemein beachtet werden sollten (Abschnitt 9.7).

9.1 Eingesetze Hardware und Software

Für Implementation wurden UNIX-Workstations unter dem Betriebssystem Solaris 2.6 eingesetzt. Die wesentlichen Merkmale diese Betriebssystems in Hinblick auf die Implementation sind:

- System V basierte Kernelschnittstelle (vgl. [229])
- STREAM-basiertes Kommunikationssubsystem (vgl. [224])
- Anwendungsschnittstelle zum Betriebssystem in der Programmiersprache C (vgl. [135]), Konformität zu IEEE Std 1003.1 und IEEE Std 1003.2 (POSIX 1 und POSIX 2, vgl. [152]) mit folgenden wichtigen Ergänzungen: POSIX.1b (Erweiterungen für Echtzeitverarbeitung), POSIX.1c (Erweiterungen für leichtgewichtige Prozesse).
- STREAM-basierte "Data Link Provider Interface (DLPI)"-Schnittstelle zum Kommunikationssubsystem (vgl. [228])
- Verfügbarkeit des Quellcodes des Betriebssystems

Der Grund für die Wahl des Betriebssystems Solaris liegt in der jahrelangen Erfahrung des Autors mit diesen Systemen, in der Unterstützung der DLPI-Schnittstelle und der Verfügbarkeit des Quellcodes. Die große Erfahrung mit einem bestimmten System ist für die Erstellung eines überlebensfähigen Systems *nicht* erforderlich (aber hilfreich). Derartige Systeme werden von einem Team erstellt, das in der Lage sein muß, sich in diese Details einzuarbeiten. Für die Erstellung eines Prototypen im Rahmen dieser Arbeit ist die vorhandene Erfahrung jedoch ausschlaggebend. Nur so kann ein funktionsfähiger Prototyp in relativ kurzer Zeit erstellt werden.

Die Verfügbarkeit des Quellcodes ist für noch zu diskutierende Anpassungen wichtig. Im UNIX-Bereich trifft diese Eigenschaft noch auf eine Reihe anderer Systeme zu, namentlich LINUX und diverse freie BSD-Derivate (Free BSD, Net BSD, Open BSD). Gegen eines der genannten BSD-Derivate sprach die mangelnde Erfahrung und die mangelnde kommerzielle Unterstützung. Gegen LINUX sprach ebenfalls mangelnde Erfahrung, ein z.Z. noch viel zu schneller Innovationszyklus und die nicht vorhandene DLPI-Schnittstelle.

Unter Solaris 2.6 wurde der herstellerspezifische Compiler verwendet, da nur dieser die Übersetzung von Kernel-Modulen ermöglicht. Zusätzlich wurden spezielle Werkzeuge im Zuge der Tests und zur statischen Sicherheitsüberprüfung verwendet, auf die in den späteren Abschnitten näher eingegangen wird.

9.2 Maßnahmen für geringen Speicherverbrauch

Bei der Diskussion der DoS-Angriffsmöglichkeiten wurde bereits darauf hingewiesen, wie wichtig es ist, daß ein zusätzlicher Dienst, wie das Gruppenmanagement, nicht die Erfolgsaussichten für einen Angriff auf das System erhöht. Dort wurde im wesentlichen argumentiert, daß die zu schützende Ressource die CPU-Zeit ist.

Neben dieser CPU-Zeit gibt es selbstverständlich eine Reihe anderer Betriebsmittel, die zu schützen sind. Bei Prozessen, die besonders langlebig sind (hier die GMP-Instanzen, die auf den Systemen als Prozesse ausgeführt werden), muß insbesondere auch darauf geachtet werden, daß der Speicherverbrauch unter einem akzeptablen Maximalwert bleibt. Es muß vermieden werden, daß der Speicherverbrauch durch Fehler in der Implementation oder durch Fehler im Betriebssystem ständig ansteigt. Dies würde zwangsläufig zu unerwünschtem Systemverhalten führen, beispielsweise zu einem "Deadlock" oder zum Beenden der GMP-Instanz durch das Betriebssystem im Zuge einer Verklemmungsvermeidung.

9.2.1 Erkennen und Vermeiden von Speicherlecks

Besonders kritisch sind in diesem Zusammenhang Programme, die mit dynamischer Speicherverwaltung arbeiten. Ein einziger Fehler bei der manuellen, dynamischen Speicherverwaltung kann zu einem monoton wachsenden "Speicherleck" führen. Es gibt zwei sich gut ergänzende Techniken, die eingesetzt werden sollten, um derartige Probleme zu vermeiden bzw. zu erkennen:

1. Der Programmierer sollte auf die Verwendung von dynamischem Speicher möglichst verzichten. Ist dies nicht möglich, so sollte für jeden in den Quelltext eingefügten Aufruf zum Anfordern von dynamischem Speicher sofort ein entsprechender Aufruf für die Freigabe dieses Speichers eingefügt werden. So kann vermieden werden, daß beim Programmieren angeforderter Speicher "vergessen" wird und somit im Laufe der Zeit ein wachsendes Speicherleck entsteht. Dies ist beispielsweise dann der Fall, wenn die Anforderung des Speichers in einem akzeptierenden Zustand des Systems erfolgt.[1]

2. Es sollten zusätzlich Werkzeuge zur automatischen Analyse des dynamischen Speicherverbrauchs im Zuge der Programmtests verwendet werden. Dies ist erforderlich, da ein Verzicht des Programmierers auf explizite Verwendung von dynamischem Speicher leider nicht hinreichend ist, um die Verwendung von dynamischem Speicher durch den Prozeß zu unterbinden. Es gibt leider eine Vielzahl versteckter Anforderungen an dynamischen Speicher, insbesondere bei der Verwendung von Bibliotheksfunktionen. Werden nun vom Programmierer Bibliotheksfunktionen verwendet,

[1] Da akzeptierende Zustände im System per Definition immer mal wieder durchlaufen werden, wird auch die Speicherallokation ohne entsprechende Freigabe immer wieder ausgeführt.

die ihrerseits Fehler bei der dynamischen Speicherverwaltung beinhalten, kann trotz großer Sorgfalt ein Speicherleck auftreten. Viel wahrscheinlicher ist jedoch, daß der Programmierer nicht erkennt, daß er selbst für die Freigabe des durch die Funktion implizit angeforderten Speichers verantwortlich ist.

Bei dem realisierten Prototypen wurden beide Methoden miteinander kombiniert. Alle Datenstrukturen (z.B. die interne Gruppenliste) sind statisch vereinbart. Zusätzlich wurde ein spezielles Werkzeug[2] zur Überwachung der Freispeicherverwaltung verwendet, um implizit entstandene Speicherlecks erkennen zu können.

Dieser Implementationsaspekt ist grundsätzlich für alle Netzdienste von großer Bedeutung. Typische Netzdienste werden einmal nach dem Starten eines Rechners instantiiert. Danach wird von diesen Diensten – insbesondere in überlebensfähigen Systemen – erwartet, daß sie bis zum Ausfall des Rechners bzw. bis zu einer administrativen Abschaltung durchgängig verfügbar sind. Dies unterscheidet Netzdienste von typischen Anwendungsprogrammen mit Laufzeiten im Sekunden- oder Minutenbereich. Fehler in deren Speicherverwaltung sind ärgerlich, in der Regel jedoch unkritisch, da spätestens beim Beenden dieser Anwendungsprogramme der gesamte Speicher (auch der in den Speicherlecks) wieder an das Betriebssystem zurückgegeben wird.

9.2.2 Zusätzliche Maßnahmen

Weitere Maßnahmen für einen geringen Betriebsmittelverbrauch sind:

- Verzicht auf Programmiersprachen mit aufwendiger Laufzeitumgebung (beispielsweise C++)
- Optimierung bei der Code-Erzeugung durch den Compiler
- Dynamisches Einbinden von Bibliotheksfunktionen
- Optimierung nach der Codeerzeugung durch Reduzieren des ausführbaren Programmcodes um Symboltabelleninformationen und Informationen zur Fehlersuche
- Entfernen des `.comment`-Bereiches aus dem erstellten "Executable and Linking Format, ELF"-Objekt

Alle eben genannten Maßnahmen können durch eine entsprechende Automatisierung im Zuge der Programmgenerierung (vgl. [183, 223]) durch den Programmierer aufwandsarm veranlaßt werden, so daß hierauf nicht näher eingegangen wird.

[2] "Purify" der Firma Rational Software Corporation

9.3 Identifikation der Prozessoren

Ein wesentlicher Aspekt, der im Zuge einer Implementierung des Protokolls aus Kapitel 7 festgelegt werden muß, ist die Frage nach eindeutigen Prozessornamen. Diese eindeutige Benennung wird benötigt, um die totale Ordnung über die Gruppenliste zu berechnen (vgl. Definition 7.1), die ihrerseits wichtig für die Überprüfung des Konsens ist.

Grundsätzlich geeignet ist jede Benennung, die

- eindeutig ist,
- eine einfache und effiziente Implementierung eines Operators zum Überprüfen der $\lessdot$-Relation (vgl. Definition 7.1) ermöglicht und gleichzeitig
- platzsparend ist.

Die beiden ersten Punkte sind unmittelbar einsichtig. Die Benennung der Prozessoren sollte zusätzlich platzsparend sein, da jeder Name als Teil des Gruppenvektors in allen HB-Nachrichten versendet wird. Je kleiner der Name (in Bits) ist, desto mehr Instanzen können vom Protokoll unterstützt werden, bis die maximale PDU-Größe (abhängig vom Übertragungsmedium) erreicht wird.

Für diesen Prototypen wurde als Prozessorname daher die Ethernet-Adresse des Rechners aus dem Segment gewählt, auf dem die Nachrichten des GMPs übertragen werden. Die Ethernet-Adresse ist relativ klein (48-Bits) und muß schon aufgrund der Ethernet-Spezifikation eindeutig sein. Der Operator für die $\lessdot$-Relation kann so einfach und effizient über eine vom Betriebssystem bereitgestellte Vergleichsoperation implementiert werden.

Alternativ hätte auch eine IP-Adresse des Rechners verwendet werden können. Auch sie ist kurz (32-Bits) und muß eindeutig sein. Allerdings würde dadurch die Anwendbarkeit unnötigerweise auf IP-basierte Workstations eingeschränkt werden. Als weitere Alternative könnte der Administrator selbst eindeutige Identifikatoren vergeben. Diese Lösung wäre in Hinblick auf die effiziente Umsetzung der $\lessdot$-Relation am besten, erfordert jedoch mehr manuelle Konfigurierung. Die Ethernet-Adresse kann die Instanz hingegen selbständig erfragen.

9.4 Analyse eintreffender Dateneinheiten

Bei der Analyse der Dateneinheiten wird zweistufig vorgegangen. Eine Funktion `hb_parse()` wertet eintreffende Dateneinheiten aus und liefert im Normalfall die aufbereiteten Informationen aus der PDU (PDU-Typ, Anzahl der Instanzen, Anzahl der Vektoreinträge, Zeiger auf ersten Vektoreintrag, Zeiger auf den Eintrag der sendenden Instanz innerhalb dieses Vektor) an die aufrufende Anwendung zurück. Hierbei werden diverse Typprüfungen durchgeführt, um Verstöße gegen die Syntax (Nicht-Einhaltung des PDU-Formates) zu erkennen. Beispiele für solche Verstöße wären:

- Anzahl der Instanzen im Vektor stimmt nicht mit der Angabe im PDU-Kopf überein,
- ungültige PDU-Typangabe,
- ungültige Gesamtlänge der PDU (Summe aus Kopflänge, Länge des Gruppenvektors und Länge des "keyed MAC").

Dies ist ein sehr wichtiger Implementationsaspekt. Wie bereits im Zuge der Protokollspezifikation angedeutet wurde, findet immer eine Unterspezifikation bezüglich der System-/Umgebungsschnittstelle statt. Das robuste System muß jedoch jenseits einer Spezifikation in der Lage sein, unspezifizierte Eingaben (hier Nachrichten) zu erkennen. Nur so können gravierende Sicherheitsmängel im Protokoll vermieden werden. Ein bekanntes Beispiel, bei dem Verstöße gegen das PDU-Format nicht erkannt wurden und vom Angreifer für einen DoS-Angriff ausgenutzt werden konnten, ist der sog. 'Ping of Death' (vgl. [107]). Hierbei nutzt ein Angreifer fehlende Plausibilitätkontrollen auf Seiten des Empfängers aus, um mit einer überlangen ICMP-Nachricht Teile des Kernels beim Empfänger zu überschreiben. Der empfangende Rechner stürzte ab und war somit bis zu einem Neustart nicht mehr verfügbar. Der Aufwand für diesen Angriff ist minimal (Senden einer einzigen Dateneinheit). Der Schaden war sehr groß: *Verfügbarkeitsverlust aller vom Empfänger angebotenen Dienste.*

In einer zweiten Analysephase kann der "keyed MAC" der PDU validiert werden. Wie bereits in Abschnitt 8.5.5 diskutiert wurde, sollte dies im Zustand "ATTACK-DETECTED" nicht erfolgen, um einen DoS durch zu aufwendige Berechnungen zu vermeiden. Daher ist diese Überprüfung in eine separate Funktion (`hb_checkhmac()`) ausgegliedert, und nicht in die aufwandsarme Plausibilitätskontrolle integriert.

9.5 Implementation der Zeitgeber

Ein weiterer, wesentlicher Aspekt ist die Implementation der Zeitgeber und die mit der Verwendung dieser Zeitgeber einhergehenden Probleme. Das Betriebssystem stellt pro Prozeß bis zu acht unabhängige Zeitgeber zur Verfügung, die mit POSIX4-kompatiblen Funktionen konfiguriert werden können. Somit kann für jedes zu generierende Zeitgeberereignis ("timeouth", "timeoutb", "timeouts") ein individueller Zeitgeber verwendet werden.

Die Zeitgeber lösen asynchron zur normalen Programmabarbeitung ein Unterbrechungs-Signal aus. Dieses Signal wird von der in der Prozeßmaske vom Programmierer angegebenen Signalverarbeitungsfunktion bearbeitet. Nach Beenden der Signalverarbeitungsfunktion wird der Programmablauf an der Stelle der Unterbrechung fortgesetzt.

Hierbei sind zwei Aspekte von besonderer Bedeutung:

1. Es steht während der Signalverarbeitung nur eine sehr eingeschränkte Anzahl an Systemfunktionen zur Verfügung. Diese sog. "wiedereintrittsfesten"-Systemaufrufe können auch durch weitere, asynchron eintreffende Signale unterbrochen werden

und liefern dennoch korrekte Ergebnisse. Der Funktionsumfang dieser “async-safe”-Systemaufrufe ist jedoch sehr begrenzt und reicht in der Regel nicht aus, um alle erforderlichen Berechnungen durchzuführen.

Für diese Implementation wurde daher auf aufwendige Berechnungen innerhalb der Signalverarbeitungsfunktion verzichtet. Es wird lediglich für jeden Zeitgeber ein monoton steigender Zähler erhöht, so daß das Eintreten eines Zeitgeberereignisses während des normalen Programmablaufs beobachtet werden kann.

2. Der normale Programmablauf wird durch die Zeitgeberereignisse unterbrochen. Nach jedem Funktionsaufruf, bei dem zeitgleich ein Zeitgeberereignis eintreten könnte, muß geprüft werden, ob die Funktion vom Zeitgeber unterbrochen wurde oder ordentlich terminierte.

Durch das Überprüfen der Rückgabewerte von Funktionen, die auf den Nachrichtenempfang warten, kann so festgestellt werden, ob diese Funktion durch eine Unterbrechung terminiert wurde. Ist dies der Fall, kann durch Prüfen der oben genannten Zähler für die aufgetretenen Zeitgeberereignisse festgestellt werden, ob ein Zeitgeber ein Signal ausgelöst hat. Davon abhängig kann die weitere Verarbeitung erfolgen.

Zusätzlich muß nach aufwendigen Berechnungen immer geprüft werden, ob sich der Zähler für einen Zeitgeber zwischenzeitlich erhöht hat. Auch in diesem Fall muß auf das entsprechende Zeitgeberereignis reagiert werden. Etwas erleichtert wird die Implementation dadurch, daß die verschiedenen Zeitgeberereignisse laut Spezifikation nicht im selben Zustand auftreten können (vgl. Abbildung 8.7). Eine einfache Implementation kann somit bei Auftritt eines Zeitgeberereignisses aufgrund des aktuellen Zustands auf den jeweiligen Zeitgeber schließen. Robuster wird eine Implementation jedoch, wenn immer der Name des abgelaufenen Zeitgebers zu einem Ereignis festgestellt wird.

9.6 Test des Timings

Im Abschnitt 7.2.5.2 wurde verdeutlicht, wie mit Hilfe der Zeitgeber erreicht wird, daß trotz auftretender Fehler schließlich Konsens im zeitgesteuerten System erreicht wird.

Wichtig für jede Implementation ist eine möglichst genaue Einhaltung dieser “Timing”-Regeln. Als Teil des Tests, bei denen überprüft werden muß, ob sich die Implementation spezifikationsgemäß verhält, steht daher das Testen des Timings im Vordergrund.

Darüber hinaus ist es wichtig, die Zustandsveränderungen des implementierten Protokollautomaten im Zuge der Tests nachzuvollziehen. Es muß geprüft werden, ob die Zustandsübergänge spezifikationsgemäß sind. Schließlich muß als Teil der sogenannten “Conformance Tests” ermittelt werden, ob die “processing rules”, also die Regeln zur Verarbeitung von Signalen in Abhängigkeit vom jeweiligen Zustand, der Spezifikation entsprechen.

Um all diesen Anforderungen an die Testbarkeit der Implementation zu genügen, mußten verschiedene Techniken miteinander kombiniert werden:

- Verwendung von "Tracing"-Techniken zur Verfolgung des Programmablaufs
- Programmierung in Hinblick auf eine Unterstützung der Fehlersuche
- Konsequente Modularisierung

9.6.1 Verwendung von "Tracing"-Techniken

Die Instrumentierung von Quellcodes zum Zweck der dynamischen Programmverifikation kann durch das Einbringen von "Kontrollpunkten" (Trace Probes, TP) in den Quellcode erreicht werden. Wird während der Ausführung des Programms ein TP erreicht, können Aktionen zur Unterstützung der Verifikation ausgeführt werden. Typischerweise werden Meldungen zum gegenwärtigen Verarbeitungszustand ausgegeben.

Für die Überprüfung des Timings wurde eine "Tracing"-Technik verwendet, die vom Betriebssystem unterstützt wird. Dadurch war es möglich, mit geringem Aufwand alle wesentlichen Aspekte des Timings zu prüfen. Die erstellten Ablaufprotokolle zeigten, daß die bei der Programmierung vorgegebene Abfolge von Zeitgeberereignissen sehr genau eingehalten werden. Es gibt lediglich sporadisch, geringe Abweichungen von diesen Vorgaben in der Größenordnung von wenigen Millisekunden. Der korrekte Ablauf der Protokollverarbeitung wird durch diese geringen Abweichungen nicht gefährdet.[3]

9.6.2 Unterstützung der Fehlersuche

Zur Unterstützung der Fehlersuche wurden für wichtige Kontrollen zusätzliche Funktionen implementiert, die im Produktionsbetrieb nicht benötigt werden und vom Protokollautomaten somit auch nicht aufgerufen werden. Dennoch sind diese Funktionen im Programmcode enthalten, sie dienen als sog. "debugger hooks" (vgl. [154]).

Der Vorteil durch diese zusätzlichen Funktionen ist, daß bei der Werkzeug-unterstützten Fehlersuche mit einem "Debugger" diese zusätzlichen Funktionen an Unterbrechungspunkten interaktiv aufgerufen werden können, um beispielsweise den Inhalt von benutzerdefinierten Datenstrukturen auszugeben. Ein Beispiel für eine solche Funktion ist `hb_print()`. Diese Funktion dient zum Protokollieren von PDUs des GMPs und wird im laufenden Betrieb nicht benötigt. Zur Fehlersuche war es jedoch sehr hilfreich, sich den Inhalt einer empfangenen Dateneinheit automatisch interpretieren zu lassen. Dazu war es lediglich erforderlich, im "Debugger" einen Unterbrechungspunkt hinter der Empfangsfunktion zu setzen. Durch den interaktiven Aufruf der Funktion `hb_print()` wurde die PDU dann transliteriert und in einer lesbaren Form auf dem Bildschirm dargestellt.

[3] Durch die Verwendung einer "Tracing"-Technik, die durch die Betriebssystemunterstützung sehr aufwandsarm ist und daher den normalen Programmablauf kaum beeinflußt, kann weitgehend ausgeschlossen werden, daß das Timing nur *durch den Einsatz* dieser Technik *zufällig* spezifikationskonform ist.

9.6.3 Konsequente Modularisierung

Auf die allgemeinen Vorteile der Modularisierung braucht hier nicht weiter eingegangen zu werden. Ein zusätzlicher Vorteil der Modularisierung wird im Zusammenhang mit den oben diskutierten "Tracing"-Techniken erkennbar.

Der Detaillierungsgrad der Ablaufprotokolle konnte u.a. durch die Angabe von Modulnamen beeinflußt werden. So war es möglich, den Ablauf zusammenhängender Funktionen durch Angabe des Modulnamens zu verfolgen. Es konnte damit auf einfache Weise der logische Ablauf des Protokollautomaten verfolgt werden, indem lediglich die Module mit den Funktionen zur Reaktion auf Nachrichten und Zeitgeberereignisse ausgewählt wurden.

Wurden dort Unregelmäßigkeiten erkannt, konnte der Detaillierungsgrad für die Fehlersuche verfeinert werden, indem einzelne Module nacheinander in die Ablaufverfolgung mit aufgenommen wurden. Die Analyse und Fehlersuche im Zuge der Tests konnte durch die Modularisierung somit "top down" erfolgen.

Abschließend kann man sagen, daß die Werkzeugunterstützung eine enorme Hilfe für die Testdurchführung ist. Der Programmierer muß hierfür zwar einen Mehraufwand bei der Erstellung des Prototypen in Kauf nehmen, erleichtert sich dadurch aber die Fehlersuche und das Testen der Implementation enorm.

9.7 Host-Sicherheit

Die GMP-Instanz ist als "Daemon"-Prozeß[4] implementiert. Dadurch kann die Instanz automatisch beim Starten des Rechners und somit auch automatisch nach einer Reparatur gestartet werden.

Da einige Systemaufrufe verwendet wurden, die nur mit Rechten eines Systemverwalters ausgeführt werden können (Wechseln in die Scheduling-Klasse der Echtzeitprozesse, Einfrieren aller benutzten Speicherseiten, um ein Auslagern ("Paging") selbiger zu verhindern, direkter Zugriff auf die Netzschnittstelle durch Umgehen der Standard-Schnittstellen [220]), mußte besondere Sorgfalt darauf gelegt werden, daß dieses Programm nicht für einen Puffer-Überlauf ausgenutzt werden kann. Andernfalls hätte ein auf dem Rechner lokal eingetragener Benutzer die Möglichkeit, mit moderatem Aufwand die Rechte eines Systemverwalters zu erlangen (vgl. [64]). Der Quellcode wurde daher mit einem speziellen Programm zum Aufdecken von Sicherheitslücken ("Security Scanner", vgl. [232]) auf bekannte Schwachstellen überprüft.

Auch an dieser Stelle sei noch einmal darauf hingewiesen, daß die eben vorgestellten Maßnahmen nur die notwendigen Ergänzungen für die zusätzliche Absicherung der überlebensfähigen Packet Screen darstellen. Die eingangs formulierte Grundannahme, daß die

[4] Ein Prozeß, der losgelöst von einem steuernden Terminal ausgeführt wird und somit auch nach dem Beenden des Vaterprozesses (beispielsweise die Shell, aus der dieser Prozeß gestartet wurde) weiter ausgeführt wird.

sequentielle Packet Screen ausreichend abgesichert ist, erfordert bereits aufwendige Maßnahmen (vgl. [90,60,93]). Diese Maßnahmen und zusätzliche Verhaltensregeln für den nicht auszuschließenden Fall, daß dennoch ein Einbruch in das System stattfindet [238], können hier nicht diskutiert werden.

9.8 Leistungsbewertung einer abgesicherten parallelen Packet Screen

Es wurde auf vier verfügbaren Instanzen das GMP installiert und folgendermaßen konfiguriert:

- $HB_{intervall} = 100$ msec, d.h. jeweils nach 1/10 sec wird eine HB-Nachricht versendet.
- $HB_{maxlost} = 10$, d.h. 1,3 sec nach einem Ausfall ist wieder Konsens hergestellt (vgl. Abschnitt 74).

Die Belastung durch die GMP-Instanzen ist sehr gering, wie die folgende Statistik einer der vier Workstations belegt (der GMP-Prozeß ist als "hbserver" eingetragen):

```
last pid:  1556;  load averages:  0.04,  0.12,  0.12              17:31:57
24 processes:  23 sleeping, 1 on cpu
CPU states: 99.8% idle,  0.0% user,  0.2% kernel,  0.0% iowait,  0.0% swap
Memory: 128M real, 65M free, 128M swap free

  PID USERNAME THR PRI NICE  SIZE   RES STATE   TIME    CPU COMMAND
 1556 root       1  33    0 1968K 1544K cpu     0:00  0.69% top
 1552 root       1  75  -20 1008K  880K sleep   0:22  0.09% hbserver
```

Die CPU ist zu 99% unbelegt ("idle"). Die GMP-Instanz benötigt lediglich 0,09% der CPU-Zeit. Es traten allerdings kurzzeitige Lastspitzen auf, die Instanz hat aber zu keiner Zeit mehr als 10% der benutzten CPU-Zeit benötigt. Diese "Peaks" sind allerdings sehr selten und möglicherweise auch nur durch das Statistikprogramm induziert.

Die folgende Statistik zeigt die Auslastung dieser Instanz während einer Messung. Die CPU ist fast ständig belegt und die meiste Zeit[5] (zu 91,2%) mit Aufgaben im Kernel beschäftigt, da das Filtermodul Teil des Kernels ist.

```
last pid:  1557;  load averages:  0.68,  0.20,  0.13              17:44:16
24 processes:  22 sleeping, 1 running, 1 on cpu
CPU states:  7.7% idle,  1.1% user, 91.2% kernel,  0.0% iowait,  0.0% swap
```

[5] Die Auslastung ist nicht 100%, da vier Instanzen in der Lage sind, die gesamte Last eines einzelnen Lastgenerators zu verarbeiten, ohne dabei ihre Belastungsgrenzen zu erreichen.

```
Memory: 128M real, 65M free, 128M swap free

  PID USERNAME THR PRI NICE  SIZE   RES STATE   TIME    CPU COMMAND
 1552 root       1  75  -20 1008K  880K sleep   0:28  0.87% hbserver
 1557 root       1  33    0 1968K 1544K cpu     0:00  0.69% top
  214 root       1 -11   15  808K  552K run     0:03  0.00% screenblank
```

Auch hier ist die Belastung durch die GMP-Instanz sehr gering (weniger als 1%). Die folgende Statistik zeigt die Belastung des Systems während eines (simulierten) Angriffs. Es wurden genügend viele Dateneinheiten mit falscher Integritätsprüfsumme gesendet, um die GMP-Instanz in den Zustand "ATTACK-DETECTED" zu versetzen. Hier schlägt sofort die Flußkontrolle zu und somit sinkt die Belastung der Maschine durch die GMP-Instanz auf geringe 0,01% CPU-Auslastung.

```
last pid:  1561;  load averages:  0.00,  0.06,  0.10              18:01:05
24 processes:  23 sleeping, 1 on cpu
CPU states: 99.8% idle,  0.0% user,  0.2% kernel,  0.0% iowait,  0.0% swap
Memory: 128M real, 65M free, 128M swap free

  PID USERNAME THR PRI NICE  SIZE   RES STATE   TIME    CPU COMMAND
 1561 root       1  33    0 1968K 1544K cpu     0:00  0.26% top
 1552 root       1  75  -20 1008K  880K sleep   0:39  0.01% hbserver
```

Das GMP ist somit in der Tat leichtgewichtig und verhält sich während eines DoS-Angriffs sehr Betriebsmittel schonend.

9.9 Zusammenfassung

Es wurde eine Implementierung eines Prototypen für ein sicheres GMP diskutiert, die der Spezifikation aus Kapitel 7 und den diskutierten Sicherheitsmechanismen aus Kapitel 8 genügt. Hierbei stellte sich heraus, daß es zur Unterstützung der Implementation und für die anschließenden Tests viele gute, teilweise sogar frei verfügbare Werkzeuge gibt. Nachteilig ist lediglich, daß die meisten Werkzeuge plattformabhängig sind.

Es stellte sich in der Tat heraus, daß die Verfügbarkeit des Quellcodes (und hier ist vor allem die Verfügbarkeit des Betriebssystem-Quellcodes gemeint) zur Zeit eine der wesentlichen Voraussetzungen für die Implementierung von Bausteinen für überlebensfähige Systeme ist (vgl. Abschnitt 2.2.4). Die in diesem Abschnitt diskutierten, für das GMP und die effiziente, parallele Packetverarbeitung erforderlichen Modifikationen an den Gerätetreibern wären andernfalls nicht möglich gewesen.

Es konnte weiterhin gezeigt werden, daß eine effiziente Implementation (und dies ist eines der wesentlichen, globalen Optimierungsziele für überlebensfähige Systeme) nur durch

hardwarenahe Programmierung, d.h. durch Modifizieren des Gerätetreibers der Netzschnittstelle im Betriebssystem, möglich ist. Die Beschränkung der allgemeinen Anwendbarkeit des spezifizierten GMPs auf lokale Netze und das einfache, aber praxisnahe Fehlermodell, *erlauben konzeptionell eine effiziente Implementierung.* Das *Erreichen einer effizienten Implementierung* ist aber nur durch eine hochgradige Spezialisierung auf eine bestimmte Architektur, teilweise mit erheblichen Erweiterungen an vorhandener Betriebssystem-Software, möglich.

Wird so vorgegangen, dann ist, wie oben gezeigt, eine hochgradig effiziente Lösung erreichbar. Die zusätzliche, sichere Ausfallbehandlung ist nur mit minimalen Leistungseinbußen verbunden. Leistungsfähigkeit, Verfügbarkeit und Sicherheit sind gleichzeitig erreichbar.

Kapitel 10

Ein überlebensfähiges Perimeternetz für Hochgeschwindigkeitsnetze

> " The whole is more than the sum of its parts. "
>
> —Aristotle (384-322 B.C.), Metaphysics

Der Grund für die Überführung der Packet Screen in ein überlebensfähiges Pendant war die Notwendigkeit, diese weit verbreitete, akzeptierte Firewall-Komponente für den Einsatz in zukünftigen Hochgeschwindigkeitsnetzen vorzubereiten. Es gibt jedoch eine große Anzahl weiterer Dienste in verteilten Systemen, die ebenfalls nicht überlebensfähig sind. Es wäre daher nützlich, wenn sich die erarbeiteten Ergebnisse dazu verwenden ließen, auch andere Dienste zu adaptieren, d.h. überlebensfähig auszulegen. Durch die steigende Nachfrage nach "Online-Diensten" sind insbesondere hoch verfügbare und leistungsfähige Lösungen gefragt [213]. In diesem Kapitel soll daher aufgezeigt werden, welche weiteren Dienste von einer dezentralen Selektion mit anschließender Parallelverarbeitung von Dateneinheiten und einem leichtgewichtigen, sicheren Gruppenmanagement profitieren können.

Aufgrund der Entwurfsentscheidung, das GMP zugunsten der Leichtgewichtigkeit nur für die Verwaltung von lokalen Prozessorgruppen auszulegen, ist es sicherlich nicht für alle verteilten Dienste gleichermaßen gut geeignet. Am besten geeignet ist es für Dienste, die auf eine lokale Gruppe von Prozessoren verteilt werden können und untereinander wenig kommunizieren. Der verteilte Selektionsalgorithmus und die in Kapitel 5 diskutierten Adaptionen können für Dienste eingesetzt werden, die wie die Packet Screen viele Dateneinheiten bzw. viele Kommunikationsbeziehungen verarbeiten müssen.

Es ist daher naheliegend, primär solche Dienste zu betrachten, die ähnliche, kommunikationsbezogene Dienstleistungen erbringen, wie die Packet Screen. Derartige Dienste befinden sich vor allem in Schutznetzen (Perimeternetzen), die zwischen zwei Sicherheitsdomänen (beispielsweise Firmennetz und Internet) errichtet werden. Es wird daher ein Architekturbeispiel für ein überlebensfähiges Perimeternetz vorgestellt. In den darauf folgenden Unterabschnitten werden die für die einzelnen Dienste in dieser Sicherheitsdomäne not-

wendigen Anpassungen diskutiert. Es wird dargestellt, wie der in Abschnitt 42 vorgestellte Selektionsalgorithmus an den jeweiligen Dienst angepaßt werden kann.

Damit wird verdeutlicht, daß sich die Überführung einer Packet Screen in eine überlebensfähige Firewall-Komponente, aufgrund der gesammelten Erkenntnisse, aufwandsarm auf andere Firewall-Komponenten und einige weitere Dienste verallgemeinern läßt.

10.1 Eine Beispielarchitektur

In Abbildung 10.1 ist eine typische Firewall-Architektur abgebildet. Es handelt sich um ein Beispiel der in Abbildung 2.4 schematisch angedeuteten Architektur "beidseitig geschirmte Bastion". Das Perimeternetz enthält neben verschiedenen Bastionen ein VPN-Gateway (VPN-GW) für die Ver- und Entschlüsselung von VPN-Kommunikation. Darüber hinaus ist hier auch ein DNS-Server angesiedelt. Dem Perimeternetz aus Sicherheitsgründen vorgelagert sind die öffentlichen Informations-Server (typischerweise HTTP-Server und FTP-Server). Im gesicherten internen Netz ist ein "Intrusion Detection"-System installiert und ein "Logging"-Server, der auch alle Arten von Audit-Informationen speichern soll.

Alle genannten Komponenten lassen sich auch als "Cluster" auslegen. Dies ist in der Abbildung bereits angedeutet: Anstelle nur eines Prozessors/Rechners werden wie bei der parallelen Packet Screen (PPS) die Instanzen auf mehrere, parallel angeordnete Rechner kopiert. Jedes der eingezeichneten "Cluster" besteht aus zwei oder mehreren Rechnern mit jeweils einem replizierten Dienst sowie einem Switch oder Hub, um Aufträge an alle Replikate des "Clusters" zu kopieren. Die notwendigen Änderungen am Selektionsalgorithmus werden für jeden Dienst bzw. für jedes dieser "Cluster" im folgenden Abschnitt diskutiert.

10.2 Überlebensfähige Firewall-Komponenten

In der Beispielarchitektur sind mehrere Firewall-Komponenten enthalten. Die folgende Unterabschnitte diskutieren die erforderlichen Veränderungen am Selektionsalgorithmus, für die Parallelisierung von Bastionen, VPN-Gateways und Monitoren.

10.2.1 Bastionen und Proxies

Bastionen und Proxies sind sehr häufig anzutreffende Firewall-Komponenten (vgl. Abbildungen 2.2–2.4): Durch die Anwendungsnähe ist ein Proxy in der Lage, eine sehr detaillierte Zugriffskontrolle und ein ebenso ausführliches Audit durchzuführen. Damit bieten Proxies in der Regel ein höheres Maß an Sicherheit als der alleinige Einsatz einer Packet Screen.

In den eingangs dargestellten Firewall-Architekturen wird die Kommunikation letztendlich durch die Packet Screens über die Bastionen gezwungen (vgl. Abbildungen 2.2–2.4). Die

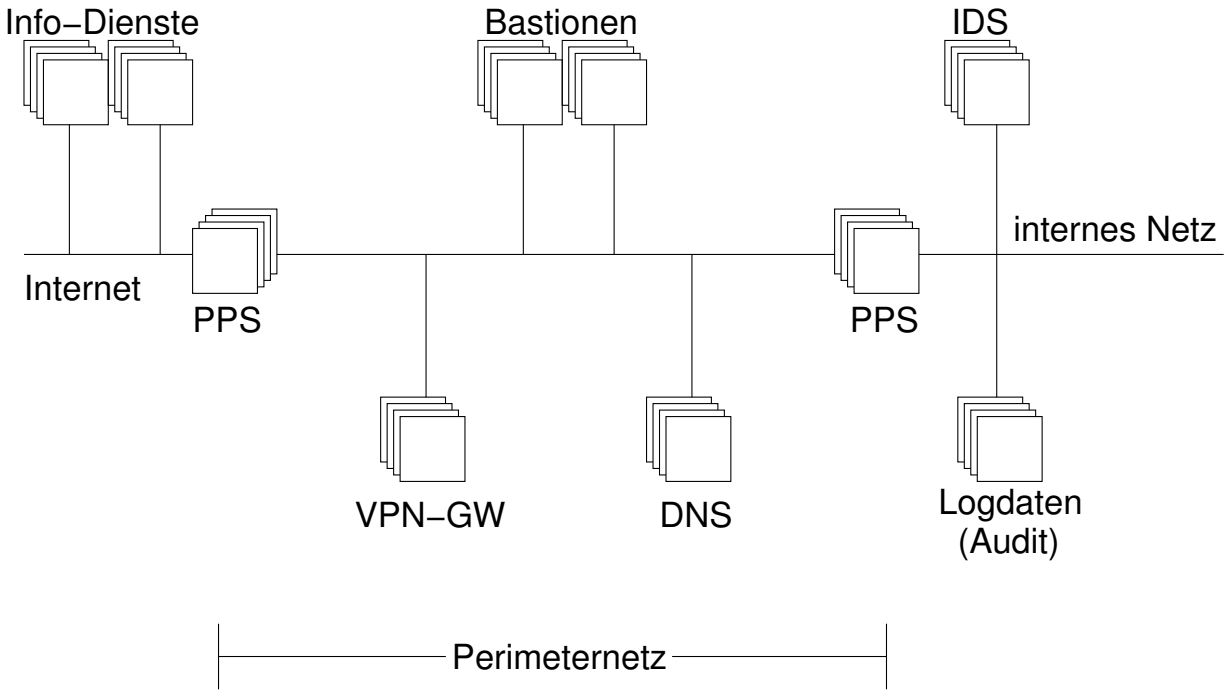


Abbildung 10.1: Perimeternetz mit mehreren "Clustern"

Packet Screens verhindern eine Umgehung der Bastionen durch Kommunikationsbeziehungen, für die ein Proxy auf der Bastion vorgesehen ist. Dadurch wird sichergestellt, daß die dienst-spezifische Zugriffskontrolle durch die Sicherheits-Proxies erfolgen kann.

Die feingranulare Zugriffskontrolle auf den Bastionen ist jedoch mit erheblichen Durchsatzeinbußen – im Vergleich zum alleinigen Einsatz einer Packet Screen – verbunden (vgl. [78, 71]). Auch in diesem Fall kann eine Dezentralisierung der Bastion helfen (vgl. [77, 78, 76, 71]). Werden hierfür jedoch zentrale Verteiler eingesetzt, ergeben sich leicht neue Engpässe und Verfügbarkeitsprobleme. Im obigen Beispiel übernehmen die parallelen Packet Screens die dezentrale Verteilung auf verschiedene (selbst wieder dezentralisierte) "Bastionen-Cluster". Die Verteilung kann damit hochgradig ausfallsicher und schnell erfolgen.

Aus Sicherheitsgründen würden für Proxies unterschiedlicher Sicherheitsstufen verschiedene Bastionen-Cluster gewählt werden. Die Parallelverarbeitung ermöglicht in diesem Fall nicht nur eine Beschleunigung der Zugriffe über die Proxies, sondern erhöht auch die Sicherheit durch das Bilden parallel arbeitender "Cluster" für verschiedenartige Zugriffskontrollen bzw. Risikopotentiale. Die Aufteilung der verschiedenen Kommunikationsbeziehungen innerhalb eines Bastionen-Clusters würde nach der in Abschnitt 5.2 vorgestellten Modifikation des Selektionsalgorithmus erfolgen, so daß Dateneinheiten einer individuellen Kommunikationsbeziehung immer von derselben Bastion verarbeitet werden.

Die für die Selektion relevanten Informationen sind in diesem Fall die IP-Adressen der Kommunikationspartner und die Dienstkennungen (Ports) in den Transportprotokollköpfen (UDP- bzw. TCP-Ports).

10.2.2 "Virtual Private Network"-Gateways

In dieser Arbeit wurde an verschiedenen Stellen darauf hingewiesen, daß Ver- und Entschlüsselung sehr aufwendige Operationen darstellen. Gerade dies ist jedoch der Hauptmechanismus, der auf allen "Virtual Private Network"-Endpunkten eingesetzt wird. Daher benötigt man für VPN-Gateways, die eine transparente Ver- und Entschlüsselung für dahinter liegende Subnetze durchführen, besonders leistungsfähige Hardware.

Auch hierfür kann eine Leistungssteigerung durch die Anpassung des Selektionsalgorithmus an eine Verbindungsorientierung erfolgen. Für den Fall, daß eine Verschlüsselung zwischen mehreren Filialnetzen erreicht werden soll, reicht eine Selektion basierend auf Sender- und Empfänger-IP-Adresse. Andere Informationen, wie beispielsweise die Port-Kennungen, können bei der Verwendung von IPSec (vgl. [12]) im "Tunnel-Mode" bei der Selektion vernachlässigt werden bzw. stehen nicht zur Verfügung, falls Verschlüsselung eingesetzt wird.

Für das Verhandeln der "Security Association, SA" ist ebenfalls nur eine Selektion über Sender- und Empfänger-IP-Adresse erforderlich. Dies führt zu einer Verteilung der "Tunnel" der lokalen Filiale zu anderen Filialen auf das VPN-Gateway-Cluster. Fällt eine VPN-Gateway-Instanz aus, würde nach einer Rekonfiguration automatisch eine neue SA aufgebaut werden.

Da die Verschlüsselung gegenüber der Filterverarbeitung eine sehr aufwendige Operation ist ($B_{fil} \gg B_{sel}$ in Formel 4.15), kann man hier mit besonders gutem Gewinn durch eine Parallelisierung rechnen.

10.2.3 "Sniffer" für "Intrusion Detection"-Systeme

Ein großes Problem bei den netzbasierten "Intrusion Detection" Systemen (IDS) ist neben der eigentlichen Erkennung einer "Intrusion", alle relevanten Ereignisse für eben diese Erkennung aufzuzeichnen. Konzeptionell ist dies nur durch sog. "audit based"-IDS möglich. Hierbei wird sämtliche Aktivität zuerst aufgezeichnet und anschließend analysiert (nachbearbeitet). Dadurch ist (zumindest konzeptionell) eine beliebig komplexe Korrelation zwischen den aufgezeichneten Ereignissen herstellbar [194].

Eine weitere Möglichkeit, nämlich die zeitnahe Erkennung einer "Intrusion" erfordert die augenblickliche Analyse der aufgetretenen Ereignisse. Dieses Vorgehen der sog. "Inline-IDS" zeichnet nur noch als relevant erkannte Ereignisse auf und versucht eine Einbruchserkennung im Zuge dieser Analyse [244].

In beiden Fällen ist jedoch vor einer Analyse eine Aufzeichnung dieser Daten erforderlich. Bereits hierbei kann es zu erheblichen Leistungsproblemen kommen, da eben alle Dateneinheiten eines Netzabschnittes aufgezeichnet werden müssen (vgl. [186]). Nur so kann sichergestellt werden, daß weder bei einer anschließenden "audit based"-Analyse noch bei einer "inline"-Analyse gerade die relevanten Ereignisse aufgrund einer Überlast verloren werden. Auch beim Netz-Monitoring hat sich Parallelverarbeitung bereits bewährt. Während in [22] mit Hilfe der Parallelverarbeitung aufwendige Analysen ermöglicht wurden, was insbesondere für "Inline-IDS" vorteilhaft ist, ermöglicht das verteilte Selektionsverfahren eine verlustfreie Aufzeichnung der Rohdaten für eine anschließende "audit based"-Analyse.

In diesem Fall kann wieder die bereits bei der parallelen Packet Screen verwendete Technik der Paketparallelität verwendet werden. Eine Synchronisation der verteilt aufgezeichneten Paketströme kann aufwandsarm durch die von den GMP-Instanzen generierten HB-Nachrichten erfolgen. Sie bieten die Möglichkeit, Ereignisse mit einer Auflösung von $HB_{intervall}$-Sekunden zu (re)synchronisieren.

Eine Unterstützung von "Inline-IDS" ist alternativ zu dem Vorgehen in [22] ebenfalls möglich. Für eine zeitnahe Auswertung der Rohdaten ist es dann jedoch sinnvoller, eine Parallelisierung anhand der Verbindungparallelität vorzunehmen. Dadurch ist sichergestellt, daß vor allem interaktive (d.h. in aller Regel verbindungsorientierte) Sitzungen erkannt werden, die versuchen, eingebaute Systemlücken ("Backdoors") auszubeuten [243].

10.2.4 Audit-Trails

Neben den Aufzeichnungen durch eine netzbasierte IDS-Komponente fallen (per Definition, vgl. Definition 2.3) auf einem Firewall bzw. auf jeder Firewall-Komponente Audit-Trails an. Da auch nach einem partiellen Einbruch in ein überlebensfähiges System, im Sinne der "Anpassung und Evolution" (vgl. Abschnitt 2.2.3), eine Auswertung der Schwachstelle, die zum Einbruch führte, möglich sein muß, ist besondere Sorgfalt bei der Sicherung der Audit-Trails angebracht. Konkret bedeutet dies, daß die *Verfügbarkeit der Log-Informationen* gewährleistet sein muß. Typischerweise werden dafür neben der lokalen Aufzeichnung auf der jeweiligen Firewall-Komponente die Audit-Daten zusätzlich an ein zentrales Audit-System weitergeleitet (dies kann auch ein IDS-System sein).

Wird im Zuge des Einbruchs jedoch auch dieses zentrale Audit-System kompromittiert, ist die Verfügbarkeit dieser Log-Informationen nicht mehr sicherzustellen. Eine mögliche Gegenmaßnahme besteht in der Verteilung der Log-Daten auf eine große Anzahl n von (potentiell nicht vertrauenswürdigen) Log-Maschinen mit einem Algorithmus, der für die Wiederherstellung der gesamten Log-Informationen nur m aus n Log-Maschinen benötigt. Ein Angreifer müßte somit mindestens $m + 1$ der n Log-Maschinen kompromittieren, um die Verfügbarkeit der Audit-Trails zu unterbinden (vgl. [11]).

Ein Problem bei der Architektur aus [11] ist, daß alle lokalen Log-Dienste hierzu angepaßt werden müssen, um die Verteilung der Informationen an die n Log-Maschinen vorzunehmen. Hierzu wäre der "Log Availability Filter" (LAF) auf allen Maschinen zu installieren,

die Audit-Trails generieren. Darüber hinaus kann bei der Verwendung des "syslog"-Dienstes zur weiteren Verteilung an die n Log-Maschinen nicht garantiert werden, daß die Informationen tatsächlich auf allen Maschinen ankommen, da der "syslog"-Dienst auf UDP basiert und somit verbindungslos arbeitet.

Der Einsatz des verteilten Selektionsalgorithmus bietet auch für diese wichtige Sicherungsaufgabe zwei Vorteile, die je nach Architektur (LAF-Architektur mit/ohne Rekonfiguration) zum Tragen kommen.

LAF-Architektur ohne Rekonfiguration: Der Vorteil dieser Architektur ist, daß keine Veränderungen an den Standard- "syslog"-Dienst der Audit-Trails generierenden Firewall-Komponenten vorgenommen werden müssen. Die Firewall-Komponenten senden ihre Log-Daten wie üblich zu einem nun dezentralisierten Log-Service. Auch hier werden die Dateneinheiten über eine "Multicast-Adressierung" an alle "Log-Instanzen" verteilt. Jede Instanz führt dann auf den Informationen die in [11] vorgeschlagenen Operationen aus und speichert lokal den errechneten Teilstring, dessen Index mit der Instanz-ID übereinstimmt. Im Vergleich mit der Selektionsoperation aus Abschnitt 42 wird hier also genau andersherum verfahren: Alle Instanzen führen dieselbe Berechnung durch (sozusagen erst B_{fil}) und speichern anschließend nur den für sie bestimmten Teil (vergleichbar mit B_{sel}). Der Parameter m, der die Anzahl der Teile angibt, die für eine Rekonstruktion der gesamten Log-Daten erforderlich sind, ist in dieser Architektur (wie auch in [11]) ein Konfigurationsparameter und kann nach den dort diskutierten Kriterien frei gewählt werden.

Da jedoch nur m aus n Log-Einträge erforderlich sind, um einen Eintrag wieder herzustellen, ist auch bei Ausfall einer Komponente keine Rekonfiguration erforderlich. Es kann also auf ein GMP verzichtet werden, so lange für die Anzahl der zeitgleichen Ausfälle (f_{total}) gilt: $f_{total} \leq n - m$.

LAF-Architektur mit Rekonfiguration: Noch größer wird jedoch der Vorteil, wenn durch ein zusätzliches GMP die aktuelle Gesamtzahl der Instanzen jederzeit bekannt ist. In diesem Fall ist es möglich, durch Rekonfiguration des Dispersions-Algorithmus ein m_{neu} festzulegen. Dies ist genau dann erforderlich, wenn der Dispersions-Algorithmus aufgrund von $f_{total} > n - m$ scheitern würde, die Verfügbarkeit der Audit-Trails somit nicht mehr gewährleistet werden kann. Die Rekonfiguration würde dann ein m_{neu} derart berechnen, daß wieder gilt: $f_{total} \leq n - m_{neu}$.

Beide hier diskutierten Varianten bieten gegenüber der Original-Lösung noch den zusätzlichen Vorteil, daß aufgrund der Robustheit durch die lokale Speicherung der Audit-Trails auf den parallelen Log-Instanzen, auf eine nicht-standardkonforme Erweiterung der "syslog"-Kommunikation über TCP verzichtet werden kann: Die Log-Daten werden nach der Verteilung nicht noch einmal (ungesichert) übertragen, sondern lokal gespeichert.

10.3 Überlebensfähige Informationsdienste

Teil typischer Kommunikationsinfrastrukturen sind Informations-Dienste. Diese Dienste werden auf einer oder mehreren Komponenten erbracht. Sie sind in der Regel entweder für einen ordentlichen Netzbetrieb (beispielsweise DNS-Dienst) oder für die Selbstdarstellung eines Unternehmens (beispielsweise HTTP-Dienst) erforderlich. In jedem Fall ist eine Überlebensfähigkeit dieser Komponenten wünschenswert.

10.3.1 DNS-Service

Die Überlebensfähigkeit des DNS-Dienstes ist für die ordentliche Auflösung von symbolischen Namen (beispielsweise www.blafasel.de) zu IP-Adressen unbedingt erforderlich. DNS (vgl. [172, 173]) bietet daher bereits die Möglichkeit, mehrere "secondary masters" für eine Zone neben dem eigentlichen "primary master" festzulegen, die alle berechtigt sind ("authoritative"), für die jeweilige Zone die Namensinformationen bereitzustellen. Fällt einer der Server aus, kann durch einen Client oder rekursiv durch einen anderen Server bei einem weiteren DNS-Server ("primary master" oder einer der "secondary master") nachgefragt werden. DNS verfügt somit bereits konzeptionell über eine erhöhte Ausfallsicherheit.

Die Antwortzeiten der einzelnen Server einer Zone sind allerdings sehr stark von der Größe dieser Zone abhängig. Je mehr Rechner sich innerhalb einer Zone befinden, für die ein DNS-"primary master" verantwortlich ist, mit desto mehr Anfragen von diesen Clients der eigenen Zone muß gerechnet werden. Aber auch die Anzahl der Anfragen von Clients anderer Zonen nach Informationen aus der eigenen Zone nehmen typischerweise mit der Zonengröße zu.

Auch wenn große Zonen unterteilt werden, um die Verantwortung für Teile dieser Zone an andere Server zu delegieren, müssen trotzdem immer noch die Informationen bezüglich der Delegation ("glue records") innerhalb der übergeordneten Zone(n) vom DNS-Server verarbeitet werden. Dies gilt insbesondere für die sog. "Top-Level-Domains" und die Name-Server, die maßgeblich (authoritative) für die Delegations-Informationen dieser Domänen sind.

Auch in diesem Fall kann eine deutliche Beschleunigung durch ein Cluster von mehreren DNS-Servern erreicht werden. Die Selektion geschieht hier nur für eintreffende UDP-Dateneinheiten, basierend auf der Sender-IP-Adresse. Alle DNS-Anfragen werden somit auf die parallelen Instanzen verteilt. Jede einzelne Instanz kann über TCP, ggf. mit einer individuellen IP-Adresse, weiterhin "Zone-Updates" versenden bzw. empfangen.

Hieraus resultiert eine lokale Hochverfügbarkeit und eine deutliche Geschwindigkeitssteigerung gegenüber einer sequentiellen Implementation. Der Vorteil dieser lokalen Hochverfügbarkeit ist, daß die Wahrscheinlichkeit für den Erhalt einer Antwort auf eine Anfrage sehr groß ist. Bei Anfragen, die ein lokaler DNS-Server an einen "dezentralisierten Kollegen" weiterreicht, wird garantiert geantwortet, und es wird in den seltensten Fällen erforderlich

sein, daß ein Anfrage nach Ausbleiben der Antwort erneut – an einen anderen DNS-Server der Zone – gestellt werden muß. Zusammen mit der durch die Parallelisierung herbeigeführten Beschleunigung ergeben sich somit sehr geringe Wartezeiten auf DNS-Antworten aus "parallelisierten" Domänen.

Die Notwendigkeit für eine solche Leistungssteigerung der einzelnen DNS-Server wird insbesondere dann offensichtlich, wenn die aus Sicherheitsgründen für diesen essentiellen Dienst seit langem überfällige Erweiterung zu einem abgesicherten DNS-Dienst (vgl. [69,237,236]) erfolgt. Spätestens dann, wenn asymmetrische kryptographische Verfahren zum Signieren von DNS-Einträgen verwendet werden, kommen die einzelnen Server sehr schnell an ihre Leistungsgrenzen, wenn sie stellvertretend für Clients ihrer Domäne die Signaturen der DNS-Antworten anderer Server überprüfen.

10.3.2 HTTP-Server

Das HTTP-Protokoll ist die Grundlage für die unter dem Begriff "World Wide Web" bekannten Dienste. Neben der Übertragung von HTML-Seiten migrieren eine Vielzahl "klassischer" Dienste, wie beispielsweise FTP und Gopher, zu einer Nutzung des HTTP-Protokolls bzw. werden durch funktional ähnliche und gleichzeitig leichter zu bedienende Dienste auf Basis des HTTP-Protokolls ersetzt.

Der Trend geht zu einer immer aufwendigeren Darstellung von Inhalten und noch größeren Interaktionsmöglichkeiten seitens der Benutzer. Dies wird durch zwei wesentliche Mechanismen erreicht:

- Verlagerung aufwendiger Berechnungen auf den Client. Hierbei sendet der Server anstelle von vollständig aufbereiteten HTML-Seiten um Programme ergänzte Seiten. Die Programme werden auf dem Client im Zuge der Seitendarstellung und der Benutzerinteraktion ausgeführt. Der Server kann bei vernünftigem Einsatz dieser Techniken ("Javascript", "Java", "Active X") erheblich entlastet werden und das Kommunikationsaufkommen zwischen Client und Server läßt sich drastisch verringern. Ein positives Beispiel ist die Ausführung von derartigem "Scriptcode" bei Eingaben eines Benutzers in ein WWW-Formular. Die formale Korrektheit der Eingabe (Formatüberprüfung) kann bereits auf dem Client erfolgen. Ein Absenden des Formulars wird verhindert, bis die Eingaben den formalen Kriterien genügen. Würde eine Überprüfung erst auf dem Server stattfinden, müßte dieser bei einer falschen Eingabe das Formular mit entsprechenden Hinweisen erneut zurücksenden.

  Der Hauptnachteil dieser Techniken sind die seit langem bekannten, hiermit verbundenen Sicherheitsrisiken. Die Skriptsprachen erlauben meist deutlich umfangreicheren Zugriff auf die lokalen Ressourcen des Clients als für eine Unterstützung der Interaktion erforderlich wäre (beispielsweise beliebigen Zugriff auf lokale Dateisysteme). Dies stellt ein ganz erhebliches Sicherheitsrisiko dar [2, 151, 95]. Aber auch der Server ist durch den unvorsichtigen Einsatz dieser Technologien gefährdet.

Verzichtet der Server beispielsweise auf eine Formatüberprüfung beim Empfang eines WWW-Formulars, da davon ausgegangen wird, daß diese Überprüfung bereits durch Berechnungen auf dem Client erfolgte, so ist damit DoS-Angriffen Tür und Tor geöffnet: Es können über das Rechnernetz systematisch Fehler auf dem Server herbeigeführt werden.

- Dynamische Erstellung von HTML-Dokumenten auf dem Server. Hierbei werden die eigentlichen Seiteninhalte erst im Zuge einer Anfrage des Clients auf dem Server dynamisch generiert. Dies geschieht meist durch den Aufruf von externen Programmen, die beispielsweise Datenbanken abfragen und die Ergebnisse in HTML einbetten, bevor diese Seite an den Client zurückgesendet wird. Es sind in diesem Umfeld aber auch eine Reihe weiterer Ansätze bekannt, beispielsweise "PHP" und "Active Server Pages" (vgl. [188]).

Die beiden oben genannten Vorgehensweisen werden in der Regel miteinander kombiniert. Es ist unmittelbar einsichtig, daß die Server-seitige Ausführung von Programmen zur dynamischen Generierung von HTML-Seiten erhebliche Anforderungen an die Leistungsfähigkeit der Server-Rechner stellt. Auch in diesem Fall kann wiederum die Parallelisierung auf Basis der Verbindungsparallelität benutzt werden, um einen parallelen HTTP-Server einzurichten.

Der Selektionsalgorithmus (vgl. Abschnitt 4.3.5) benutzt als Eingabe für die Streufunktion (Hash-Funktion) die IP-Adressen und Port-Nummern von Sender und Empfänger, wie im Pseudo-Code in Abschnitt 5.2.4 dargestellt. Dadurch werden die einzelnen Anfragen auf die parallelen HTTP-Server-Instanzen des HTTP-Server-Clusters verteilt und somit auch die aufwendigen Server-seitigen Berechnungen dezentralisiert.

10.4 Zusammenfassung

In diesem Kapitel wurde noch einmal verdeutlicht, daß die im Detail vorgeführte Überführung einer Packet Screen in eine überlebensfähige Komponente, auf Basis von Parallelverarbeitung, für eine Vielzahl weiterer Sicherheitskomponenten und sonstiger (Internet-)Dienste mit geringen Anpassungen möglich ist.

Die wesentliche Anpassung bezieht sich auf den Selektionsalgorithmus, der entweder nach dem Paradigma der Paketparallelität oder nach dem Paradigma der Verbindungsparallelität konfiguriert werden muß. Diese Konfigurierung ist zwar dienstspezifisch, der zugrundeliegende, verteilte Selektionsalgorithmus ist hingegen vielseitig verwendbar, wie die hier diskutierten Beispiele zeigen.

Bei allen hier vorgestellten Diensten ist nach einer Dezentralisierung auch ein Gruppenmanagement wünschenswert bzw. erforderlich. Zwar könnte beim DNS-Dienst auf ein Gruppenmanagement verzichtet werden, da die Architektur dieses Dienstes ohnehin Redundanz vorsieht, jedoch würde auch dieser extrem wichtige IP-Basisdienst – wie alle anderen hier

vorgestellten Dienste – von einem Gruppenmanagement und der damit einhergehenden Erhöhung der Verfügbarkeit profitieren.

Kapitel 11

Resümee und Ausblick

In diesem Kapitel werden noch einmal die Ergebnisse im Zusammenhang dargestellt. Neben den bisher schon präsentierten Ergebnissen in den Einzelabschnitten wird hier auch noch einmal auf die Methodik an sich eingegangen.

11.1 Zusammenfassung der Ergebnisse

Das primäre Ziel war die Überführung einer Packet Screen in ein überlebensfähiges System, das in den heutigen und zukünftigen Hochgeschwindigkeitsnetzen eingesetzt werden kann. Für diese ganzheitliche Betrachtung von Systemen in Hinblick sowohl auf Leistungsfähigkeit, Verfügbarkeit, als auch auf Sicherheit, gibt es kaum durchgeführte Studien. In der vorliegenden Arbeit wurde primär der "Tradeoff" zwischen diesen gleichzeitigen Optimierungszielen dargestellt und exemplarisch durch zusätzliche Mechanismen aufgelöst (s.u.).

Die dafür, aus pragmatischen Gründen als sinnvoll erachtete Methode, ein vorhandenes System schrittweise zu verbessern und die mit den Verbesserungen einhergehenden Probleme beim nachfolgenden Schritt zu betrachten, war erfolgreich. Es konnte der Prototyp einer "überlebensfähigen Packet Screen" implementiert und bewertet werden.

Ursächlich für diese Überlegungen war die Erkenntnis aus vorangegangener Forschungsarbeit, daß Firewall-Komponenten zum Engpaß in Hochgeschwindigkeitsnetzen werden können. In Kapitel 3 führte dies zu einer methodischen Leistungsanalyse. Es wurde eine an den typischen Einsatzszenarien von Packet Screens orientierte Aufstellung der Last-, Leistungs- und Konfigurationskenngrößen zur Bestimmung der Engpässe vorgestellt. Die Notwendigkeit zur systematischen Bestimmung der Leistungskenngrößen erforderte umfangreiche Erweiterungen an Meßwerkzeugen (vgl. Anhang C). Trotz der zielgerichteten Erweiterung der Meßwerkzeuge zur Leistungsanalyse von Packet Screens können damit auch andere Netzkomponenten analysiert werden. Insbesondere die Erweiterungen zur Synchronisation mehrerer Lastgeneratoren sind hier exemplarisch zu nennen. Diese Erweiterungen ermöglichen auch die Bewertung von noch deutlich leistungsfähigeren Netzkomponenten. Mit

diesen Meßwerkzeugen konnten die Engpässe beim Einsatz von Packet Screens aufgezeigt werden. Der bloße Einsatz einer einzelnen Packet Screen kann zu einem Engpaß werden. Der Engpaß verschärft sich grundsätzlich durch Hinzufügen von Filterregeln.

In Kapitel 4 wurde dargestellt, wie durch einen skalierbaren Mechanismus dieser Engpaß nachhaltig beseitigt werden kann. Durch den Einsatz von Parallelverarbeitung, der effizienten Nutzung von ohnehin vorhandenen Netzkomponenten zum Kopieren der Last an alle parallelen Instanzen und einer dezentralen Lastverteilung konnte eine erhebliche Leistungssteigerung herbeigeführt werden. Da dieser Mechanismus vor der eigentlichen Filterbearbeitung auf der Packet Screen stattfindet, und nicht in die Filtersoftware sondern in das Betriebssystem integriert wurde, ist sogar eine Unabhängigkeit von der Packet Screen-Software gegeben. Die Nachhaltigkeit dieses Konzeptes konnte mit einem äußerst realitätsnahen, analytischen Modell belegt werden, mit dem Leistungsprognosen erstellt wurden, die die Leistung der parallelen Packet Screen unter Verwendung schnellerer Hardware und weiterer Instanzen angeben. Darüber hinaus bestätigen sich die mit dem analytischen Modell errechneten Werte und die Meßwerte gegenseitig. Die Validität der durchgeführten Messungen und der erstellten Prognosemodelle ist damit vollständig nachgewiesen worden.

In Kapitel 5 wurde dargestellt, wie der in Kapitel 4 vorgestellte Basisalgorithmus zur verteilten Selektion von Aufträgen variiert werden kann, um erweiterten Filteraufgaben gerecht zu werden und heterogene Hardware einsetzen zu können. Damit wurde vor allem die Praxisnähe des vorgeschlagenen Konzeptes erweitert. Optimale Leistungsfähigkeit kann zugunsten zusätzlicher Funktionalität aufgegeben werden, ohne daß ein völlig neues Konzept für die Parallelverarbeitung auf Packet Screens erarbeitet werden müßte.

In Kapitel 6 wurde der "Tradeoff" zwischen Leistungsfähigkeit und Verfügbarkeit dargestellt. Es wurde gezeigt, daß die Verfügbarkeit des Gesamtsystems durch die Parallelverarbeitung herabgesetzt wird. Darüber hinaus verringert der Ausfall einer Instanz die Gesamtleistungsfähigkeit der Packet Screen, unabhängig von der Anzahl der ausgefallenen und verbleibenden Instanzen. Eine parallele Packet Screen, die unter Teilausfällen (eine oder mehrere Instanzen) leidet, kann nachweislich nicht die potentielle Leistungsfähigkeit der verbleibenden Instanzen erreichen. Die hierfür erforderliche formale Betrachtung wurde wiederum mit Standardmethoden, basierend auf dem formalen Modell aus Kapitel 4, durchgeführt. Dieser unbefriedigende Zustand, die reduzierte Verfügbarkeit, führte zu einer allgemeinen Darstellung des Fehlermanagements mit einer Diskussion eines Fehlermodells für die parallele Packet Screen. Schließlich wurde motiviert, daß für eine erforderliche Rekonfiguration im Fehlerfall ein Standardmechanismus aus dem Bereich der fehlertoleranten Systeme eingesetzt werden kann. Es muß ein Gruppenmanagement-Protokoll eingesetzt werden, um Ausfälle zu erkennen und eine automatische Anpassung vorzunehmen.

Durch die gleichzeitige hohe Anforderung an die Leistungsfähigkeit konnte kein bekanntes Protokoll verwendet werden. Es wurde daher in Kapitel 7 ein leichtgewichtiges Gruppenmanagement-Protokoll mit dem Ziel entworfen, die erreichte, gute Leistungsfähigkeit durch den zusätzlichen Aufwand, der mit den Protokollabläufen verbunden ist, nur wenig zu schmälern. Es wurde argumentiert, daß der Verzicht auf starken Konsens und die

effiziente Benutzung von "Multicast"-Kommunikation die wesentlichen Mechanismen zum Erreichen der Zielvorgabe sind. Dieses innovative Protokoll, das auf die wesentliche Aufgabe des Gruppenmanagements reduziert ist, wurde mit Standardmethoden spezifiziert. Es kamen semiformale Beschreibungen der Protokollabläufe mit Hilfe von "Message Sequence Charts" zum Einsatz, die zu einer formalen Spezifikation mit endlichen Automaten und der "Specifikation and Description Language" (SDL) führten (vgl. auch Anhang B). Schließlich konnte gezeigt werden, daß die Verfügbarkeit durch den zusätzlichen Einsatz dieses Protokolls enorm gesteigert werden kann. Anstelle eines hochgradig ausfallanfälligen "n aus n"-Systems erhält man ein hochverfügbares "1 aus n"-System, bei dem jede der n Instanzen als aktive (funktionsbeteiligte) Redundanz für alle anderen Instanzen fungiert. Erneut zeigte sich, daß die Parallelverarbeitung ein geeigneter Mechanismus ist, diesmal zum Erreichen einer erhöhten Verfügbarkeit.

In einer eingehenden Sicherheitsanalyse, wurde in Kapitel 8 aufgrund von allgemeinen Sicherheitsanforderungen an Gruppenmanagement-Protokolle gezeigt, daß das zuvor spezifizierte Protokoll zu neuen Sicherheitsrisiken führt. Es wurden zusätzliche Mechanismen für eine Authentifikation entwickelt, die Erfolgsaussichten für "Denial of Service"-Angriffe berechnet und eine aufwandsarme Angriffserkennung entwickelt, die das Protokoll zusätzlich absichern. Darüber hinaus wurde für den Fall eines erkannten "Denial of Service"-Angriffs ein zusätzlicher Zustand eingeführt, in dem sich das Protokoll weitestgehend passiv verhält, um dem Angriff entgegenzuwirken und wichtige Betriebsmittel für die Hauptfunktion der Packet Screen (Zugriffskontrolle) freizugeben. Es wurde gezeigt, daß gerade Methoden zur Sicherheitsanalyse in verteilten Systemen bisher noch nicht in Standardwerkzeuge bzw. in Standardspezifikationssprachen integriert sind. Es wurde daraufhin eine Methode entwickelt, um verschiedene formale Ansätze zur Beschreibung von Sicherheitsproblemen bei der Protokollverarbeitung in "Message Sequence Charts" zu integrieren. Dadurch ist eine Sicherheitsanalyse bereits während der Protokollspezifikation mit Standardwerkzeugen durchführbar. Am Beispiel wurde gezeigt, daß durch diesen gleichzeitigen Einsatz der formalen Methoden im Zuge der Standardspezifikation eine elegante Abwägung zwischen alternativen Sicherheitsmechanismen in Hinblick auf ihren Betriebsmittelverbrauch gelingt (vgl. auch Abschnitt A). Dies ist ein wichtiger Beitrag für die integrierte Sicherheits- und Leistungsanalyse im Vorfeld einer Implementation.

In Kapitel 9 wurde schließlich die Vorgehensweise bei der Implementation des Prototypen der überlebensfähigen, parallelen Packet Screen dokumentiert. Es wurde gezeigt, daß auch bei der Implementation sorgfältig auf Verfügbarkeit, Leistungsfähigkeit und Sicherheit geachtet werden muß. Die Verfügbarkeit kann durch Programmierfehler gefährdet werden. Exemplarisch wurde das Vermeiden und Auffinden von Speicherlecks, die unweigerlich zu Verfügbarkeitsproblemen führen würden, mit Hilfe von Programmiermethodik und Werkzeugunterstützung dargestellt. Es wurde gezeigt, daß bei der Implementation auch auf die Rechnersicherheit geachtet werden muß und wie ein Teil der hierfür erforderlichen Sicherheitsüberprüfungen mit Hilfe von Werkzeugen unterstützt werden kann. Schließlich wurde das Testen von Softwareprodukten mit Hilfe von Werkzeugen am Beispiel dargestellt. Es konnte der besonders wichtige zeitliche Ablauf des Gruppenmanagement-Protokolls nach-

vollziehbar getestet werden. Eine abschließende Leistungsanalyse konnte zeigen, daß die gleichzeitige Erhöhung der Leistungsfähigkeit und Verfügbarkeit gelingen kann, sogar dann, wenn zusätzliche Sicherheitsprobleme durch diese Verbesserungen ebenfalls behandelt werden. Damit wurde eine durchgängige Darstellung des "Tradeoffs" zwischen Sicherheit, Verfügbarkeit und Leistungsfähigkeit von der ersten Evaluation über die Spezifikation bis hin zur Implementation an einem wichtigen Beispiel dargestellt.

In Kapitel 10 konnte darüber hinaus gezeigt werden, daß sich diese Ergebnisse mit relativ geringem Aufwand an andere Dienste in verteilten Systemen anpassen lassen. Obwohl am Beispiel der Packet Screen durchgeführt, ist sowohl die Parallelisierungsmethode als auch das leichtgewichtige Gruppenmanagement für eine Vielzahl weiterer Dienste einsetzbar. Dies führt zu überlebensfähigen Bausteinen für verteilte Systeme, die in Hinblick auf ihre spezielle Anwendungssituation miteinander kombiniert werden können.

Es zeigte sich somit, daß eine ganzheitliche Sichtweise, bei der Verfügbarkeit, Leistungsfähigkeit und Sicherheit gleichermaßen betrachtet werden müssen, nicht nur möglich, sondern auch in vielen Fällen sinnvoll und praktikabel ist. Hierbei können weitestgehend Standardtechniken, sowohl bei der Analyse als auch bei der Synthese, verwendet werden. Notwendige Erweiterungen an Werkzeugen und formalen Methoden scheinen mit moderatem Aufwand realisierbar zu sein (vgl. Anhänge).

11.2 Offene Fragestellungen

Es bleibt zu klären, ob die gewählte Vorgehensweise auch auf andere Komponenten übertragbar ist. Für diese Übertragbarkeit spricht die dargestellte Verwendung von Standardtechniken, sowohl für Analyse, Spezifikation als auch für die Implementation. Gegen diese allgemeine Übertragbarkeit der Ergebnisse sprechen im wesentlichen die aus pragmatischen Gründen selbst auferlegten Beschränkungen aus Kapitel 2. Bei vielen realen Systemen dürfte eine erste Analyse bzw. Evaluation der "scenarios" multiple, verschiedenartige Defizite aufzeigen. Es ist fraglich, ob dann die relativ zielgerichtete Leistungsoptimierung mit der iterativen Auflösung der Verfügbarkeits- und Sicherheitsprobleme, in der hier vorgeführten Methodik anwendbar ist, oder ob gleichzeitig multiple Defizite eines Systems verbessert werden müssen. Diese Befürchtung wird dadurch verstärkt, daß durch die Wahl der Packet Screen als Beispiel für die Überführung a priori von vielen "Benutzereffekten" ("usage scerarios") abgesehen werden konnte. Die Packet Screen arbeitet weitgehend auf Netz- und Transportebene, unabhängig von anderen Anwendungen, auf eigenen Systemen.

Weiterhin sind die formalen Methoden zur Beschreibung von Sicherheitsaspekten bisher noch unvollständig. Es ist z.Z. nicht möglich zu beweisen, daß ein bestimmtes Protokoll in allen Szenarien "sicher" ist. Zur Zeit kann man lediglich zeigen, daß ein Protokoll unter Annahme bestimmter Szenarien und unter Einsatz bestimmter Sicherungsmechanismen diesen Angriffen widerstehen kann. Somit ist das Antizipieren von möglichen Angriffen nach wie vor im wesentlichen abhängig von der Erfahrung und Vorstellungskraft des Sicherheitsex-

perten. Dies ist eine der Motivationen für die in Abschnitt 2 empfohlene, ständige Anpassung und Entwicklung des überlebensfähigen Systems. Neue Erkenntnisse, insbesondere zu neuen Angriffsmöglichkeiten, müssen zu einer Anpassung des Systems führen.

Weiterhin wurde in der vorliegenden Arbeit auf hochgradig spezialisierte Analysen verzichtet. Beispielsweise wurde keine Kryptoanalyse für den verwendeten Authentifikationsmechanismus des Gruppenmanagement-Protokolls durchgeführt. Es müßte beispielsweise geklärt werden, ob und unter welchen Umständen ein Wechsel des Schlüssels vor Erreichen der größtmöglichen Sequenznummer notwendig ist. Dies muß in Abhängigkeit vom verwendeten Hash-Algorithmus, der Schlüssellänge und dem tatsächlichen Nachrichtenaufkommen (wiederum abhängig vom Timing und der Anzahl der Instanzen) erfolgen. Das skizzierte Vorgehen verdeutlicht, daß der Grad der durchzuführenden Analysen weitgehend unklar ist und fast beliebig viel Aufwand in Analysen gesteckt werden kann. Dennoch werden wohl auch bei überlebensfähigen Systemen immer Restrisiken und Unsicherheiten übrig bleiben, die durch organisatorische Maßnahmen aufgefangen werden müssen.

Anhang A

Spezifikation mit "Message Sequence Charts" und "Specification and Description Language"

In diesem Anhang wird eine kurze Einführung in die graphische Notation der "Message Sequence Charts" (MSCs) und der "Specification and Description Language" (SDL-GR) gegeben. Lesern ohne Kenntnisse dieser Notationen soll damit die Interpretation der Spezifikation aus Anhang B und der Sicherheitsanalysen aus Kapitel 8 ermöglicht werden. Ausführlichere Darstellungen und Details werden beispielsweise in [81,125,123,124,122,121,118] gegeben.

In Abschnitt A.2 werden hierauf aufbauend neue Beschreibungsmöglichkeiten für eine Sicherheitsanalyse, basierend auf der MSC-Notation, diskutiert. Für das Verständnis der SDL-Spezifikation des Gruppenmanagement-Protokolls in Anhang B sind die hier vorgestellten Grundkenntnisse ebenfalls erforderlich.

A.1 Einführung in die graphische Notation der "Message Sequence Charts" und der "Specification and Description Language"

Die "Message Sequence Charts" (MSCs) sind graphische Notationen für den Nachrichtenaustausch zwischen Systemkomponenten und ihrer Umgebung. Sie sind eng mit der SDL-basierten Protokollspezifikation verbunden, da mit MSCs häufig die grundlegenden Kommunikationsabläufe als Basis für die Protokollspezifikation mit SDL modelliert werden.

Mit automatisch aus der SDL-Spezifikation erstellten Modellen können dann Kommunikationsabläufe simuliert werden, die konform mit der Spezifikation sind. Typischerweise

werden diese automatisch generierten Abläufe wieder als MSCs dargestellt (beispielsweise die Abbildungen 7.5 – 7.7). Bei der Validation von Protokollspezifikationen können somit MSCs und SDL-basierte Protokollspezifikationen kombiniert werden, um zu zeigen, daß bestimmte Nachrichtenabläufe entweder konform mit der Spezifikation sind oder gegen die Spezifikation verstoßen.

A.1.1 Einführung in “Message Sequence Charts”

Die wesentlichen Elemente eines MSCs sind Symbole für Systemkomponenten und Pfeile für den Nachrichtenaustausch zwischen diesen Komponenten und der Umgebung. Da in vielen Fällen eine Implementierung dieser Systemkomponenten als Prozesse erfolgt, sind weitere graphische Elemente in den Notationen zu finden, die (Prozeß-)Zustände, Anweisungen, Prozeduraufrufe und Zeitgeber-Ereignisse darstellen.

In Abbildung A.1 sind die wichtigsten Symbole dargestellt. Es gibt eine Instanz einer Systemkomponente mit dem Namen “Name der Systemkomponente”. Diese Instanz ist von einem bestimmten Typ, der ebenfalls angegeben ist: “Typ der Systemkomponente”. Durch den zusätzlichen Namen können mehrere Systemkomponenten desselben Typs unterschieden werden. Die Symbole mit den Inschriften “Zustand” und “Folgezustand” stellen Zustände der Systemkomponente dar. Durch verschiedene Zustände können kausale Zusammenhänge dargestellt werden, beispielsweise der Zustandswechsel aufgrund einer eintreffenden Nachricht. Nachrichten werden mit Pfeilen dargestellt. Der Pfeil zeigt vom Sender zum Empfänger, die Beschriftung des Pfeils gibt den Typ der Nachricht und ggf. in Klammern dahinter die Parameter der Nachricht an. Nachrichten können mit der Umgebung oder mit anderen Instanzen ausgetauscht werden. Nachrichten von bzw. zu der Umgebung werden mit Pfeilen gekennzeichnet, die an den Rand der Abbildung gehen.

Der zeitliche Ablauf der Kommunikation wird bei MSCs durch die vertikale Anordnung der Symbole verdeutlicht. Beispielsweise findet die lokale Berechnung (als Rechteck dargestellt) vor dem Senden der Nachricht vom Typ “Nachricht_B” statt, jedoch nach dem Empfang der Nachricht vom Typ “Nachricht_A”. Die Berechnung erfolgt somit im Zustand “Zustand”, da das Symbol für den Zustandswechsel über dem Versenden der Nachricht angeordnet ist. Eine Instanz bleibt so lange in einem Zustand, bis auf der vertikalen Zeitachse ein weiterer Zustand angegeben wird (hier beispielsweise der Zustand “Folgezustand”).

Durch die Aufteilung des Gesamtablaufs in mehrere Einzelabbildungen, die an den Zuständen mit gleichem Namen zusammengefügt werden können, kann mit einer begrenzten Anzahl von MSCs eine unendliche Anzahl von Protokollabläufen dargestellt werden. In den folgenden Abbildungen ist dies durch zwei alternative Abläufe aus dem Zustand “Folgezustand” dargestellt. Es wird ein Zeitgeber gestellt und auf eine Nachricht von der Umgebung gewartet. Trifft die Nachricht “Nachricht_C” rechtzeitig ein, dann wird der Zeitgeber zurückgesetzt und in den neuen Zustand gewechselt. In Abbildung A.2 ist dies der schon bekannte Zustand “Zustand”.

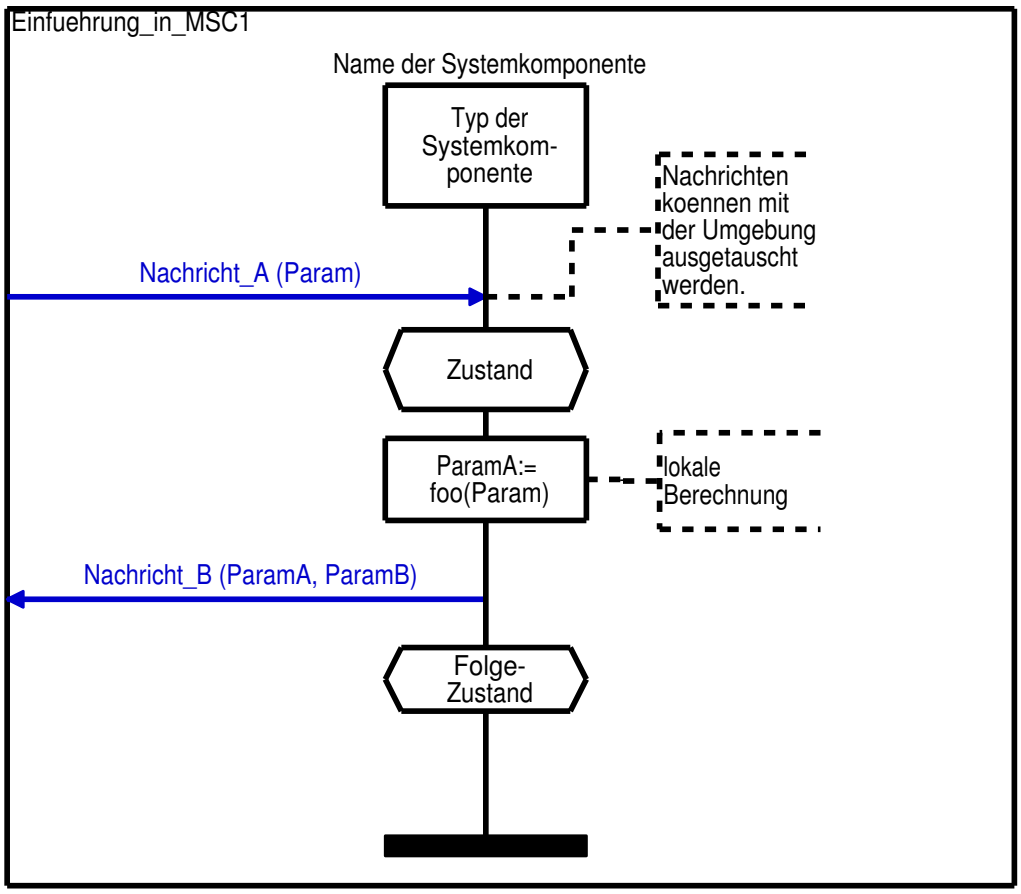


Abbildung A.1: Beispiel MSC 1/3

Alternativ kann aber auch ein Zeitgeber-Ereignis eintreten und die Instanz folgt dann einem anderen Ablauf. Beispielsweise terminiert die Instanz beim Ablaufen des Zeitgebers in Abbildung A.3. Die Termination ist durch ein Kreuz am Fuß des Instanzsymbols gekennzeichnet.

A.1.2 Einführung in die “Specification and Description Language”

SDL ermöglicht eine graphisch/formale Spezifikation von verteilten Systemen. Grundlage ist ein formales Modell, das die Semantik der hier vorgestellten Piktogramme eindeutig und vollständig definiert [126].

SDL ist in Hinblick auf einen “Top-Down”-Entwurf konzipiert worden. Ein System ist in Blöcke unterstrukturiert, die untereinander und mit der Systemumgebung durch Kanäle (“channels”) verbunden sind. Kanäle transportieren Nachrichten (“signals”) von und zu

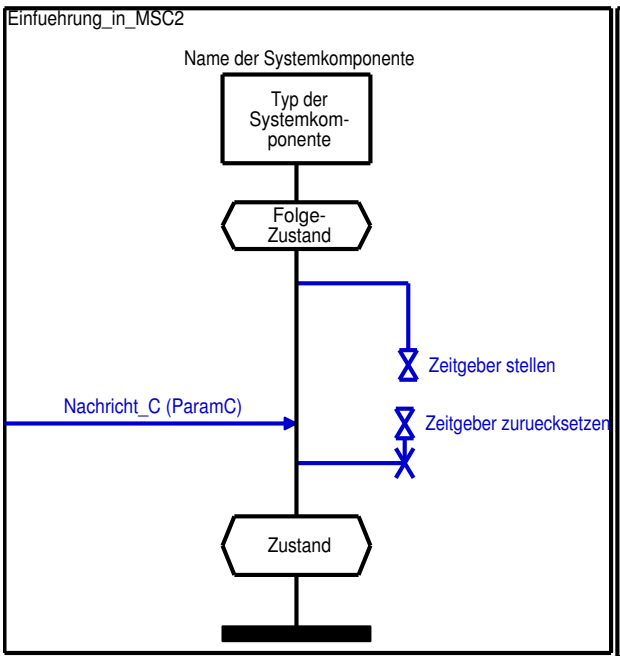


Abbildung A.2: Beispiel MSC 2/3

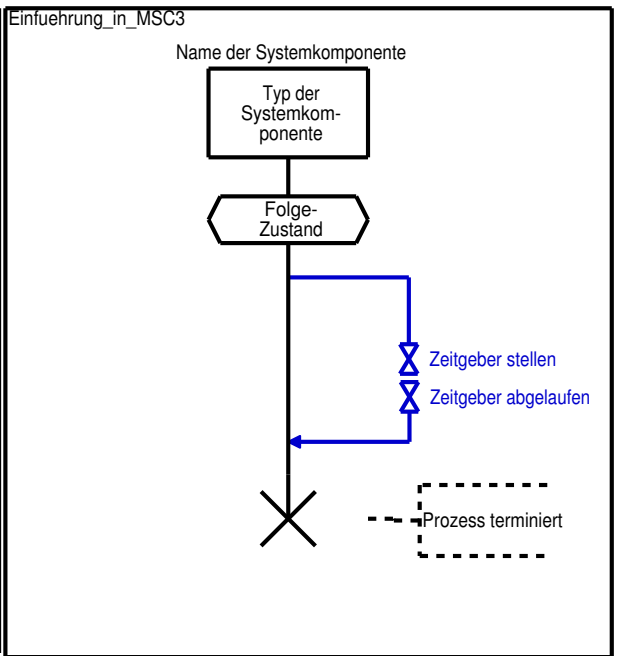


Abbildung A.3: Beispiel MSC 3/3

den Blöcken. Kanäle sind unidirektional oder bidirektional, verzögern die Nachrichten um eine nicht-deterministische Zeitdauer und können nur ihnen ausdrücklich zugeordnete Nachrichten transportieren. Abbildung A.4 zeigt ein System bestehend aus zwei Blöcken. Die Blöcke sind untereinander mit einem Kanal verbunden. Ein weiterer Kanal verbindet einen Block mit der Systemumgebung.

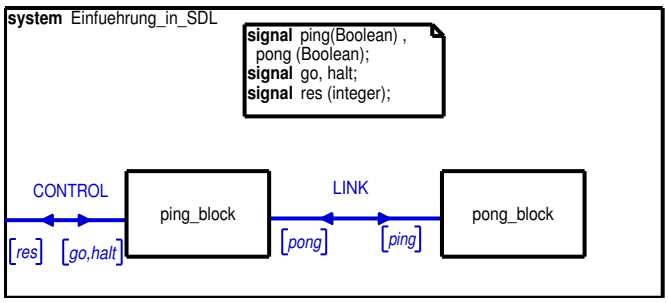


Abbildung A.4: SDL Systemstruktur

Blöcke können beliebig weiter durch (Sub-)Blöcke verfeinert werden. Auf jeder Blockebene können wieder neue Blöcke angegeben werden, die ihrerseits mit Kanälen verbunden sind. Auf der Blockebene (jedoch nicht auf der Systemebene) können auch Prozesse eingetragen werden. Mehrere Prozesse im selben Block werden mit nicht-verzögernden Kanälen (“routes”) verbunden. Diese “routes” können an der Blockgrenze den Kanälen zugeordnet

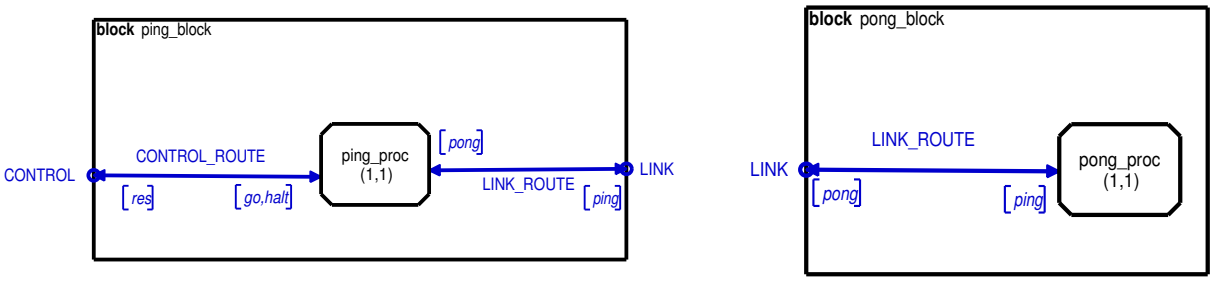


Abbildung A.5: SDL-Blockstruktur

werden. Die Zuordnung von Kanälen zu “routes” kann $1 : n$ erfolgen. Hierdurch ist eine Aufteilung der Signale, die über einen Kanal zu einem Block gelangen, auf verschiedene Prozesse im Block möglich. Abbildung A.5 zeigt die Verfeinerung der beiden Blöcke.

Prozesse können wiederum durch Prozeduren (Unterprogramme) verfeinert werden. Alle Prozesse im System laufen nach ihrem Start bis zur Termination mit allen anderen Prozessen im System parallel ab. Prozesse können weitere Prozesse zur Laufzeit dynamisch generieren.

Prozesse entsprechen erweiterten, endlichen Automaten. Die zu empfangenden Nachrichten und Zeitgeberereignisse kann man sich als Eingabesymbole des Automaten vorstellen. Die Menge der zu versendenden Nachrichten entspricht dem Ausgabealphabet des Automaten. Es handelt sich formal um einen *erweiterten*, endlichen Automaten, da SDL-Prozesse auf eine beliebige (aber festgelegte) Anzahl von Variablen lesend und schreibend zugreifen können. SDL kennt die üblichen Typen prozeduraler Programmiersprachen, beispielsweise ganze Zahlen (“INTEGER”), Wahrheitswerte (“BOOLEAN”) und bietet darüber hinaus die Möglichkeit, eigene, abstrakte Datentypen zu erstellen (“NEWTYPE”). Variablen sind auch in umschlossenen Blöcken sichtbar, beispielsweise auf der Blockebene, obwohl sie auf der Systemebene deklariert sind. Da der Wertebereich dieser Variablen nicht beschränkt ist, ergibt sich somit ein potentiell unbeschränkter Zustandsraum.

Hieraus lassen sich unmittelbar die wesentlichen Piktogramme von SDL ableiten: Es gibt Symbole für Eingabe, Ausgabe, Zeitgeber sowie lokale Berechnungen mit Zugriff auf Variablen. Diese Symbole sind miteinander verbunden, um den kausalen und zeitlichen Ablauf bei den Zustandsübergängen zu verdeutlichen. Zustandsübergänge werden durch eintreffende Nachrichten ausgelöst. Es werden dann alle Aktionen bis zum nächsten Folgezustand ausgeführt. Der SDL-Automat verbleibt im Folgezustand, bis eine weitere Nachricht eintrifft oder ein Zeitgeberereignis eintritt. Einzige Ausnahme ist der Startzustand, aus dem auch ohne eintreffende Nachrichten ein Zustandsübergang in einen Folgezustand möglich ist. Zusätzliche Kommentare können durch Kommentarsymbole an jedes andere Piktogramm angebracht werden. Abbildungen A.6 und A.7 zeigen die Verfeinerung der Prozeßsymbole aus Abbildung A.5. Zustände sind als Ovale dargestellt. Der Pfeil ist das Sendesymbol, der Wimpel steht für den Nachrichtenempfang. Nachrichten können von allen Prozessen oder von der Umgebung stammen oder intern durch ablaufende Zeitgeber

generiert werden. Die Inschriften in beiden Symbolen geben das Signal (oder den Zeitgeber) und deren Parameter an und benennen ggf. die zu verwendende “Route” und den Empfangsprozeß. Beim Stellen von Zeitgebern kann auf die aktuelle Zeit zugegriffen werden. Die aktuelle Zeit wird in der implizit deklarierten Variable “NOW” geführt. Ein mit dem Minuszeichen markierter Folgezustand bedeutet, daß ein Zustandsübergang in denselben Zustand erfolgt. Entscheidungen (Rautensymbol) können in Abhängigkeit von den Werten der Variablen oder eintreffenden/ausbleibenden Nachrichten gefällt werden (alternative Nachrichten/Zeitgeber-Wimpel unter einem Zustand). Dadurch sind Verzweigungen in den Zuständen möglich.

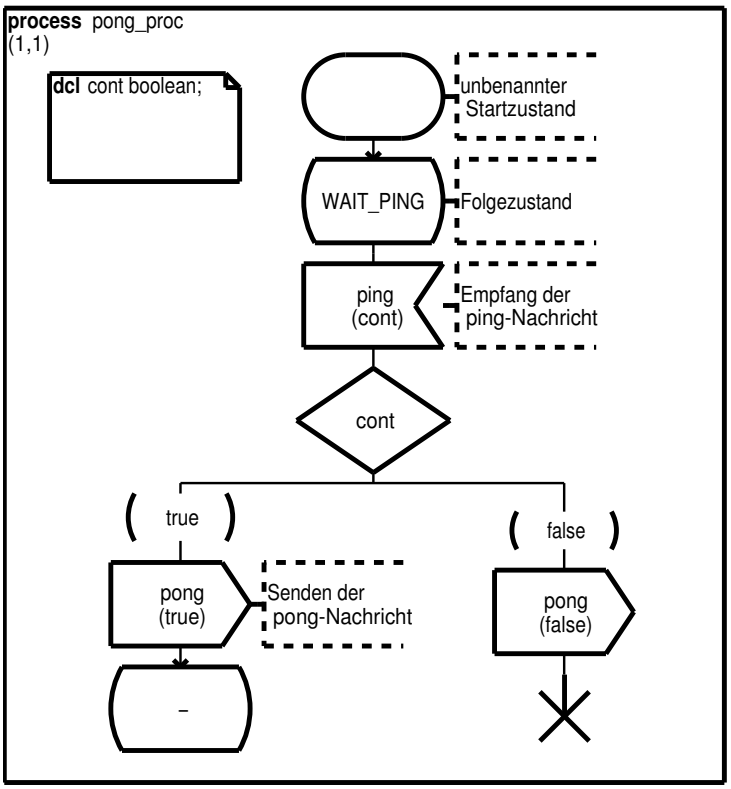


Abbildung A.6: SDL Prozeßstruktur “pong_proc”

Die Spezifikation beschreibt ein einfaches Protokoll zwischen zwei Instanzen. Eine Instanz (Prozeß “PING”) sendet “Ping”-Nachrichten an die andere Instanz (“Pong”-Prozeß) und zählt die von ihr zurückgesendeten “Pong”-Nachrichten. Der Parameter “TRUE” der Nachrichten zeigt an, daß noch weitere Nachrichten gesendet werden. Läuft der Zeitgeber ab, wird schließlich eine “Ping”-Nachricht mit dem Parameter “FALSE” gesendet. Dies veranlaßt den “Pong”-Prozeß, nach einer finalen Antwort (“Pong”-Nachricht, Parameter “FALSE”) zu terminieren. Der “Ping”-Prozeß sendet die Anzahl der empfangenen “Pong”-Nachrichten an die Umgebung und terminiert danach ebenfalls. Alternativ zum Ablauf des Zeitgebers kann

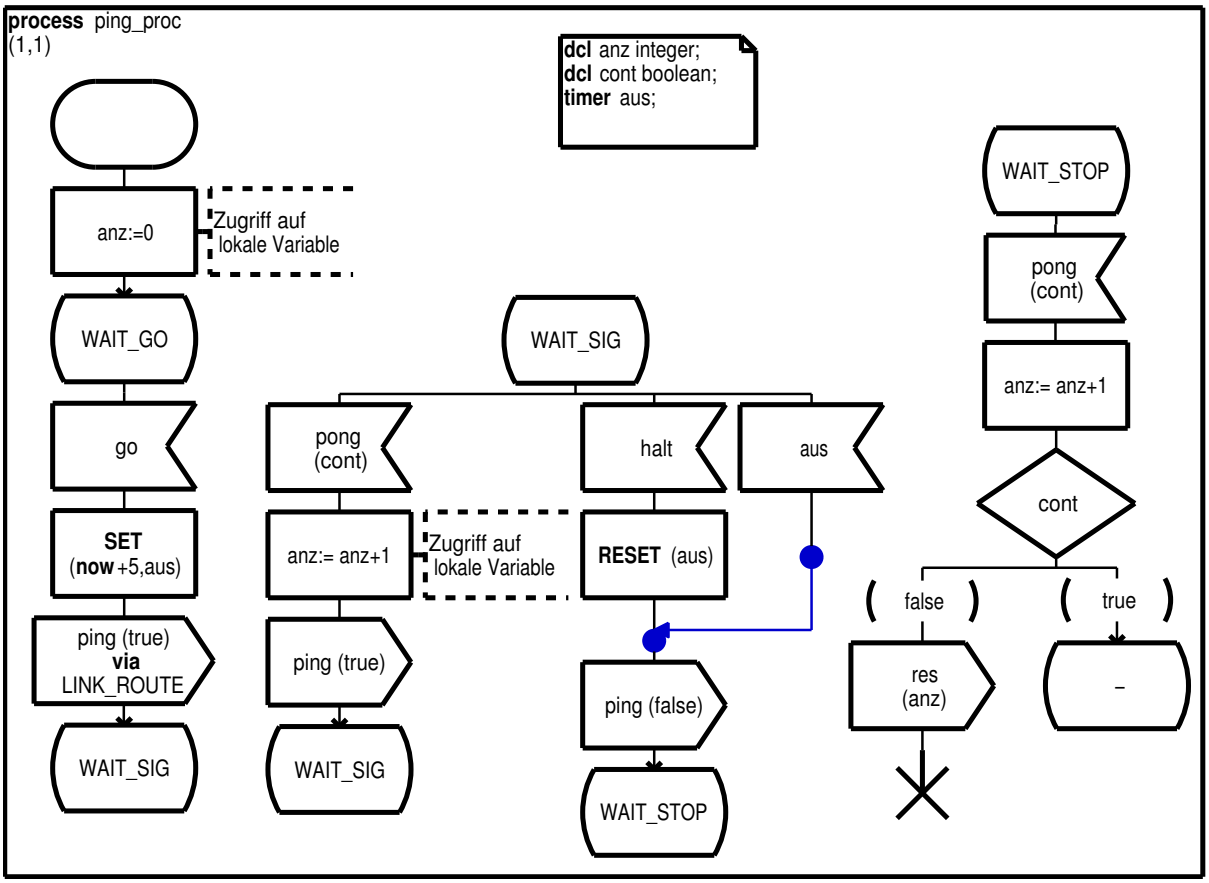


Abbildung A.7: SDL Prozeßstruktur “ping_proc”

die Termination auch durch eine “HALT”-Nachricht von der Umgebung ausgelöst werden. Zwei Abläufe, die mit einem aus der Spezifikation automatisch generierten Modell erstellt wurden, sind in Abbildung A.8 wiederum als MSCs dargestellt.

A.2 “Message Sequence Charts” als standardisierte, semiformale Methode zur Sicherheitsanalyse von Protokollen

In diesem Abschnitt soll die Methode zur Protokollanalyse mit MSCs (vgl. [122, 125]) verallgemeinert werden. In der vorliegenden Arbeit werden MSCs an vielen Stellen eingesetzt, um grundsätzliche Kommunikationsabläufe darzustellen. MSCs bieten dazu eine Reihe von definierten Symbolen, um beispielsweise:

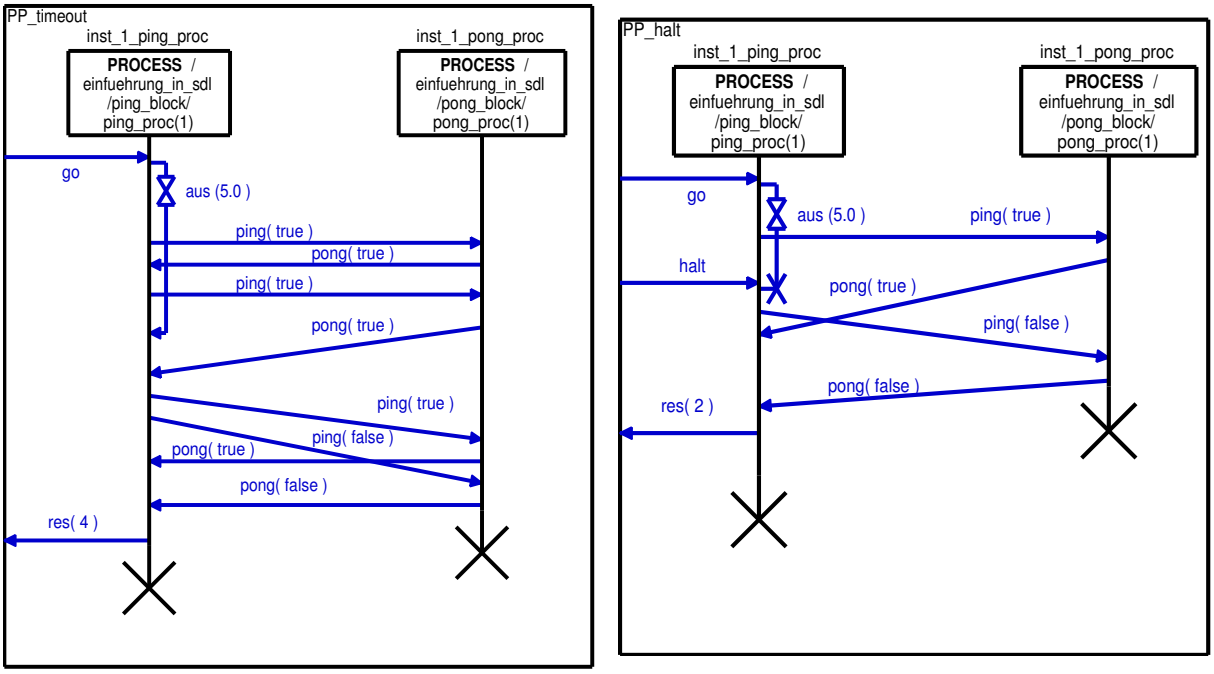


Abbildung A.8: Zwei Abläufe des Beispielsystems

- verschiedene Kommunikationspartner darzustellen,
- lokale Zustände von globalen Zuständen zu unterscheiden,
- kausale Zusammenhänge zwischen eintreffenden Nachrichten, internen Ereignissen und ausgehenden Nachrichten aufzuzeigen,
- explizit das Timing zu modellieren.

Der Hauptvorteil bei der Verwendung von MSCs ist die Standardisierung dieser halbformalen Beschreibungssprache durch die ITU-T in den zitierten Empfehlungen (insbesondere in [122, 125]). Hiervon profitiert die Verbreitung dieser Beschreibungssprache.

In diesem Abschnitt soll daher gezeigt werden, wie verschiedene Konnotationen auf einfache Weise in die MSCs integriert werden können. Der grundsätzliche Vorteil dieser Vorgehensweise ist, daß die verschiedenen Analysemethoden integriert werden und somit ein Bruch in der Darstellung vermieden werden kann: Die Sicherheitsanalyse des Protokolls wird in die Protokollspezifikation eingegliedert.

Abschnitt A.2.1 beschreibt die Integration der BAN-Logik in die MSCs. Abschnitt A.2.2 beschreibt die Integration von “Strands” in MSCs und schließlich zeigt Abschnitt A.2.3 die Integration von Annotationen bezüglich der Verarbeitungskosten in MSCs.

Ein weiterer Vorteil wird in Abschnitt A.2.4 dargestellt. Die MSCs ermöglichen auch den kombinierten Einsatz mehrerer Annotationen.

A.2.1 “Message Sequence Charts” und BAN-Logik

Die BAN-Logik [41] und ihre Erweiterung durch die GNY-Logik [99] sind weit verbreitet und werden häufig zur Analyse von Protokollen mit Sicherheitsfunktionalität eingesetzt. Trotz vielfältiger Kritik haben sich diese beiden Analysemethoden durchaus zur Protokollvalidation etabliert.

Nach Auffassung des Autors ist es aber wichtig, in Hinblick auf eine Protokollspezifikation, die eine Implementierung eines kryptographischen Protokolls unterstützen soll, bereits die Sicherheitsanalyse in einer Darstellung vorzunehmen, die als Grundlage für die spätere Spezifikation des Zustandsautomaten geeignet ist. Dies ist bei der BAN-Logik besonders leicht und auch naheliegend: Die BAN-Logik versucht, den *korrekten* Protokollablauf zu beschreiben und anschließend Rückschlüsse auf das erreichte Wissen der Kommunikationspartner zu deduzieren. Dieses Vorgehen ist auf der Ebene der Protokollspezifikation ganz ähnlich. Auch dort werden in der Regel erst die *erwünschten* Kommunikationsabläufe skizziert und anschließend um zusätzliche Fälle (in der Regel die antizipierten Fehlerfälle) erweitert.

In diesem Sinne muß es somit für jeden mit der BAN-Logik dargestellten *korrekten* Protokollablauf einen *erwünschten* Protokollablauf geben, der typischerweise vor der Spezifikation des SDL-Automaten mit MSCs dargestellt wird.

Darüber hinaus läßt sich ein wesentlicher Kritikpunkt an der BAN-Logik und deren Erweiterungen durch den kombinierten Einsatz mit MSCs ausräumen. In Abschnitt 8.4.1 wurde als wesentliche Kritik formuliert, daß der Abstraktionsprozeß, bei dem die informale Protokollspezifikation in die formale BAN-Darstellung überführt wird, sehr fehleranfällig ist. Ein Grund für diese Fehleranfälligkeit ist, daß bei der Überführung von vermeintlich unwichtigen Protokollaspekten abstrahiert wird. Dies kann besonders leicht passieren, wenn die Überführung in die BAN-Notation *erst nach der Spezifikation* des Protokolls als eigenständiger Analyseschritt stattfindet.

Sinnvoller ist es, im Zuge der Protokollsynthese die Überführung durchzuführen. Der Vorteil daran ist, daß alle Aspekte der Syntax, der Semantik und des Timings des Protokolls im Zuge der Protokollsynthese dokumentiert und spezifiziert werden müssen. Bei der Überführung in die BAN-Notation ist somit offensichtlich, *unter welchen Randbedingungen* von bestimmten Protokollaspekten abstrahiert werden kann. Die Sicherheitsanalyse darf dann natürlich nur unter diesen und nicht unter anderen Randbedingungen durchgeführt werden. Andernfalls wäre die Überführung in die BAN-Logik in der Tat (retrograd) fehlerhaft, da sie das Szenario, das bei der Sicherheitsanalyse angenommen wird, nicht abdeckt.

Andererseits könnte die Überführung in die BAN-Logik auch ergeben, daß bestimmte Teile der Protokollspezifikation in der Tat *redundant* sind, d.h., daß unabhängig von den Randbedingungen, von diesen Protokollaspekten abstrahiert werden kann. In diesem Fall führt die Überführung in die BAN-Logik zu einer Verbesserung des Protokollentwurfs, da die redundanten Aspekte aus der Spezifikation herausgenommen werden können.

In jedem Fall kann aber bei der kombinierten MSC/BAN-Spezifikation bereits festgelegt werden, wie bestimmtes Wissen der BAN-Logik durch den Protokollautomaten hergestellt werden soll. Ist es beispielsweise erforderlich, daß zu einer Nachricht ($Header, Inhalt, Trailer$) der Kommunikationspartner den Inhalt identifizieren kann (GNY-Logik: $\phi(Inhalt)$), dann kann und sollte das MSC die hierfür durchzuführenden Prozeduren benennen (beispielsweise “Auswerten von $Header$ liefert Typ und Länge von $Inhalt$.”). Dieser Vorgang muß durch eine formale Spezifikation zwar noch im Detail beschrieben werden, aber es wird bereits durch die MSC-Abläufe verdeutlicht, auf welche Informationen sich der Abstraktionsprozeß stützt. Beim obigen Beispiel ist somit ersichtlich, daß $\phi(Inhalt)$ direkt und ausschließlich von $Header$ abhängig ist.

Für die Integration ist es hilfreich, sich zu verdeutlichen, daß

- eingangs gemachte Annahmen und durch Deduktion neu erlangte Erkenntnisse durch Zustände oder Kommentare an die MSCs angebracht werden können;
- gemeinsames Wissen immer durch (Kommentare an) globale Zustände abgebildet werden kann (beispielsweise der vielbenutzte symmetrische Schlüssel K);
- abstrakte Nachrichten der BAN-Logik direkt als Beschriftungen der Nachrichten in MSCs verwendet werden können.

Das folgende Beispiel verdeutlicht die Vorgehensweise. In Abbildung A.9 ist die aus [41] vorgenommene Analyse für das Otway-Rees Protokoll als MSC dargestellt. In der MSC-Darstellung werden lediglich die Symbole für *Glauben* ($\models\!\equiv$) und *Sehen* ($\lhd$) ausgeschrieben. Dies ist aber auch in anderen Veröffentlichungen so üblich (vgl. [225, 160, 161]).

Anstelle die Nachrichtensymbole (Pfeile) mit den idealisierten Inhalten zu beschriften (wie in Abbildung A.9 dargestellt), können auch die tatsächlich als sinnvoll für eine Implementation erachteten PDU-Formate angegeben werden. In diesem Fall würden die idealisierten Nachrichten als Kommentare an die Nachrichtensymbole angebracht werden. Die Aussagekraft steigt dadurch weiter, da so auch unmittelbar der Abstraktionsprozeß verdeutlicht wird. Dies ist besonders wichtig, weil sich die meiste Kritik an der BAN-Logik und deren Erweiterungen auf diese informale Idealisierung (Abstraktionsprozeß) bezieht. Abbildung 8.2 zeigt ein Beispiel für die Analyse, bei der die idealisierten Nachrichteninhalte als Annotationen zum eigentlichen PDU-Format eingezeichnet sind.

A.2.2 “Message Sequence Charts” und “Strands”

Die in [91] vorgestellten “Strand Spaces” dienen neben der Darstellung der kausalen Zusammenhänge vor allem der Modellierung der Fähigkeiten von Angreifern. Der dort dargestellte Formalismus benennt “Strands” als lokal beobachtbare Folgen von empfangenen Nachrichten ($-M$) und gesendeten Nachrichten ($+M$). Kausale Zusammenhänge werden

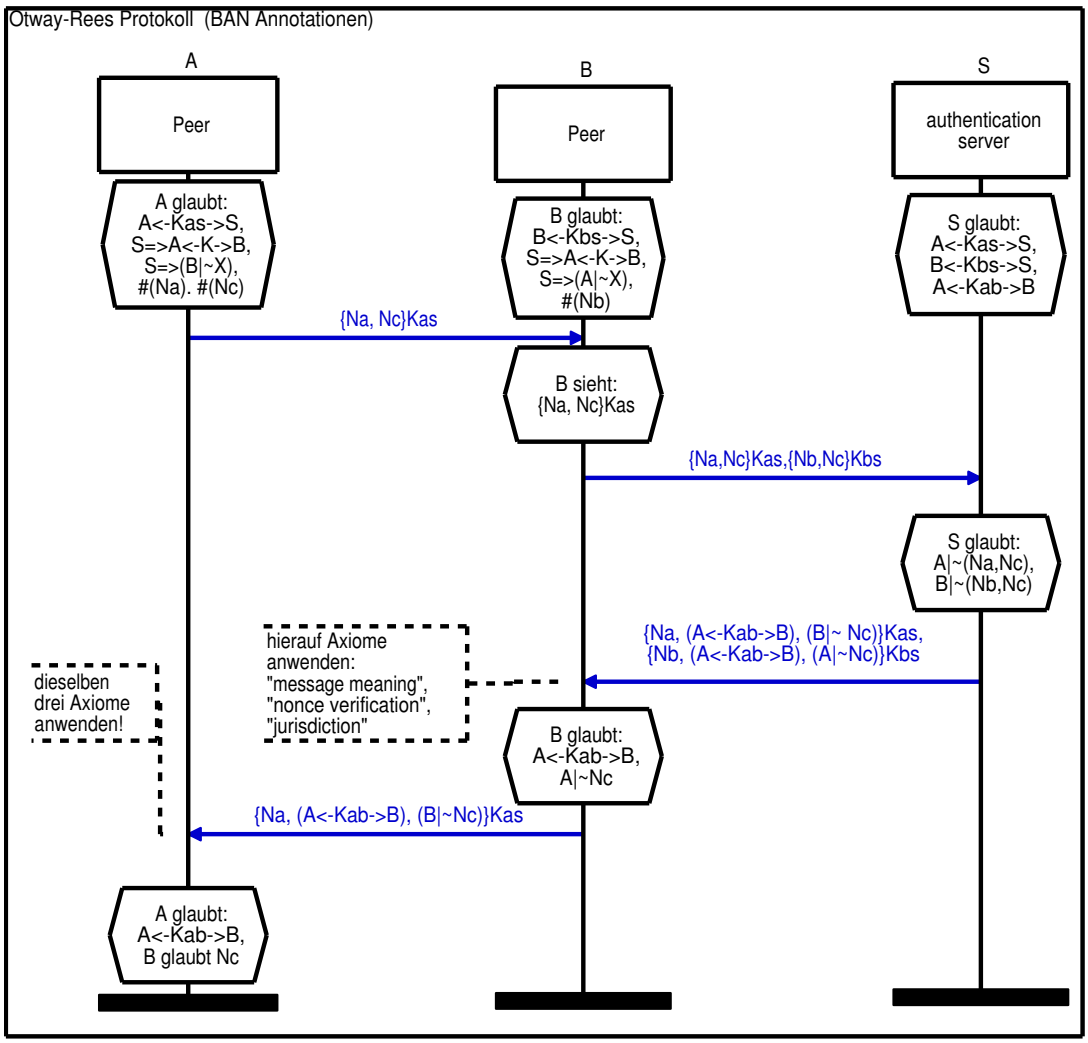


Abbildung A.9: MSC mit BAN-Logik Annotationen für das Otway-Rees Protokoll

modelliert durch die $\Downarrow$-Relation, die angibt, daß zwei Ereignisse kausal, unmittelbar nacheinander im selben Strand erfolgen. Die $\rightarrow$-Relation gibt an, daß es zu einer gesendeten Nachricht $-M$ einen passenden Nachrichtenempfang $+M$ gibt (also $n1 \rightarrow n2$ bedeutet, daß $term(n1) = +M, term(n2) = -M$ gilt). Diese Relation stellt somit kausale Zusammenhänge zwischen verschiedenen Strands her.

Für die Integration in MSCs ist hilfreich:

- Knoten desselben Strands lassen sich durch Zustände derselben Instanz abbilden.
- Die bei MSCs implizite Annahme über den zeitlichen Ablauf bildet (automatisch) die $\Downarrow$-Relation nach.

- Von der Instanz abgehende Nachrichtensymbole entsprechen den $+M$-Termen eines Strands.
- An einer Instanz ankommende Nachrichtensymbole entsprechen den $-M$-Termen eines Strands.
- Die Beschränkung auf die lokale Sicht (auf einen Strand) wird durch das Weglassen der Partnerinstanz(en) modelliert. Die Nachrichten werden dazu in den MSCs mit der Umgebung ausgetauscht.

In Abbildung A.10 und Abbildung A.11 werden zwei der in [91] diskutierten Fähigkeiten eines Angreifers modelliert. Als Beispiel werden hier folgende Fähigkeiten des Angreifers nachgebildet: Das Aufsplitten von Nachrichten und das Verschlüsseln von Klartexten mit einem abgefangenen Schlüssel und einer abgefangenen Nachricht.

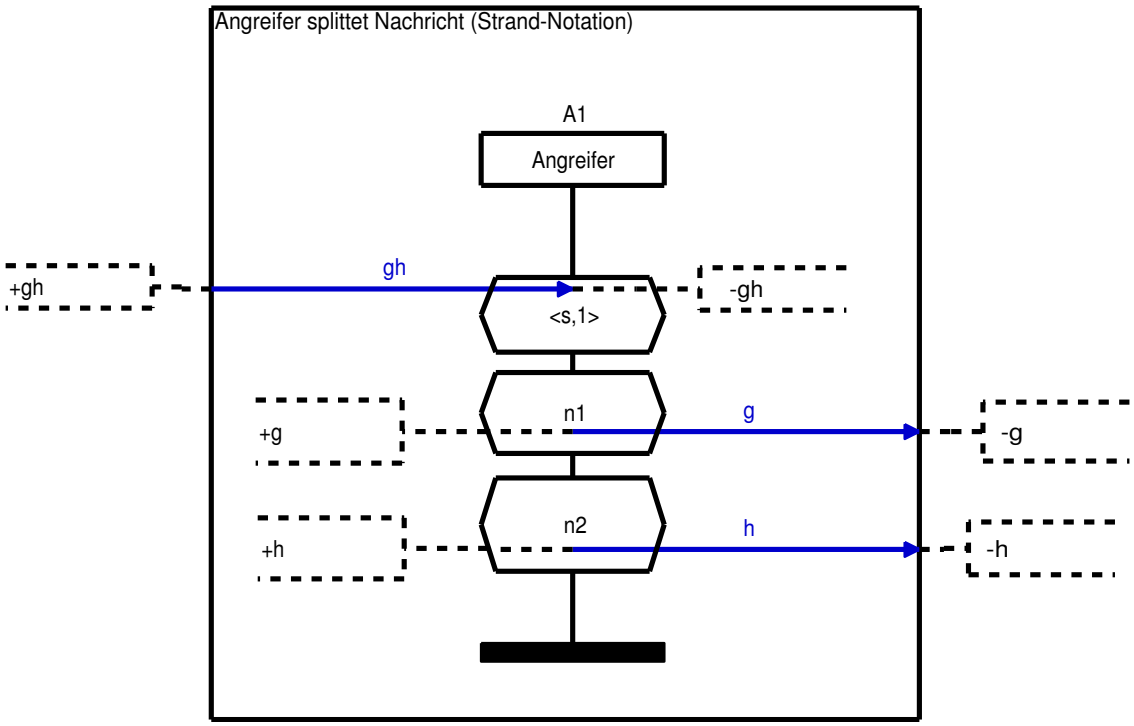


Abbildung A.10: MSC mit Angreifer-Strand für das Aufsplitten von Nachrichten

Auch in diesen Fällen sind zusätzliche Annotationen als Teil der Kommentare möglich.

A.2.3 “Message Sequence Charts” und Verarbeitungskosten

Obwohl die Verfügbarkeit als Schutzziel seit langem von Sicherheitsexperten anerkannt ist (beispielsweise [187]), werden formale Beschreibungen für die Bewertung des Aufwands von

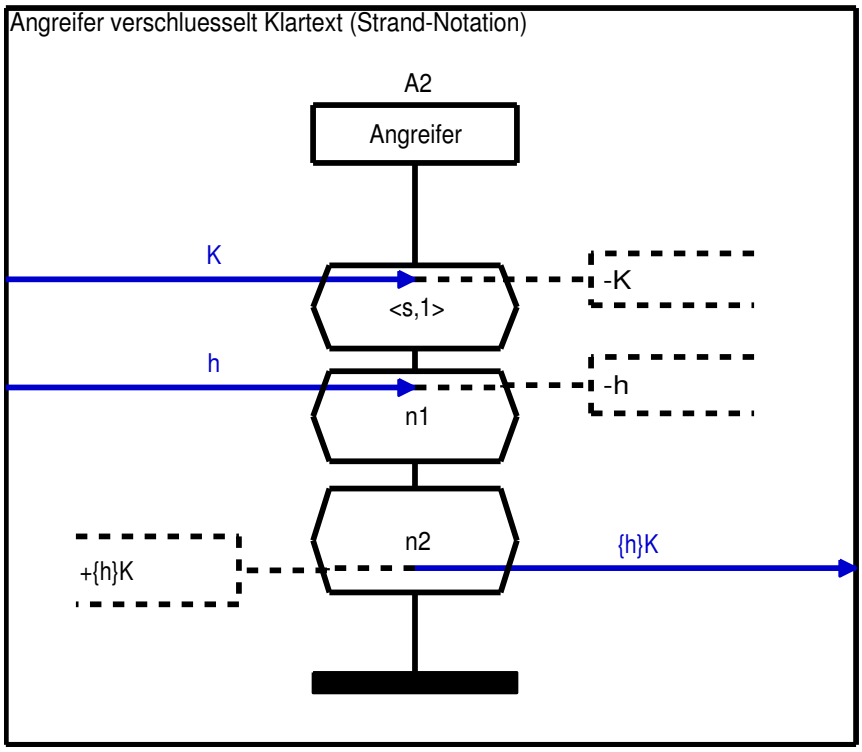


Abbildung A.11: MSC mit Angreifer-Strand für das Verschlüsseln von Nachrichten

Angriffen auf die Verfügbarkeit noch gesucht. Erste Ansätze dazu sind in [163] enthalten. Es wird versucht, Verarbeitungskosten für die Kommunikation anzugeben, um hierüber auf Angriffe auf die Verfügbarkeit schließen zu können. Dieser Ansatz hat weitere Vorteile: Er erfordert die Benennung von Verarbeitungskosten in Abhängigkeit vom Protokollverlauf. Dies ermöglicht bereits die Diskussion des Aufwands verschiedener Protokollalternativen im Zuge des Entwurfs und ist somit ein geeignetes Werkzeug zur Entscheidungsfindung bei alternativen Ansätzen.

Grundsätzlich werden Nachrichten (M) von Alice (A) nach Bob (B) in der üblichen Form dargestellt:

$$A \rightarrow B \ : \ M \tag{A.1}$$

Diese Notation wird nun um die Aktionen von Alice und Bob erweitert, so daß die Sequenz $P_1, \ldots, P_k$ die Aktionen beschreibt, die Alice ausführen muß, um die Nachricht zu generieren. Analog beschreibt die Sequenz $A_1, \ldots, A_l$ die Aktionen, die Bob durchführen muß, um die Nachricht auszuwerten. Man erhält somit eine in [163] vorgestellte, *kommentierte Alice-und-Bob Spezifikation* als Sequenz von Ausdrücken der Form:

$$A \rightarrow B \ : \ P_1, \ldots, P_k || M || A_1, \ldots, A_l \tag{A.2}$$

Der Aufwand für die Bearbeitung kann dann durch eine Kostenfunktion notiert werden. Diese Kostenfunktion (δ) bildet aus der Menge aller kostenverursachenden Aktionen (Berechnungen) auf entsprechende Kosten ab. Für den Empfang der Nachricht aus Formel A.2 entstehen somit die folgenden Kosten in Abhängigkeit von den Einzelaufträgen (A_i):

$$\delta'(A_l) = \delta(A_1) + \ldots + \delta(A_l) \tag{A.3}$$

Durch diesen Formalismus wird es möglich, den Aufwand für Sender, Empfänger und ggf. einen Angreifer abzuschätzen. Es wird eine weitere Funktion definiert, in der noch eventuelle Folgekosten aufgelistet sind, die über den Empfang der Nachricht hinausgehen. Dies sind Kosten, die durch das Verarbeiten der empfangenen Nachricht durch den Protokollautomaten entstehen.[1] Der Hauptnachteil an diesem Formalismus ist, daß die Notation schnell unübersichtlich wird. Wenn man versucht, die Aktionen mit verständlichen Namen zu benennen, ergeben sich mehrzeilige Akronyme. Darüber hinaus ist in diesen Annotationen nicht unmittelbar ersichtlich, ob die Aktionen (und somit die Kosten) zum Nachrichtenempfang oder zu den darauffolgenden Verarbeitungskosten zu zählen sind.

Auch in diesem Fall sind die MSCs bestens geeignet, um sowohl eine übersichtliche Darstellung zu erreichen, als auch genau zu definieren, an welcher Stelle[2] die Kosten anfallen:

- Die einzelnen Aktionen des Senders und des Empfängers (P_i, A_j) können direkt als MSC-Tasks in die Instanzen eingezeichnet werden.
- MSC-Tasks vor einem abgehenden Nachrichtensymbol beschreiben in ganz natürlicher Weise die Aktionen für die Generierung von Nachrichten ($P_1, \ldots, P_k$).
- MSC-Tasks unter einem eintreffenden Nachrichtensymbol beschreiben in ganz natürlicher Weise die Aktionen für den Empfang der Nachrichten ($A_1, \ldots, A_l$).
- Die Kosten können direkt als Kommentare an die Tasks (partielle Kosten) bzw. Zustände (Gesamtkosten) geschrieben werden.
- Ist eine Unterscheidung zwischen Kosten für den Nachrichtenempfang und Kosten für die anschließende Verarbeitung erforderlich, gelingt dies mit MSCs durch eine Verfeinerung[3] der jeweiligen Instanz.

[1] In diesen Kosten würde man typischerweise die Kosten aufsummieren, die für die Generierung einer entsprechenden Antwort erforderlich sind.

[2] Derartige Fallunterscheidungen sind immer dann wichtig, wenn man versucht, effiziente Implementierungen zu erreichen. Bereits beim Entwurf der parallelen Packet Screen wurde darauf hingewiesen, daß an verschiedenen Stellen im Endsystem eine Selektion erfolgen kann und am effizientesten die Selektion unmittelbar nach dem Empfang ist. Neben der dort durchgeführten Selektion im Gerätetreiber fallen für den Empfang von Dateneinheiten beispielsweise an den folgenden Stellen Kosten an: Netzschnittstelle (“Ethernetkarte”), Gerätetreiber, höher liegende STREAM-Module (beispielsweise IP, UDP, TCP, Packet Screen Modul) und schließlich auf der Anwendungsebene und bei den Anwendungen. Der “Empfang” einer Dateneinheit ist somit ein mehrphasiger Prozeß mit zunehmendem Aufwand.

[3] Dies ist ohnehin ein Standardverfahren zur Erhöhung der Genauigkeit im Zuge einer Protokollspezifikation.

Abbildung A.12 zeigt eine solche Kostenannotation. Es wird der relative Aufwand (“preiswert”, “mittlerer Aufwand” und “hoher Aufwand”) als Annotation zu den einzelnen Prozeduraufrufen angegeben, die dem Senden der Nachricht vorausgehen bzw. dem Empfang der Nachricht folgen. Der Gesamtaufwand ist an den jeweiligen Folgezuständen angegeben (beim relativen Aufwand reicht es im Beispiel aus, hier die “Größenordnung” anzugeben, also in beiden Fällen ein hoher Aufwand). Die Aufwandsabschätzung ist im Beispiel folgendermaßen: Tabelleneinträge nach Schlüssel absuchen ist nicht aufwendig. Das Verschlüsseln und Entschlüsseln mit symmetrischen Schlüsseln ist mit mittlerem Aufwand behaftet. Das Anfertigen und Prüfen einer “public-key” Signatur (asymmetrische Kryptographie) ist extrem aufwendig [145, 233]. Der Gesamtaufwand ist somit in beiden Fällen sehr hoch.

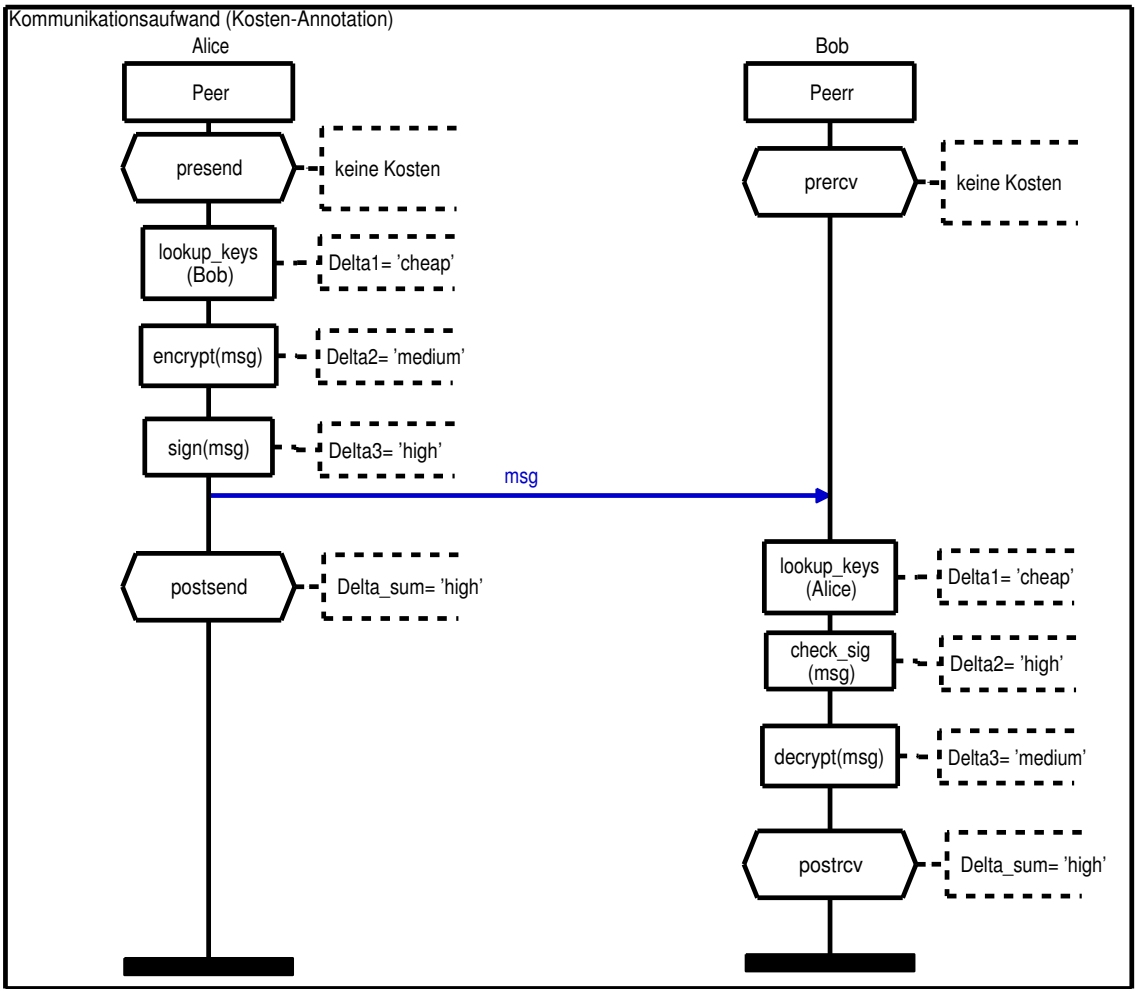


Abbildung A.12: MSC mit Kostenannotation

Eine Verfeinerung des Nachrichtenempfangs wird in Abbildung A.13 gegeben. Es werden drei Ebenen unterschieden: Die Netzschnittstelle, der Gerätetreiber im Betriebssystem und

die Anwendungsebene. Die Verfeinerung zeigt deutlich, daß bereits weit vor der Nachrichtenverarbeitung auf der Anwendungsebene bereits im Betriebssystem erheblicher Aufwand entsteht.

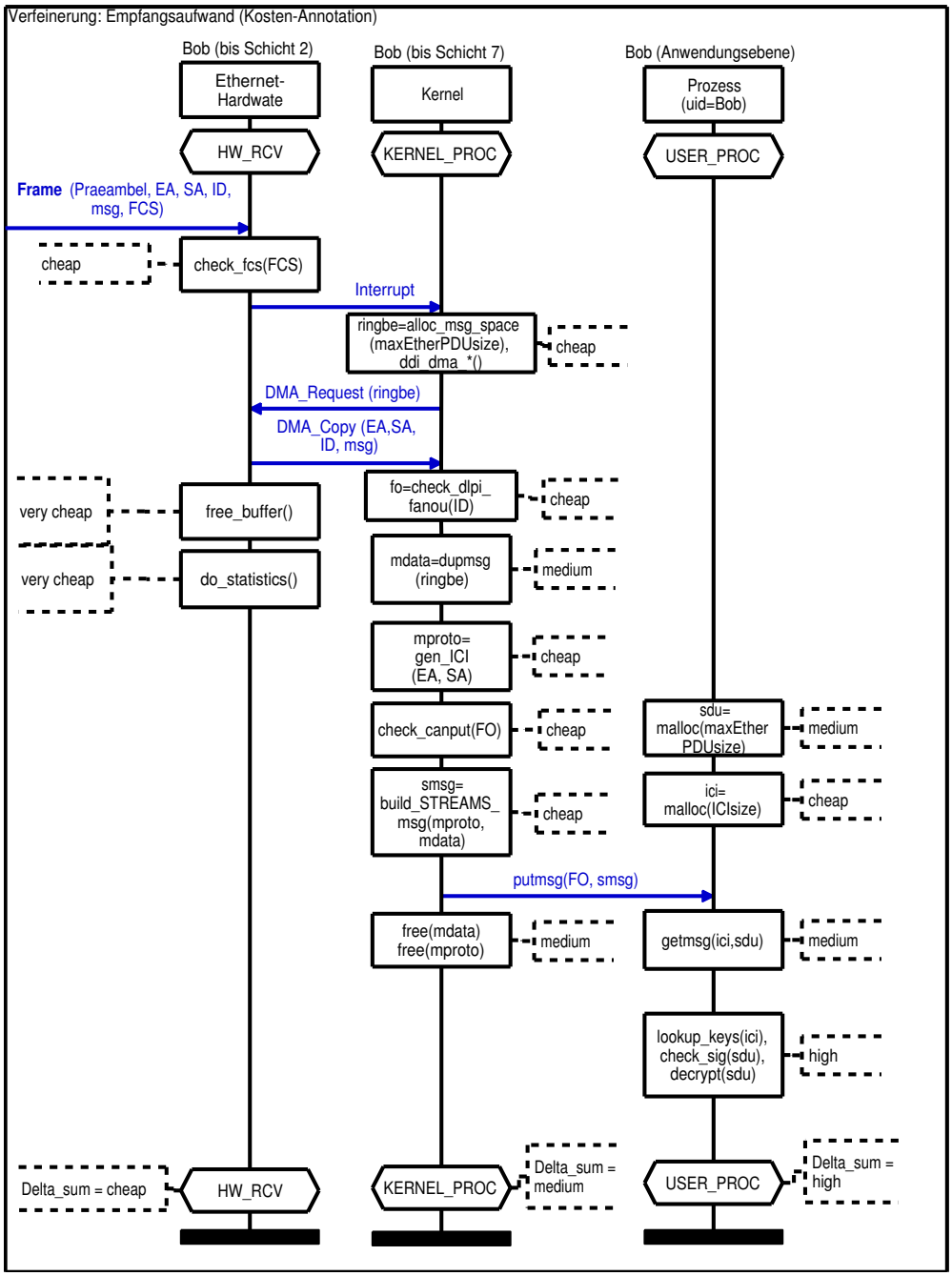


Abbildung A.13: Verfeinertes MSC mit Kostenannotation

Während die Operationen (Prüfsumme berechnen, Statistikzähler führen) seitens der Ethernet-Hardware (EHW) als (vernachlässigbar) billig anzusehen sind, ist der Aufwand im Kernel des Betriebssystems bereits deutlich höher. Die EHW meldet den vollständigen Empfang einer korrekten Nachricht durch eine Unterbrechung (interrupt). Das Betriebssystem führt dann den entsprechenden Code des Gerätetreibers aus. Typische Funktionen für Netzhardware sind in der Abbildung angedeutet: Ein freier Eintrag in einem Ringpuffer muß gesucht werden, diverse `ddi_dma_*()`-Funktionen müssen aufgerufen werden, um den “direct memory access” (DMA) vorzubereiten. Danach werden die wesentlichen Bestandteile der PDU in den Kernel kopiert, was wegen der DMA-Unterstützung relativ aufwandsarm ist.

Eine anschließende Analyse der ID (hier: Ethernettyp-Kennung) ergibt einen Hinweis darauf, ob ein SAP für Dateneinheiten dieses Typs vorhanden ist (hier angedeutet durch `check_dlpi_fanout()`). Ist mindestens ein SAP vorhanden, wird die Nachricht kopiert (`dupmsg()`). Dies ist eine vergleichsweise aufwendige Operation, da hierfür neuer Speicher angefordert (verwaltet) werden muß und die Nachricht physikalisch aus dem Ringpuffer kopiert wird. Danach folgen mehrere Operationen zur Erstellung einer DLPI-konformen STREAM-Nachricht (ICI-Teil mit Adreßinformationen generieren, SDU-Teil mit Zeiger auf Nutzdaten generieren), die schließlich mit einer Folge von `putmsg()`-Aufrufen die Kette der STREAM-Module “hochgereicht” wird. Dieser Aufwand wird dem empfangenden Benutzerprozeß als Kernelarbeitszeit angerechnet.

Auch auf der Anwendungsebene fällt bereits vor dem Empfang der SDU-Daten Arbeitsaufwand an. Die dynamische Speicherverwaltung stellt Speicher für die STREAM-Nachricht vom Kernel bereit. Es wird beim Ausführen des `getmsg()`-Aufrufs ein zweites Mal kopiert (Kernelspeicher in Benutzerspeicher), was wiederum vergleichsweise hohen Aufwand darstellt. Erst danach ist die Dateneinheit auf der Anwendungsebene empfangen! Dieser Aufwand wird dem Benutzerprozeß bereits als Benutzerarbeitszeit angerechnet.

Nun beginnt die anwendungsspezifische Verarbeitung der SDU-Daten der Ethernet-PDU, mit dem bereits in Abbildung A.12 angedeuteten hohen Aufwand.

Abschließend sei zur Kostenannotation noch angemerkt, daß hierbei zwischen verschiedenen Betriebsmitteltypen unterschieden werden kann (Speicher, CPU-Zeit, DMA-Kanal, Ethernet-HW). Die in den Beispielen durchgeführte Abschätzung bezog sich lediglich auf die CPU-Zeit. Durch Messungen an bereits vorhandenen Systemen könnten dann auch quantifizierbare Werte angegeben werden, die genaue Aussagen über die aufzuwendenden CPU-Zeiten ermöglichen. Die hier vorgeschlagene Integration von Kostenannotationen ist vergleichbar mit ähnlichen Annotationen, die in [84, 133, 171] vorgeschlagen werden. Die dort vorgestellten Annotationen sind jedoch für eine automatische Leistungsanalyse optimiert und erfordern daher die exakte Angabe von Verarbeitungszeiten. Mit der hier vorgestellten Erweiterung der MSCs um Annotationen gemäß [163] wird die Sicherheitsanalyse besser unterstützt, da primär die Ursachen für die enstehenden Kosten notiert werden.

A.2.4 Gleichzeitige Integration mehrerer Annotationen

Ein weiterer Vorteil bei Verwendung von MSCs ist, daß die verschiedenen Annotationen auch miteinander kombiniert werden können. Zur Verdeutlichung wird hier noch einmal auf Formel 8.9 aus Abschnitt 8.5.1 eingegangen, die im wesentlichen aussagt, daß es für die Authentifikation unerheblich ist, ob eine verschlüsselte Nachricht eintrifft oder eine Nachricht, über die mit demselben Schlüssel ein “keyed MAC” berechnet wurde. In jedem Fall kann der Empfänger prüfen, ob die Nachricht (ursprünglich[4]) von jemandem generiert wurde, der im Besitz des geheimen Schlüssels ist.

Der BAN-Logik entzieht sich jedoch die Bewertung des Aufwands der Alternativen vollständig. Durch eine gemeinsame Anwendung der BAN-Logik und der Aufwands-Annotationen läßt sich der Unterschied verdeutlichen. Abbildung A.14 zeigt den Aufwand und die Schlußfolgerung bei Empfang einer verschlüsselten Nachricht im Vergleich zum Empfang einer Nachricht mit angefügtem “keyed MAC”.

Die beiden Kommunikationspartner (Alice und Bob) teilen sich den Schlüssel K. Alice sendet eine hiermit verschlüsselte Nachricht (SDU) an Bob, die in eine Protokolldateneinheit eingebettet ist. Bob empfängt diese verschlüsselte Nachricht (sieht $\{sdu\}_K$) und würde laut “Message Meaning Axiom” der BAN-Logik sofort auf den Klartext (sdu) schließen und glauben, daß dieser von Alice generiert wurde ($Bob|\equiv Alice|\sim sdu$). Hierbei mißachtet die BAN-Logik den Aufwand für die Entschlüsselung von $\{sdu\}_K$ völlig.

Durch die zusätzliche Kostenannotation im Task ($sdu = descrypt(\{sdu\}_K)$) wird dieser Aufwand jedoch explizit dargestellt und sdu als direktes Ergebnis dieser Verarbeitung benannt. Der Protokollautomat von Bob erreicht den Folgezustand (“Nachricht ist entschlüsselt”). In realen Systemen kann erst jetzt der – in der Abbildung auch als Kommentar eingezeichnete – Schluß gezogen werden: $Bob|\equiv Alice|\sim sdu$.

Anschließend wird eine zweite PDU empfangen, die aus einem unverschlüsselten SDU-Teil besteht ($sdu2$) und einem mit einer Einweg-Streufunktion und dem gemeinsamen Geheimnis K berechneten “keyed MAC” über diese SDU ($H(K, sdu2)$). Die BAN-Logik-Annotationen (mit den Erweiterungen aus [99]) besagen in diesem Zustand folgendes: Bob sieht eine Nachricht, die nicht von ihm selbst generiert wurde und einen dazugehörenden Hash-Wert.[5] Auch in diesem Fall wäre unter der Zuhilfenahme der Regel “Message Interpretation I3” sofort darauf zu schließen, daß Alice irgendwann einmal den Hash-Wert generiert und die SDU irgendwann einmal als Nachricht versendet haben muß.

In realen Systemen ist jedoch die eingezeichnete Berechnung der Prüfsumme durch Bob für einen Vergleich mit der übertragenen Prüfsumme von Alice erst zu berechnen. Auch hierfür

[4] Es kann nicht überprüft werden, wann und von wem die Nachricht *gesendet* wurde. Daher wird hier der Begriff “generiert” verwendet. Soll darüber hinaus geprüft werden, ob die Nachricht “frisch” ist, müssen weitere Mechanismen eingesetzt werden, beispielsweise die Verwendung von Sequenznummern, wie in Abbildung 8.4.

[5] Die GNY-Logik wäre sogar in der Lage auszudrücken, daß der Hash-Wert und die Nachricht als solche erkennbar sind. Diese Präzision wird hier indes nicht benötigt.

kann man eine Kostenannotation angeben. Erst nach dem im Folgezustand angegebenen Vergleich kann schließlich die Integrität und Herkunft der Nachricht gefolgert werden.

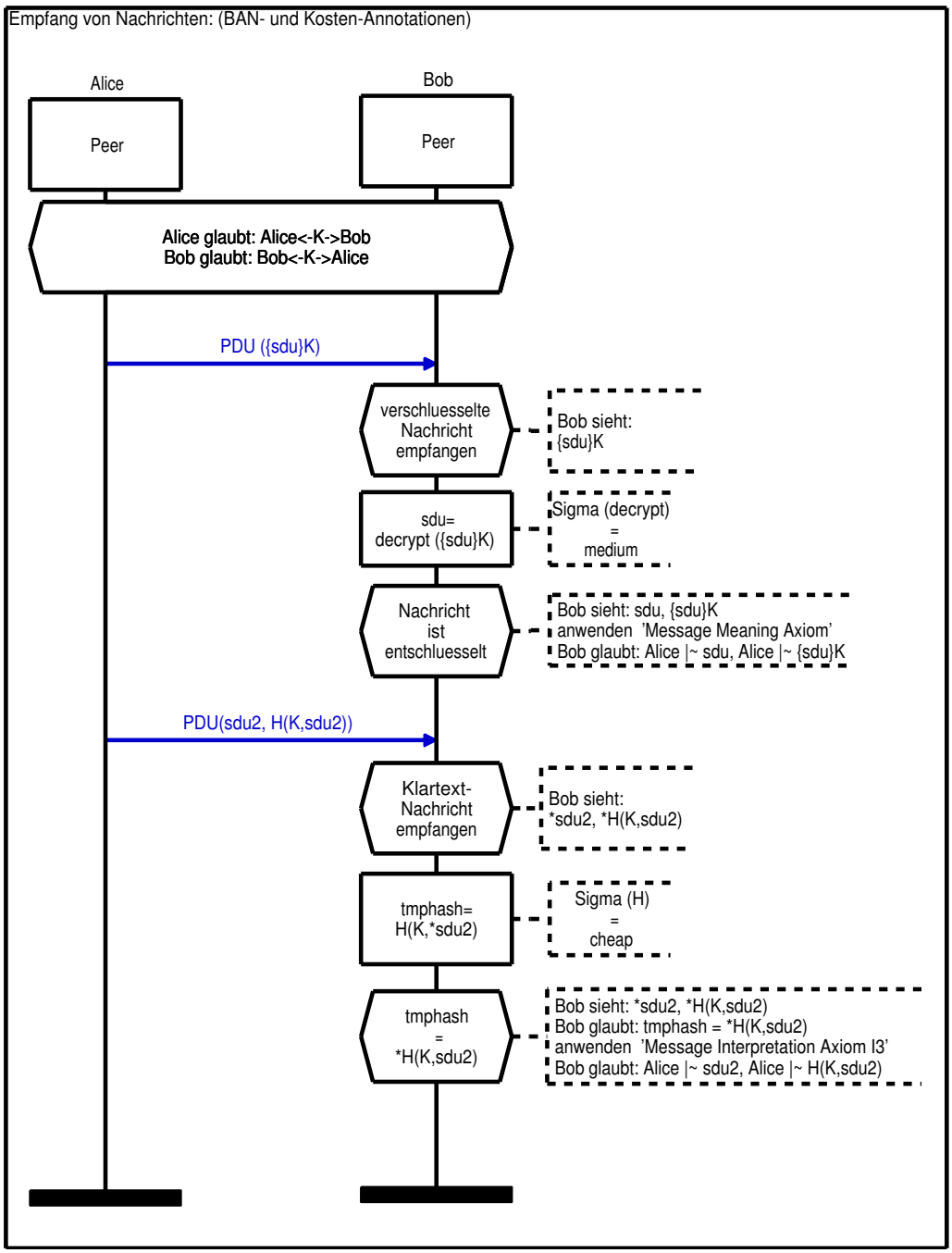


Abbildung A.14: MSC mit BAN- und Aufwands-Annotationen

Darüber hinaus verdeutlicht die Darstellung als MSC einen weiteren wichtigen Unterschied. Bei der Verschlüsselung hat Bob erst nach dem Entschlüsseln Zugriff auf den SDU-Teil der Nachricht. Beim Einsatz einer Hash-Funktion ist dieser Zugriff unmittelbar nach dem Empfang möglich, also insbesondere vor dem Berechnen der Integritätsprüfsumme. Dies ist ein immenser Vorteil: Bob kann bereits vor dem Ausführen aufwendiger Berechnungen entscheiden, ob die Nachricht für ihn relevant ist oder aufwandsarm verworfen werden kann. Im Falle der Verschlüsselung fällt der noch höhere Aufwand ($Sigma(decrypt) > Sigma(H)$) jedoch in jedem Fall an!

Diese sehr realitätsnahe Modellierung des *Aufwands für die Authentifikation* ist nur durch den kombinierten Einsatz beider Methoden möglich.

A.3 Zusammenfassung

In diesem Abschnitt wurde eine kurze Einführung in die formale Systemspezifikation mit SDL gegeben und gezeigt, wie MSCs und SDL gemeinsam für die Analyse und Synthese von verteilten Systemen eingesetzt werden können. Die in Kapitel 8 angesprochenen Defizite, durch den Darstellungsbruch bei einer zusätzlichen Sicherheitsanalyse mit Hilfe der BAN-Logik und der Kostenannotationen, konnten durch die in Abschnitt A.2 vorgestellten Methoden zur Integration dieser Techniken in MSCs beseitigt werden. Die gleichzeitige Verwendung mehrere Annotationen in einem MSC kann darüber hinaus zu weiteren Einsichten in das Protokoll führen, die durch den alleinigen Einsatz einer Technik nicht erkennbar wären.

Anhang B

Protokollspezifikation mit der "Specification and Description Language"

In diesem Anhang wird die erste Version des Gruppenmanagement-Protokolls (GMPs) in der bekannten graphischen SDL-Repräsentationen (SDL-GR) dargestellt. Wie bereits in Kapitel 7 diskutiert wurde, ist zusätzlich die Spezifikation eines "Broadcast"-Mediums erforderlich. Diese Spezifikation befindet sich in Anhang B.1. Anschließend folgt die eigentliche Protokollspezifikation in Anhang B.2.

B.1 Hilfsspezifikation: "Broadcast"-Medium

Ein wesentlicher Mechanismus des leichtgewichtigen GMPs ist die Verwendung von "Broadcast"- bzw. "Multicast"-Kommunikation. Anstelle allen $n-1$ anderen Gruppenmitgliedern jeweils eine individuelle Nachricht zu senden, sendet ein (potentielles[1]) Gruppenmitglied die Dateneinheiten an eine allen Gruppenmitgliedern bekannte, Medium-abhängige Gruppenadresse. Abhängig von der Wahl dieser Adresse unterstützen verschiedene physikalische Übertragungsmedien (Ethernets, TokenRing, FDDI, usw.) dann die aufwandsarme Übertragung dieser Nachrichten an alle oder auch an nur einige angeschlossene Endgeräte. Damit ist der Aufwand für die Adressierung der anderen (potentiellen) Gruppenmitglieder konstant. Bei Verwendung von Übertragungsmedien, die keine direkte Unterstützung für eine Gruppenkommunikation bieten (beispielsweise X.25, FrameRelay und ATM) wäre der Aufwand proportional zur Anzahl der Gruppenmitglieder.

Für die Spezifikation des GMPs ist es nicht erforderlich, genau festzulegen, welche "Multicast"- oder "Broadcast"-Adressen zu verwenden sind. Dies kann ohnehin erst im Zuge einer

[1] Bewerber sind potentielle Gruppenmitglieder.

Implementation festgelegt werden, da dann eine Entscheidung für ein bestimmtes Medium getroffen werden muß.[2] Es ist an dieser Stelle allerdings erforderlich, die wesentliche Eigenschaft eines solchen Netzes zu modellieren: Eine Dateneinheit, die ein angeschlossener Rechner sendet, wird an alle anderen angeschlossenen Netzkomponenten übertragen. Genau diese Eigenschaft wird mit der folgenden Spezifikation des Prozeß-Typs “BCASTCHAN” modelliert.

In Abbildung B.1 sind die erforderlichen Deklarationen dargestellt. Da SDL eine a priori Kenntnis über die zu verarbeitenden Signale benötigt, ist hier bereits der Nachrichtentyp “MSG” definiert, auf den alle HB-PDUs abgebildet werden können. Die Nachrichten “NEWPR” und “DELPR” werden benötigt, um den physikalischen Anschluß (beispielsweise Kabel verbindet Ethernet-Karte mit Switch) eines Rechners an das Netz zu simulieren. Der Prozeß “BCASTCHAN” spezifiziert einen SDL-Automaten, der die drei Nachrichten

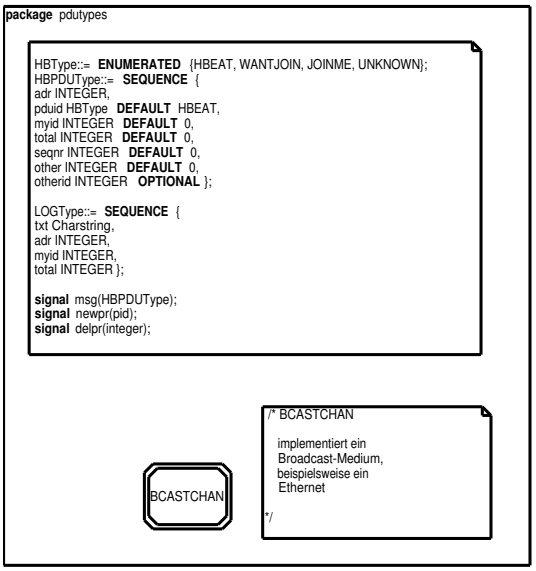


Abbildung B.1: Deklarationen für ein“Broadcast”-Medium

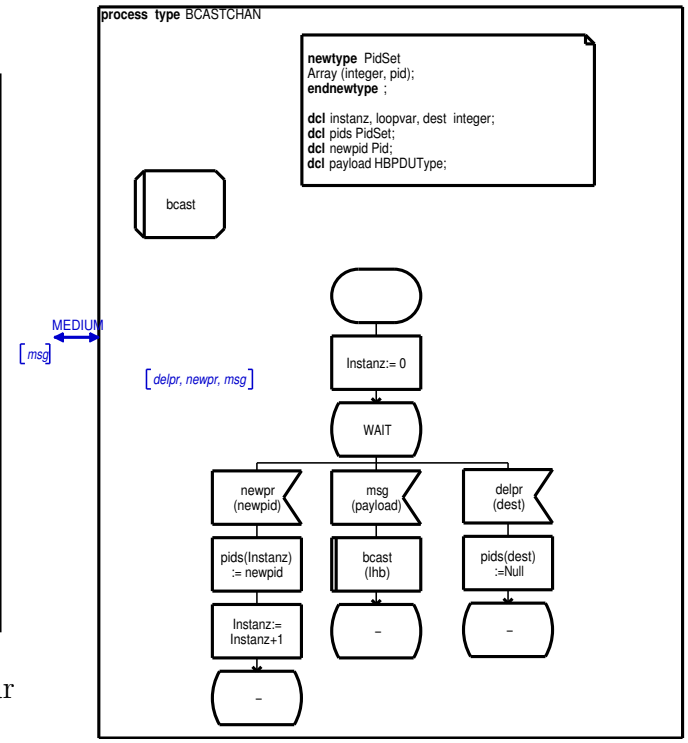


Abbildung B.2: Prozeß-Template: “Broadcast”-Medium

[2] Ganz korrekt ausgedrückt: Im Zuge der Implementation muß der Zugriff auf eine Gruppenadresse ermöglicht werden, so daß beim Senden von Dateneinheiten an diese Adresse das Übertragungsmedium die Weiterleitung an alle (potentiellen) Gruppenteilnehmer unterstützt. Verwendet man hierfür abstrakte Schnittstellen für einen Schicht-2-Dienst, wie beispielsweise das “Datalink Provider Interface”, [228], braucht der Benutzer (hier das GMP) sich nicht mehr um die Wahl des Netzes zu kümmern, sondern kann mit entsprechenden Dienstprimitiven ein Netz auswählen, das die Gruppenkommunikation unterstützt.

"NEWPR,DELPR" und "MSG" verarbeiten kann (vgl. Abbildung B.2). Dazu ist lediglich ein Zustand "WAIT" erforderlich, in dem eintreffende Nachrichten nacheinander abgearbeitet werden. In der lokalen Datenstruktur "PIDSET" merkt sich dieser Prozeß, ob ihm ein neuer Teilnehmer über eine "NEWPR"-Nachricht genannt wurde. Trifft eine Nachricht vom Typ "MSG" ein, wird sie an alle bisher bekannten Teilnehmer kopiert. Dies erfolgt mit dem Makro "BCAST" (vgl. Abbildung B.3).

B.2 Spezifikation eines Gruppenmanagement-Protokolls

Das GMP wird von einer oder mehreren kooperierenden Protokollinstanz(en) ausgeführt, um Konsens über eine gemeinsame Gruppe herbeizuführen. Jede einzelne Instanz kommuniziert hierzu mit einem lokalen Benutzer, der die Aufnahme in die Gruppe bzw. das Verlassen der Gruppe anfordert. Die Protokollinstanz muß mit diesem Benutzer über entsprechende Dienstelemente kommunizieren. Diese Dienstelemente wurden in Abschnitt 7.2.1 vorgestellt und sind in Abbildung B.4 formal als SDL-Paket spezifiziert.

B.2.1 Systemstruktur

SDL erfordert die Modellierung einer Systemstruktur, in der die Kommunikationswege zwischen internen Komponenten (Blöcken) einerseits und mit der Umgebung andererseits dargestellt werden. Die Blöcke werden dann zunehmend bis auf die Prozeßebene verfeinert. Abbildung B.5 zeigt diese Systemstruktur für das Gesamtsystem, bestehend aus einer oder mehreren Instanzen im Block "HBSERVER" und dem "Broadcast"-Medium, das im Block "NETZ" spezifiziert wird. Die Signallisten "S2U" und "U2S" fassen die Dienstelemente für die Kommunikation mit dem Benutzer zusammen. Der Benutzer wird als Teil der Umgebung angesehen und ist daher nicht Teil der Systemspezifikation. Der zusätzliche Kanal "IF_NETKONFIG" ermöglicht die Kommunikation mit dem Block "NETZ". Hierüber kann das Signal "DELPR" gesendet werden, um einzelne Instanzen vom Netz abzumelden. Damit kann der Ausfall einer Netzschnittstelle auf einem Rechner simuliert werden.

B.2.2 Blockstruktur

Die Verfeinerung des Blocks "HBSERVER" ist in Abbildung B.6 dargestellt. Die "Signalroutes" ("LOG,TRANCEIVER,KONFIG" und "CTRL") reichen die Signale der "Channels" ("IF_LOG, IF_NETZ" und "IF_KONFIG") von der Systemebene an die verschiedenen Prozesse durch. Es gibt zwei verschiedene Prozesse. Der Prozeß "ADMIN" ist für die Initialisierung der Simulation erforderlich. Nach dem Start der Simulation sendet der Benutzer diesem Prozeß mit dem Signal "ANZ" die Anzahl der zu startenden Protokoll-Instanzen. Der "ADMIN"-Prozeß startet darauf eine entsprechende Anzahl von "HBSERVER"-Prozessen und meldet diese am Netz an (Signal des Typs "NEWPR" über Route "KONFIG").

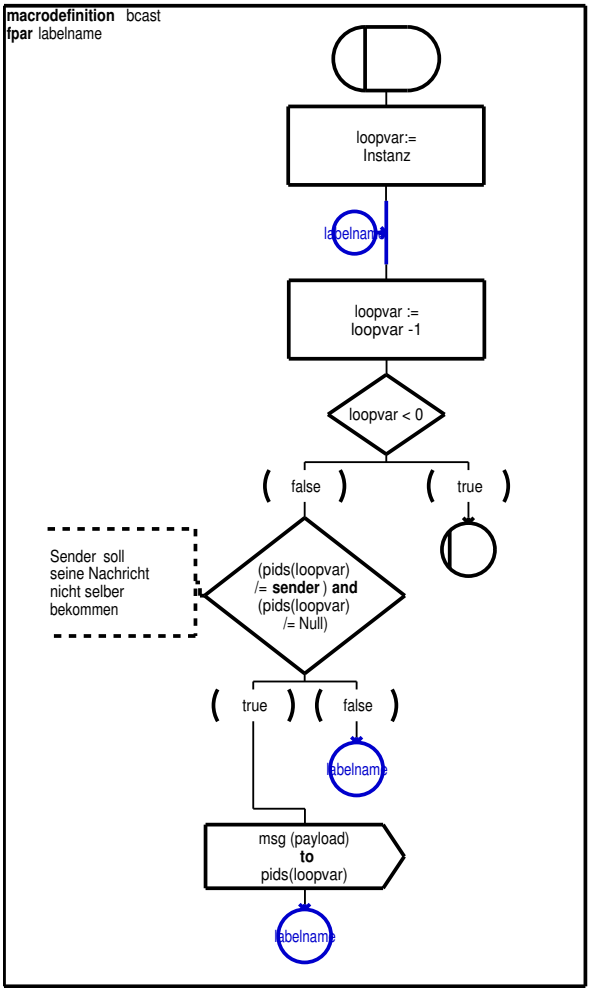


Abbildung B.3: Makro: “Broadcast” der Nachricht

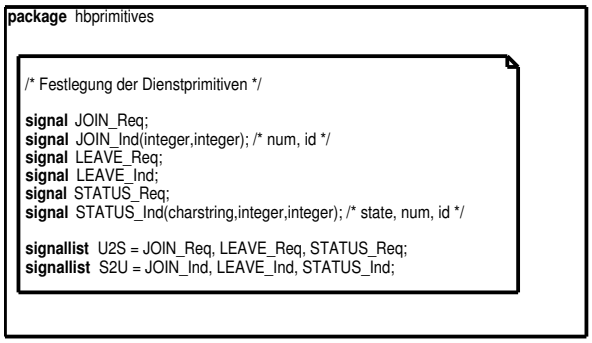


Abbildung B.4: Spezifikation der Dienstelemente

In Abbildung B.7 ist die Verfeinerung des Blocks “Netz” dargestellt. Es gibt lediglich einen Prozeß mit dem Namen “Ethernet” vom Typ “BCASTCHAN”. Hier wird somit unter Rückgriff auf die Hilfsdefinitionen aus Abschnitt B.1 das “Broadcast”-Medium spezifiziert. Da es sich um ein Vorkommnis einer Prozeßschablone handelt, mußten die Signalroutes zusätzlich an die in Abschnitt B.1 spezifizierten “Gates” des Prozesses gebunden werden (hier “MEDIUM”). Die Routes “Admin” und “AdminNet” sind lediglich für die Steuerung der Simulation erforderlich. Die “Broadcast”-Kommunikation benutzt lediglich die Route “Medium10Base2”.

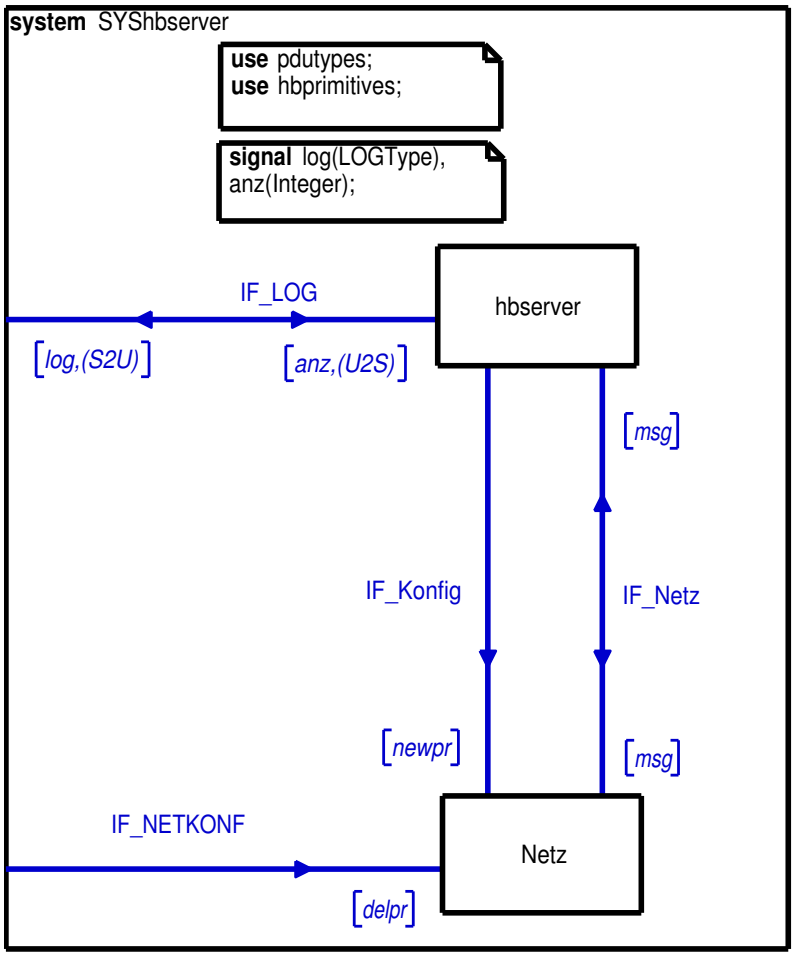


Abbildung B.5: Systemspezifikation

B.2.3 Prozeßstruktur (Protokollinstanz)

Eine Protokollinstanz des GMPs wird im SDL-Modell als Prozeß spezifiziert. SDL basiert auf erweiterten endlichen Automaten. Eine wesentliche Erweiterung gegenüber den endlichen Automaten ist die Möglichkeit, auf lokalen Variablen zu arbeiten. Im wesentlichen ermöglicht dies die deutliche Reduzierung von Zuständen, da innerhalb eines Zustandes weitere Zustandsunterscheidungen auf Grundlage der Variablen möglich sind. Bei der Modellierung hat man somit die Freiheit, sich auf die wesentlichen konzeptionellen Zustände des Protokolls zu beschränken.

In Abbildung B.8 sind die erforderlichen Datenstrukturen und Variablen definiert. Eine wichtige Datenstruktur, in der die Sicht der Instanz auf die Gruppenzusammensetzung (die Gruppenliste) gespeichert wird, ist vom Datentyp "DATASEQ1". Dieser Datentyp ermöglicht die Indizierung (wichtig für die Herstellung einer Reihenfolge für die Gruppenliste

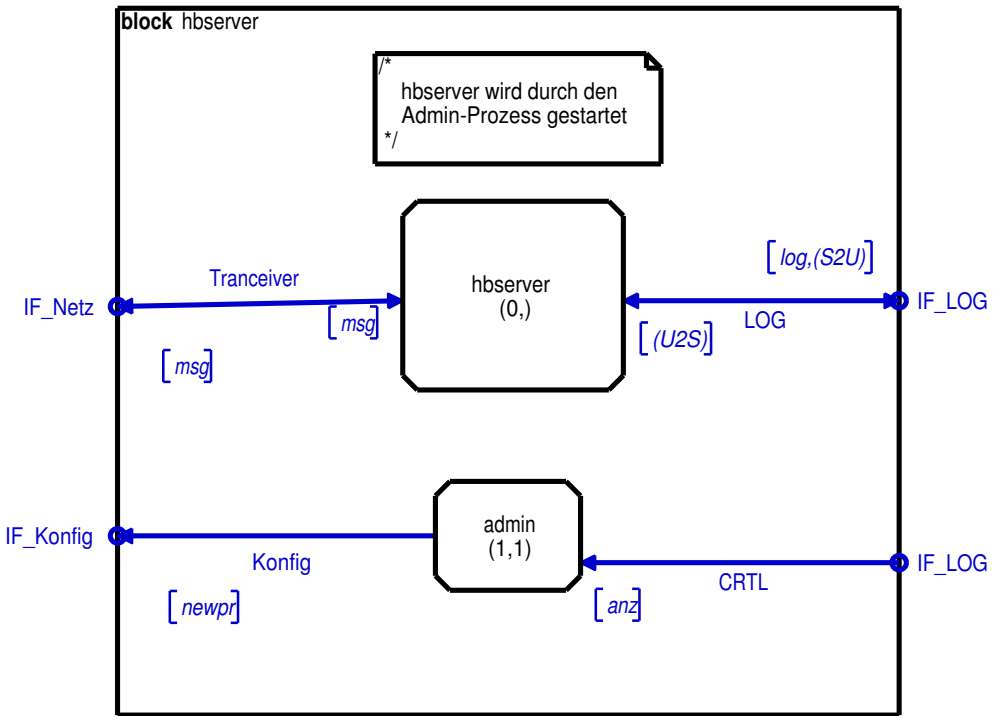


Abbildung B.6: Block: HBServer

nach Definition 7.2) von Einträgen des Typs “DBENTRY”. “DBENTRY” ermöglicht die Speicherung der Instanznamen und des Zeitpunktes, an dem zuletzt eine HB-Nachricht von dieser Instanz empfangen wurde.

Ein paar einzelne Variablen sind ebenfalls erwähnenswert. In “MYID” wird der lokale Name (repräsentiert durch eine ganze Zahl) gespeichert. In “TOTAL” wird bei der Aufnahme in eine Gruppe die Gesamtanzahl aller Gruppenmitglieder gespeichert. Die anderen Variablen dienen mehr oder weniger als Hilfsspeicher oder zur weiteren Differenzierung innerhalb eines Zustandes.

- **Start Zustand, Zustand “Calc-Conf”**

SDL-Prozesse sind wiederum durch einzelne Zustände, Funktionen und Makros strukturierbar. In Abbildung B.9 sind zwei Zustände einer Instanz dargestellt. Der Startzustand wird durch das Oval (ohne Beschriftung) dargestellt. Die einzelnen Anweisungen werden der Reihe nach (von oben nach unten) ausgeführt. Eine neu gestartete Instanz beginnt demnach mit einer Initialisierung (als Kommentar in Task “INIT”). Die darauf folgenden Makroaufrufe (Makros fassen mehrere Aktionen unter einem gemeinsamen Namen zusammen), übernehmen diverse lokale Initialisierungsaufgaben. Es erfolgt das Vorbelegen lokaler Variablen (Makro “INITLOCAL”), das Vorbelegen einer Nachricht (Makro “INITPDU”) und

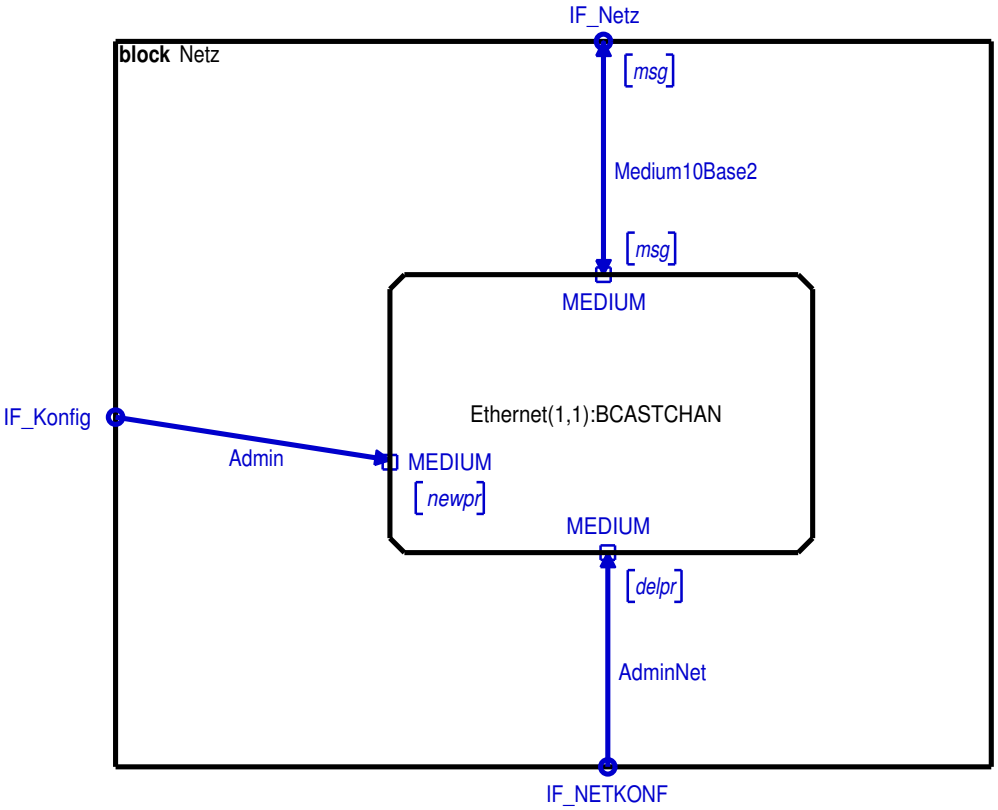


Abbildung B.7: Block: Netz

das Eintragen der lokalen Instanz in die eigene Gruppenliste (Makro "ADDSELF"). Die Instanz geht danach in den Zustand "PS" über. Dies ist ein Zwischenzustand, der erforderlich ist, da SDL im Startzustand keine Nachrichten empfangen kann. Er hat keinerlei semantische Bedeutung und kann somit als "Erweiterung" des Startzustandes angesehen werden.

Der ebenfalls in der Abbildung aufgenommene Zustand "CC" (Calc-Conf) wird durchlaufen, wenn in einem anderen Zustand erkannt wird, daß sich die Gruppenzusammensetzung geändert hat. In "CC" werden dazu zwei Prozeduren benutzt: Die Prozedur "CALCCONF" aktualisiert die Gruppenliste, die Prozedur "RECONF" benachrichtigt den Benutzer über die Änderung mit einer "STATUS_IND"-Nachricht. Es wird anschließend das Makro "INITPDU" aufgerufen, das im wesentlichen die Sequenznummer in der Nachrichtenstruktur erhöht. Daraufhin wird mit einer HB-Nachricht die neue Konfiguration auch den Gruppenmitgliedern mitgeteilt. Der Folgezustand ist "WB" (Wait-Beat).

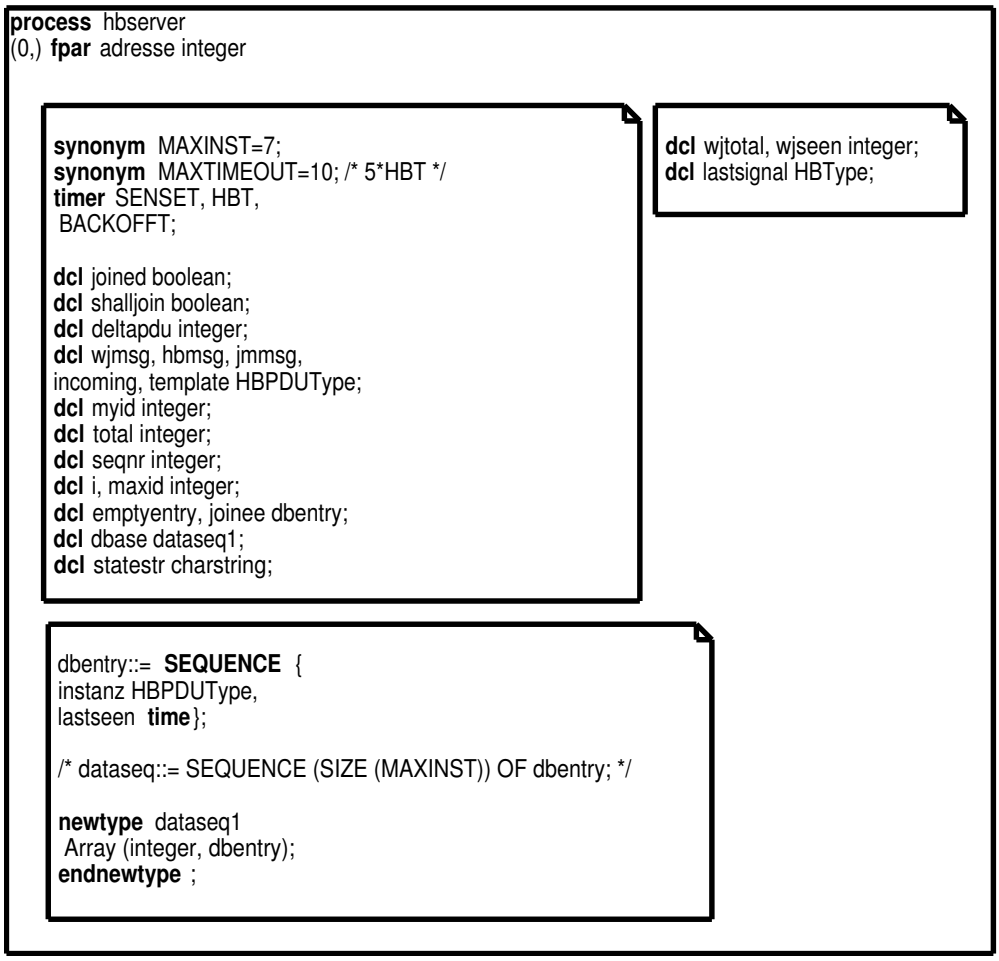


Abbildung B.8: Prozeß: GMP-Instanz

- **Zustand “Past-Start”**

Der Zustand “PS” (Past-Start) ist, wie bereits oben angedeutet, erforderlich, da SDL im Startzustand keine Möglichkeit für einen Nachrichtenempfang vorsieht. Nach dem Start einer Instanz ist jedoch als erstes auf eine Nachricht des Benutzers zu warten. Dieser könnte alle “Request”-Nachrichten senden. Insbesondere aber die Nachricht “JOIN_REQ” ist erforderlich, damit sich die Protokollinstanz um die Aufnahme in die Gruppe bemüht.

Ist die boolesche Variable “SHALLJOIN” wahr (shalljoin = true), kann direkt mit der Protokollverarbeitung fortgefahren werden. Dies ist erforderlich, falls die Instanz über den Zustand “BACKOFF” wieder in den Startzustand gelangt und erneut versuchen soll, in die Gruppe zu gelangen. Die Variable “SHALLJOIN” legt demnach fest, ob es noch eine ausstehende “JOIN_REQ”-Aufforderung des Benutzers gibt.

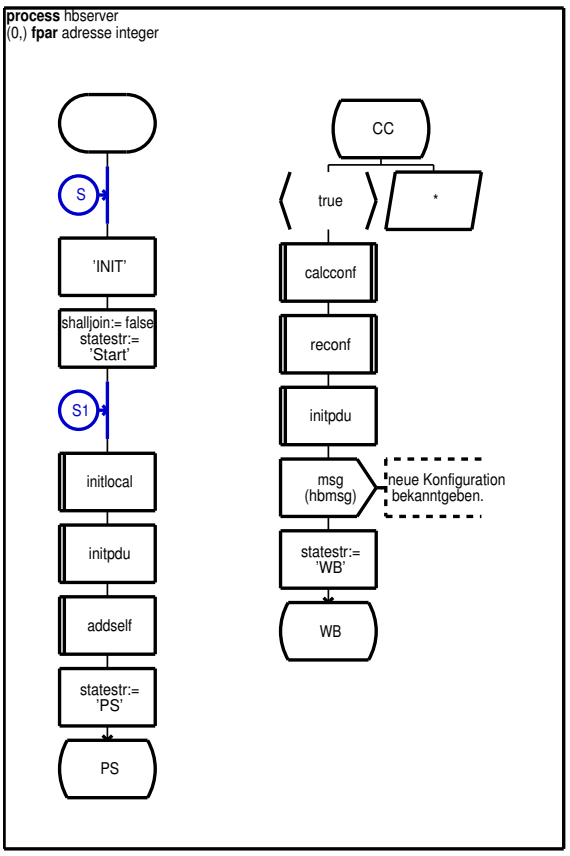


Abbildung B.9: Zustände: Start und Calc-Conf

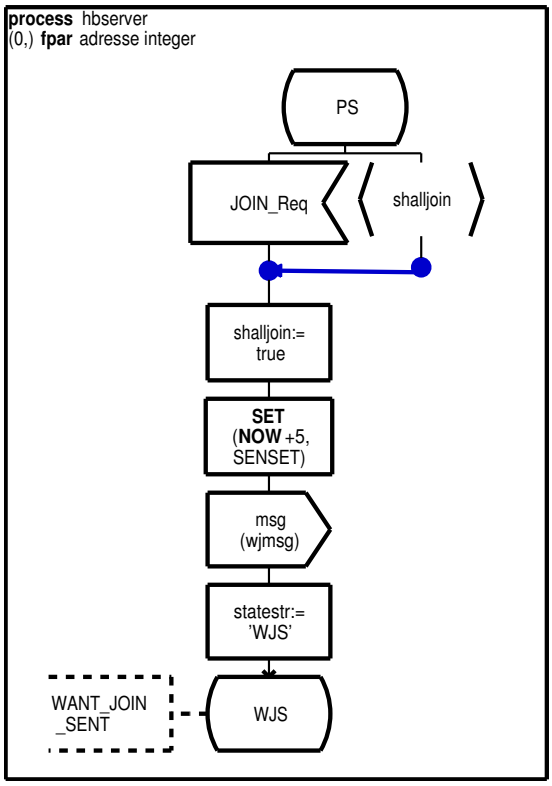


Abbildung B.10: Zustände: PS

Im Zustand "PS" wird erstmalig auch ein Zeitgeber (Timer) verwendet. Der Timer "SENSET" wird gestellt, um die folgende Phase ggf. zu beenden. Dieser Timer garantiert, daß der Protokollautomat im Folgezustand "WJS" (Want-Join-Sent) trotz potentiell ausbleibender Nachrichten anderer Instanzen Fortschritte macht. Dies ist beispielsweise der Fall, falls es nur diese eine Instanz gibt, und somit niemand auf die in diesem Zustand gesendete "WANT-JOIN"-Nachricht reagiert (vgl. Abbildung B.10).

- **Zustand "Want-Join-Sent"**

Nach Absenden der Nachricht "WANT-JOIN" können im Zustand "WJS" (Want-Join-Sent) zwei Ereignisse eintreten. Es könnte der zuvor in "PS" gestellte Timer ablaufen oder es könnte eine Nachricht einer anderen Instanz eintreffen.

Falls der Timer "SENSET" abläuft, führt die Instanz die in Abschnitt 7.2.4.3 diskutierten Aktionen durch, um unmittelbar als erste und einzige Instanz die neue Gruppe zu gründen. Hierzu wird die neue Gruppenliste berechnet (Prozedur "CALCONF"), eine (pro forma) Nachricht "JOIN-ME" sowie die erste HB-Nachricht versendet. Anschließend kann der

“JOIN_Req” des Benutzers mit einem “JOIN_Ind” beantwortet werden. Der Folgezustand nach Ablauf des Timers “SENSET” im Zustand “WJS” ist der Zustand “WB” (Wait-Beat).

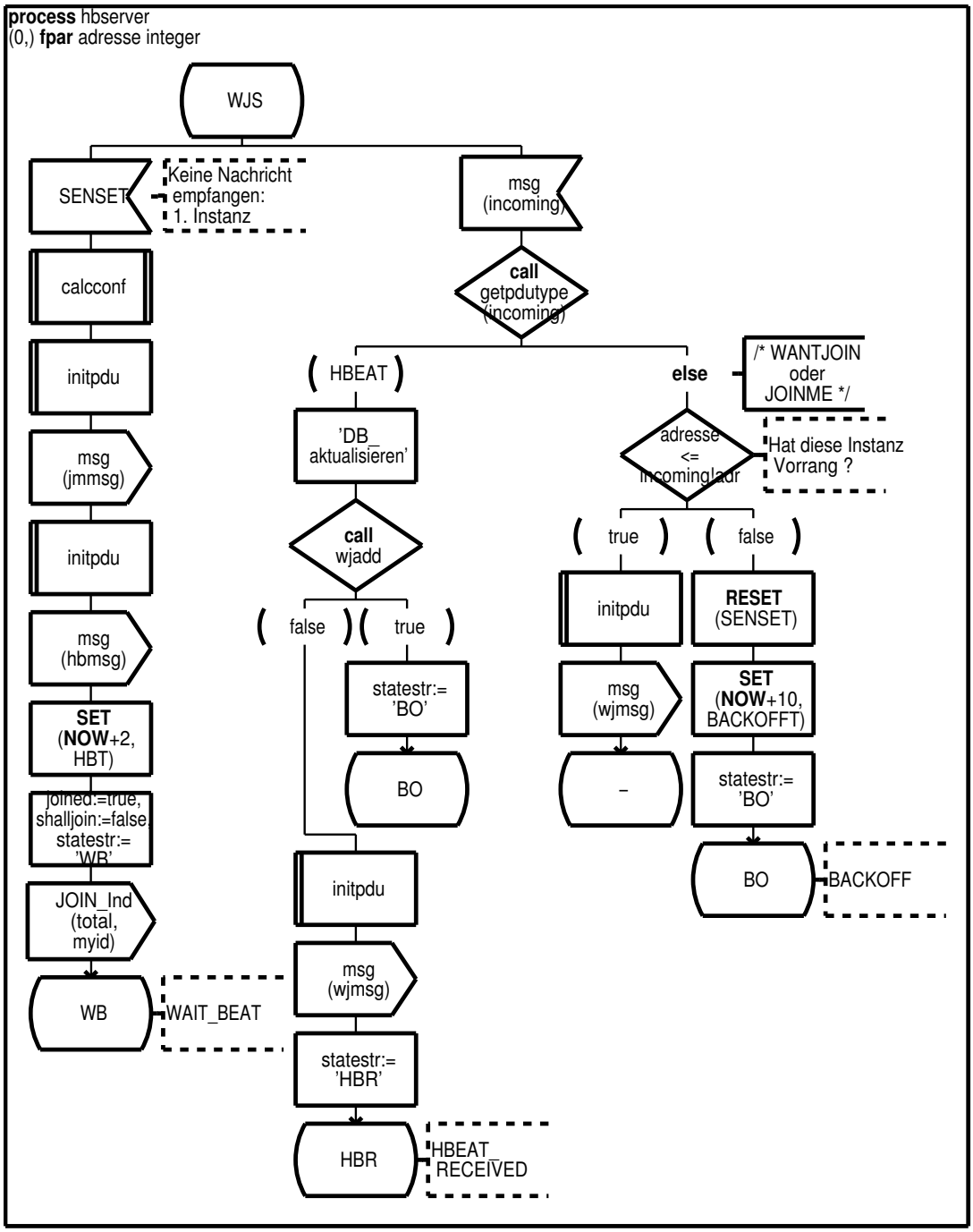


Abbildung B.11: Zustand: Want-Join-Sent

Wird jedoch eine Nachricht von einer anderen Instanz empfangen, ist zunächst der Typ der Nachricht zu prüfen. Dazu wird die Prozedur "GETPDUTYPE" aufgerufen, die den PDU-Typ zurückliefert. Handelt es sich um eine HB-Nachricht, wird die lokale Gruppenzusammensetzung aktualisiert (Prozedur "WJADD"). Wird hierbei festgestellt, daß kein Konsens unter den bereits erkannten Gruppenmitgliedern herrscht, geht die Instanz in den Zustand "BO" (Backoff) über. Andernfalls wird die eigene "Want-Join"-Nachricht erneut gesendet und in den Zustand "HBR" ("Heart-Beat-Received") gewechselt. Dieser Folgezustand zeigt an, daß die Instanz nicht allein ist, sondern es bereits mindestens ein Mitglied in der Gruppe gibt.

Trifft eine andere Nachricht ein (möglich wäre ein "JoinMe" oder ein "Want-Join" einer anderen Instanz), wird geprüft, ob die lokale Instanz Priorität vor dieser anderen Instanz hat (vgl. Definition 7.1). Falls die andere Instanz Vorrang hat, wechselt die lokale Instanz in den Zustand "BO", andernfalls wird die eigene "Want-Join"-Nachricht wiederholt und im Zustand "WJS" verweilt (vgl. Abbildung B.11).

- **Zustand "Heart-Beat-Received"**

Der Zustand "HBR" (Heart-Beat-Received) wird aus dem Zustand "WJS" erreicht, falls von mindestens einer anderen Instanz eine HB-Nachricht empfangen wurde. Die Instanz muß sich einen Überblick über die aktuelle Gruppenzusammensetzung verschaffen, da sie nur aufgrund dieser Gruppenzusammensetzung in der Lage ist, ihre eigene Position in der Gruppenliste zu berechnen. Die vorausblickende Assoziation der eigenen Position innerhalb der Gruppe — nach einem potentiellen Beitritt in diese Gruppe — ist die wesentliche Voraussetzung für eine konsistente Gruppenliste. Hierzu sammelt die Instanz die eintreffenden HB-Nachrichten und überprüft, ob sie von allen anderen Instanzen[3] eine HB-Nachricht erhalten hat. Falls dies der Fall ist, kann die Instanz mit einer "Join-Me"-Nachricht ihren Beitritt bekanntgeben, die erste eigene HB-Nachricht mit der lokalen Gruppenliste versenden und in den Zustand "WB" wechseln. Stellt die Instanz während der Auswertung der HB-Nachrichten (Prozedur "WJADD") fest, daß kein Konsens herrscht, geht sie wiederum in den Zustand "BO" über (vgl. Abbildungen B.12 bis B.14).

Ähnlich wie bereits im Zustand "WJS" können auch in diesem Zustand wiederum "JoinMe"- oder "Want-Join"-Nachrichten anderer Instanzen eintreffen. In diesem Fall muß wiederum entschieden werden, wer höhere Priorität hat. Gegebenenfalls muß die lokale Instanz in den Zustand "BO" wechseln (vgl. Abbildung B.12).

Es kann auch passieren, daß der Timer "SENSET" abläuft. Aus irgendeinem Grund hat die lokale Instanz es in diesem Fall nicht geschafft, der Gruppe beizutreten. Beispielsweise könnten alle anderen Instanzen schlagartig ausfallen[4], und die lokale Instanz hat noch nicht von allen eine HB-Nachricht empfangen. Die lokale Instanz wäre dann in einem "Deadlock", da keine weiteren HB-Nachrichten eintreffen können, was für einen Wechsel in den

[3] Im PDU-Format ist dafür der Eintrag "hbtotal" vorgesehen (vgl. Abbildung 7.4).

[4] Der Ausfall eines Stromkreises könnte dies bewirken.

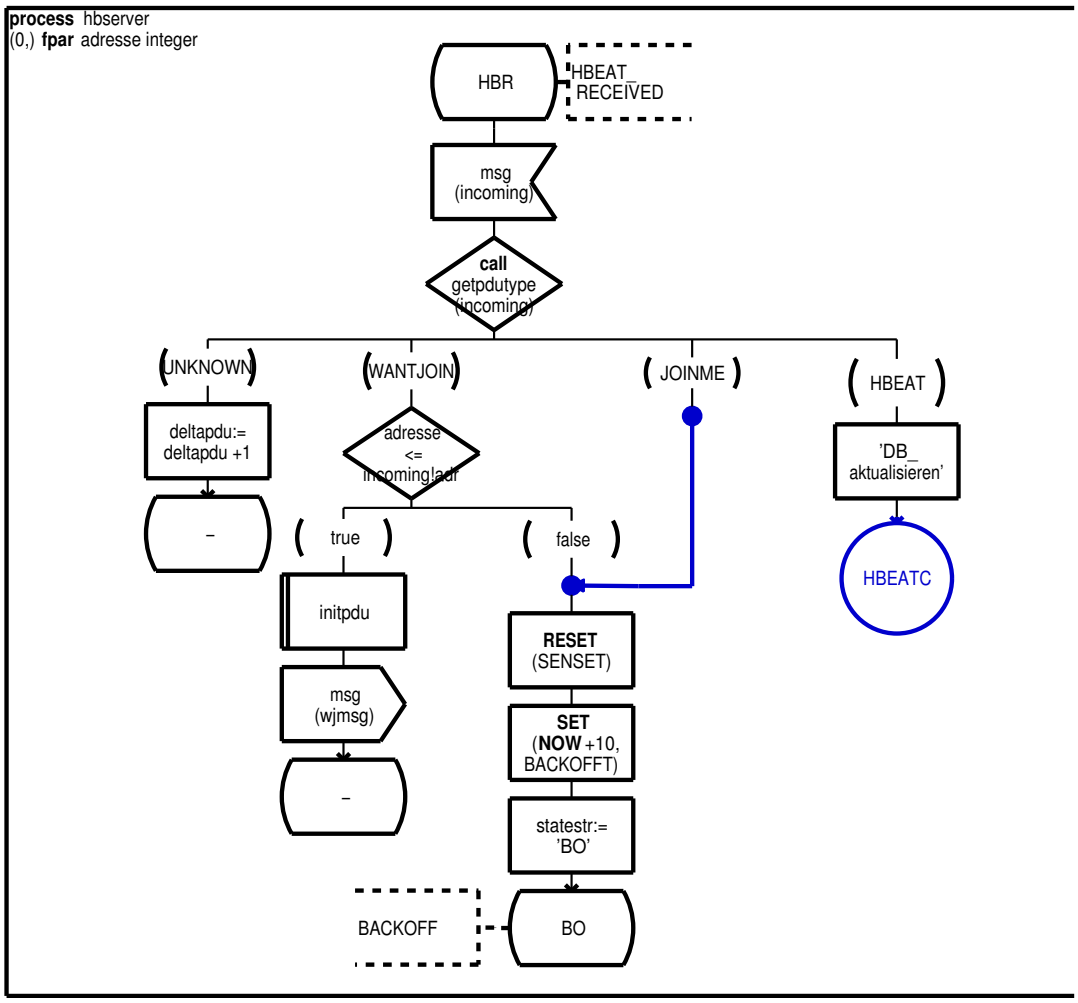


Abbildung B.12: Zustand: HB-Rec 1/3

Folgezustand “WB” jedoch unbedingt erforderlich wäre. Der ablaufende “SENSET”-Timer bringt die lokale Instanz in diesem Fall in den Zustand “BO”. Auch in diesem Fall sichert der Timer “SENSET” den Fortschritt (“Liveness”) des Protokolls (vgl. Abbildung B.15).

- **Zustand “Wait-Beat”**

Auch im Zustand “WB” (Wait-Beat) muß die Instanz auf alle eintreffenden Nachrichten reagieren können. Es wird die übliche Fallunterscheidung in Abhängigkeit vom PDU-Typ durchgeführt. Eintreffende HB-Nachrichten dienen dazu, die Zeitstempel in der lokalen Gruppenliste zu aktualisieren. Diese Zeitstempel werden periodisch bei Ablauf des lokalen “HBT”-Timers (Heart-Beat-Timer) überprüft. Dazu wird die Prozedur “CHK_TIMEOUT”

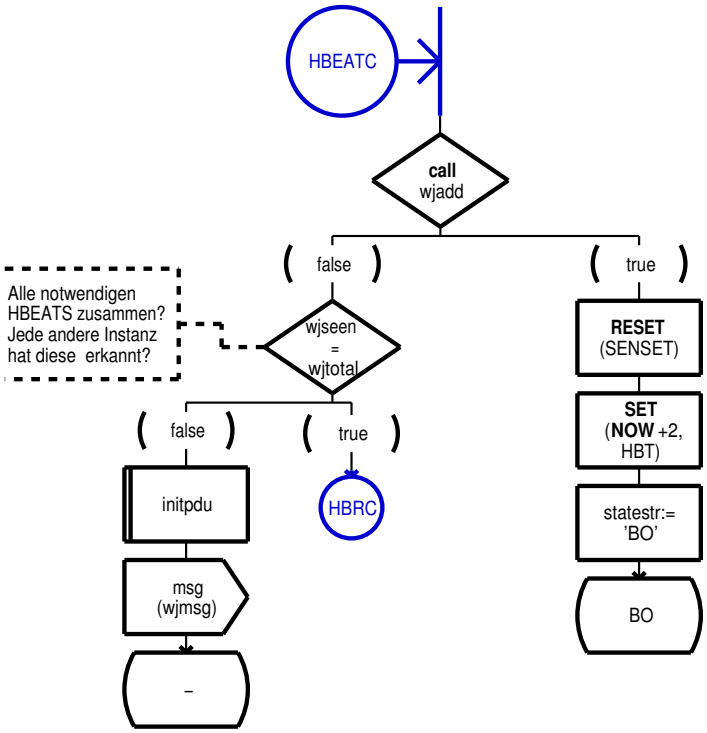


Abbildung B.13: Zustand: HB-Rec 2/3

aufgerufen, die für jeden Eintrag prüft, ob dieser jünger als die aktuelle Zeit abzüglich $HB_{intervall} * (HB_{maxlost} + 1)[sec]$ ist. Falls sich eine Instanz zu lange nicht mehr gemeldet hat, muß von einem Ausfall ausgegangen werden und die Gruppenliste wird neu berechnet (Wechsel in Zustand "CC").

Solange weiterhin Konsens herrscht, wird bei Ablauf des Timers "HBT" eine neue HB-Nachricht versendet. Falls eine "JOINME"-Nachricht eintrifft, wird die Gruppenliste um das neue Mitglied erweitert (Prozedur "INSTDB") und durch Wechseln in den Zustand "CC" die erforderlichen lokalen Anpassungen durchgeführt.

Der Zustand "WB" ist der Normalzustand des Protokolls für alle Instanzen, die fehlerfrei in der Gruppe arbeiten. Dies ist sozusagen ein akzeptierender Zustand. Das Protokoll macht Fortschritt, wenn dieser Zustand immer wieder erreicht wird. Der Zustand "WB" ist in Abbildung B.15 dargestellt.

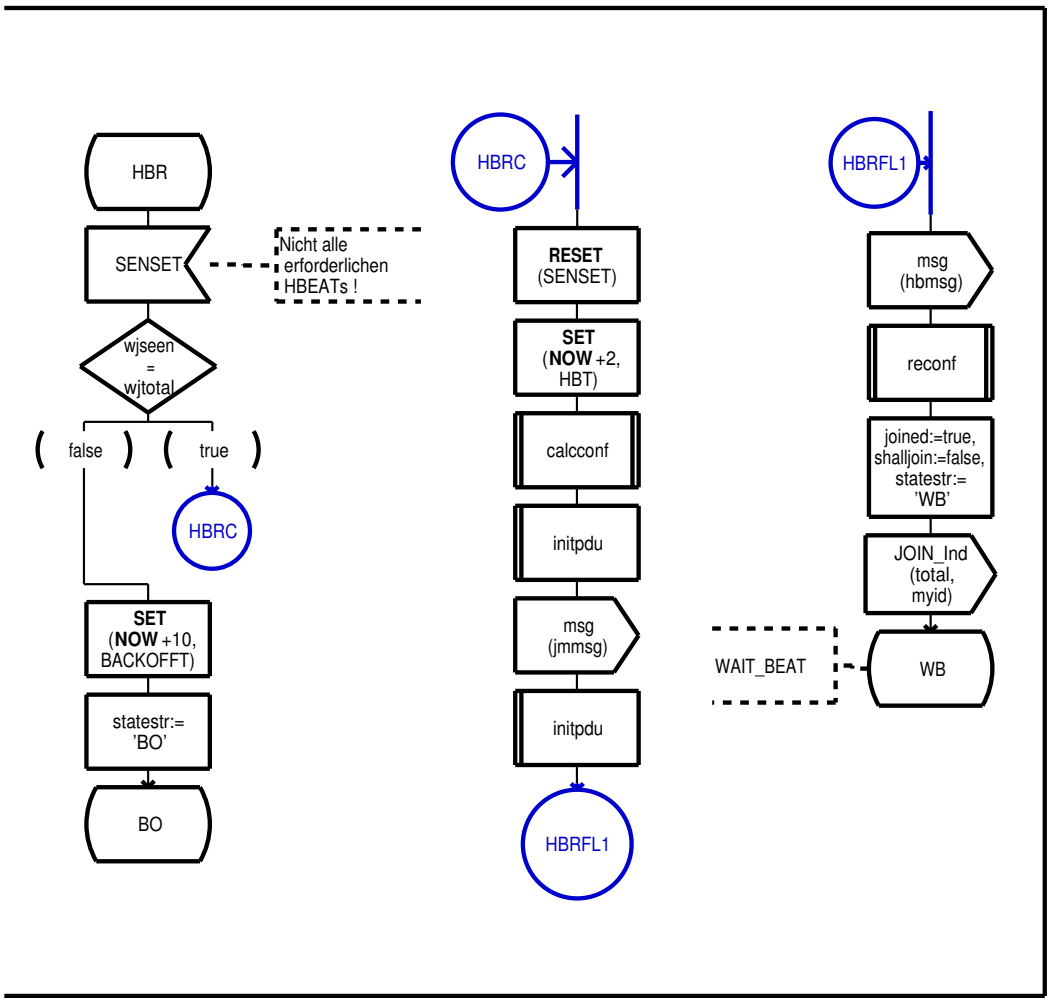


Abbildung B.14: Zustand: HB-Rec 3/3

- **Zustand “Backoff”**

Der Zustand “BO” (Backoff) wird für zwei Zwecke verwendet. Bei einer Kollision zweier oder mehrerer gleichzeitiger Beitrittsgesuche (“WantJoin”-Nachrichten) werden bis auf die Instanz mit der höchsten Priorität alle anderen Instanzen, die noch nicht in der Gruppe sind, in diesem Zustand abwarten. Außerdem wird in diesem Zustand verweilt, bevor auf eine “LEAVE_Req”-Aufforderung des Benutzers mit einem “LEAVE_Ind” geantwortet wird.

Wollen mehrere Instanzen quasi gleichzeitig der Gruppe beitreten, soll ihr Beitritt sequentialisiert werden, um zu häufige Fluktuationen in der Gruppenmitgliedschaft und eine uneinheitliche Sichtweise innerhalb der Gruppe zu vermeiden. Durch die Sequentialisierung

wird somit die Zeitspanne, in der es eine nicht einheitliche Sicht auf die Gruppe geben kann, minimiert.

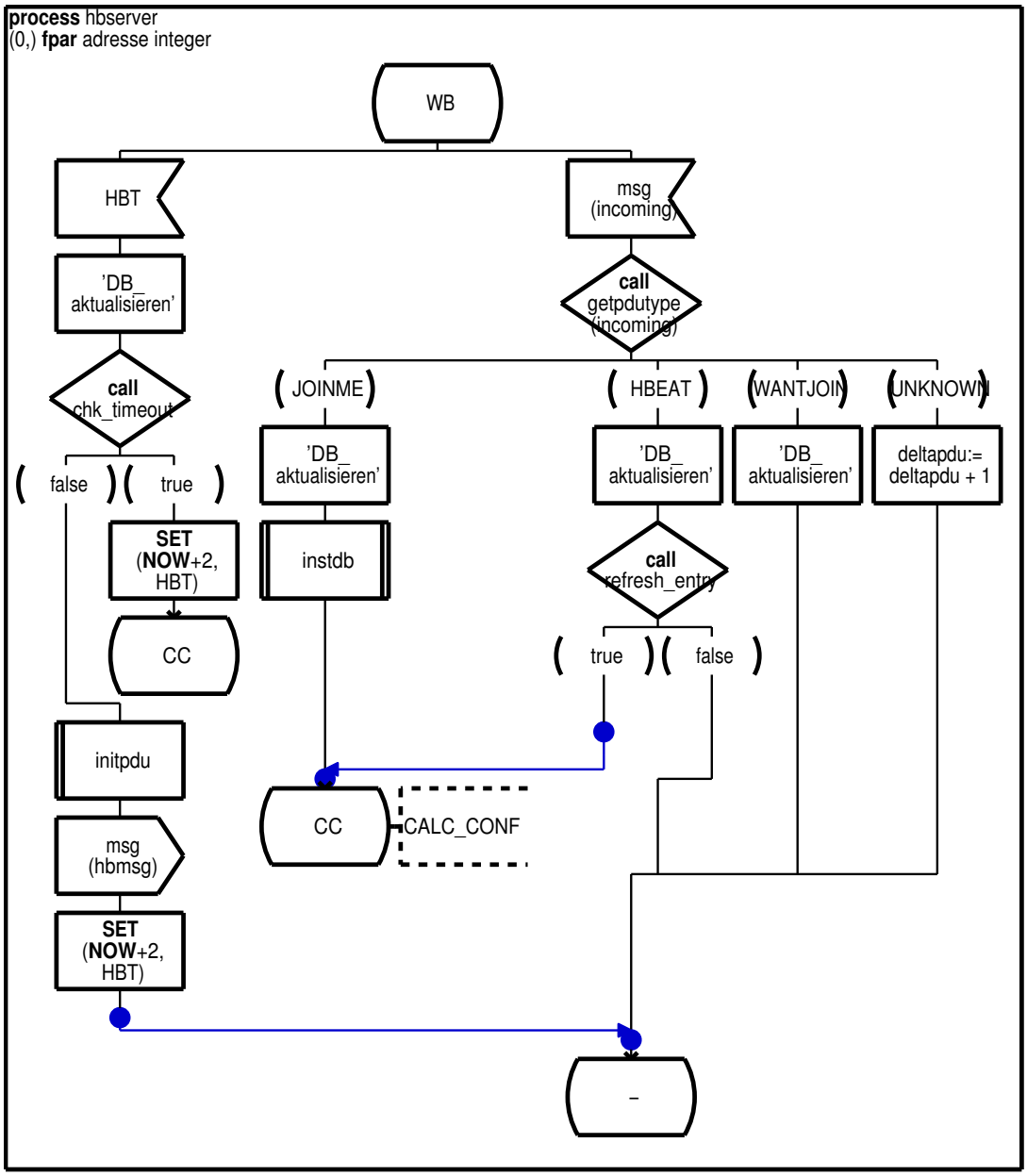


Abbildung B.15: Zustand: Wait-Beat

Für diese Sequentialisierung ist es erforderlich, einen Wartezustand einzuführen, in den die Instanzen mit geringer Priorität wechseln, wann immer sie erkennen, daß eine höher priorisierte Instanz ebenfalls der Gruppe beitreten möchte. In diesem "Backoff"-Zustand wird im wesentlichen auf das Ablaufen des Timers "BACKOFFT" gewartet. Die Wartezeit

ist groß gegenüber der Zeitdauer $HB_{intervall}$ und der Zeit, die eine priorisierte Instanz mit dem “Abtasten” in den Zuständen “WJS” und “HBR” verbringen kann. Hierdurch wird erreicht, daß die Aufnahme der priorisierten Instanz nicht durch zu häufige Nachrichten vom Typ “WantJoin” anderer Instanzen gestört wird. Es gilt also für einen beliebigen Zeitpunkt (t_0), an dem auf drei verschiedenen Instanzen (i, j, k) zufällig gleichzeitig jeweils einer der Zeitgeber gestellt wird, daß:

$$\forall t_0, i, j, k: \quad t_0 + \mathrm{HBT}_i < t_0 + \mathrm{SENSET}_j < t_0 + \mathrm{BACKOFFT}_k \tag{B.1}$$

Die Anforderungen an das Timing wurden bereits in Abschnitt 7.2.5.2 diskutiert. Für die Simulation wurden folgende Werte für die Timer verwendet:

$$\mathrm{HBT} = 2,\ \mathrm{SENSET} = 5\ \mathrm{BACKOFFT} = 10 \tag{B.2}$$

Der Timer “BACKOFFT” gibt die Wartezeit an, Timer “SENSET” die Abtastzeit und Timer “HBT” das $HB_{intervall}$. Diese Werte stellen für die Simulation eines Ausfalls eine günstige Wahl dar, da so in wesentlich geringerer Simulationszeit die Erkennung des Ausfalls durch andere Instanzen herbeigeführt werden konnte. Bei einer Implementation müßte der Timer “HBT” um eine Größenordnung kleiner eingestellt werden, um mit der in der Simulation verwendeten Einstellung für $HB_{maxlost} = 5$ die Ungleichungen aus Abschnitt 7.2.5.2 erfüllen zu können.

Wie in Abbildung B.16 zu erkennen ist, wird in Abhängigkeit davon, ob die Instanz unmittelbar vor dem Wechsel in den Zustand “BO” Gruppenmitglied war, eine “LEAVE_Ind”-Nachricht an den Benutzer gesendet. Dies ist aufgrund der Dienstspezifikation (vgl. Abschnitt 7.2.1) erforderlich, da ein Gruppenmitglied[5] nur aufgrund einer “LEAVE_Req”-Aufforderung in den Zustand “BO” wechseln kann. Der Folgezustand ist in jedem Fall der Startzustand.

• Standardverhalten in allen Zuständen

Es gibt zusätzlich einige Nachrichten, auf die in allen Zuständen gleich oder zumindest ähnlich reagiert werden muß. SDL bietet hierfür in der graphischen Notation den “*” als Zustandsnamen an. Dieser Platzhalter deutet an, daß die im folgenden aufgeführten Nachrichten in allen Zuständen bearbeitet werden müssen; mit Ausnahme der Zustände, die explizit in Klammern genannt sind.

In Abbildung B.17 ist diese Vorgehensweise für die drei Signale “STATUS_Req”, “JOIN_Req” und “LEAVE_Req” aufgeführt. Eine “STATUS_Req”-Aufforderung des Benutzers kann in allen Zuständen (außer “CC”) beantwortet werden, da lediglich der Inhalt von lokalen Variablen zurückgegeben werden muß. Interessanter ist dagegen der Fall, in dem

[5] Man beachte, daß im Gegensatz zum Gruppenmitglied ein Bewerber (potentielles Gruppenmitglied) genau dann in den Wartezustand wechselt, falls er Dissens in der Gruppe feststellt oder mindestens einen höher priorisierten Bewerber erkennt.

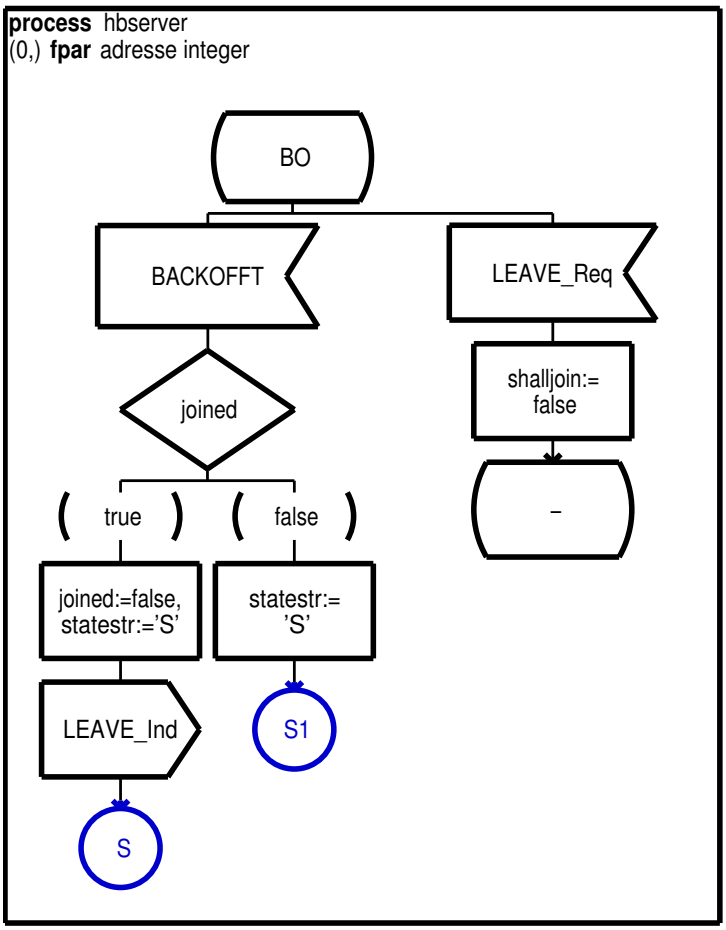


Abbildung B.16: Zustand: Backoff

der Benutzer eine "LEAVE_Req"-Aufforderung sendet. Es soll in den Zustand "BO" gewechselt werden, damit sichergestellt werden kann, daß die anderen Gruppenmitglieder das Ausscheiden aus der Gruppe erkennen. Erst danach wird dem Benutzer eine "LEAVE_Ind"-Mitteilung aus dem Zustand "BO" gesendet. Daher muß für den Zustand "BO" auch eine Sonderbehandlung spezifiziert werden (s.o.).

Bevor in den "BO"-Zustand gewechselt wird, werden zuvor alle eventuell aktiven Timer zurückgesetzt (RESET-Anweisungen). In der letzten Fallunterscheidung wird schließlich geprüft, ob direkt wieder in den Startzustand gewechselt werden kann. Es kann auf den Durchgang durch den Wartezustand jedoch nur verzichtet werden, falls die Instanz noch nicht Gruppenmitglied war (JOINED = false). In diesem Fall kann dem Benutzer direkt eine "LEAVE_Ind"-Mitteilung gesendet werden.

Auch eine "JOIN_Req"-Aufforderung kann in fast allen Zuständen gleich behandelt werden. Ist die Instanz bereits Gruppenmitglied, kann direkt die "JOIN_Ind"-Mitteilung an

den Benutzer gesendet werden. Ist die Instanz noch kein Gruppenmitglied, dann ist sie entweder durch eine zuvor empfangene “JOIN_Req”-Nachricht auf dem Weg dahin (Zustände “BO”, “WJS”, “HBR”) oder gerade gestartet worden, so daß das Signal im Zustand “PS” gesondert behandelt wird.

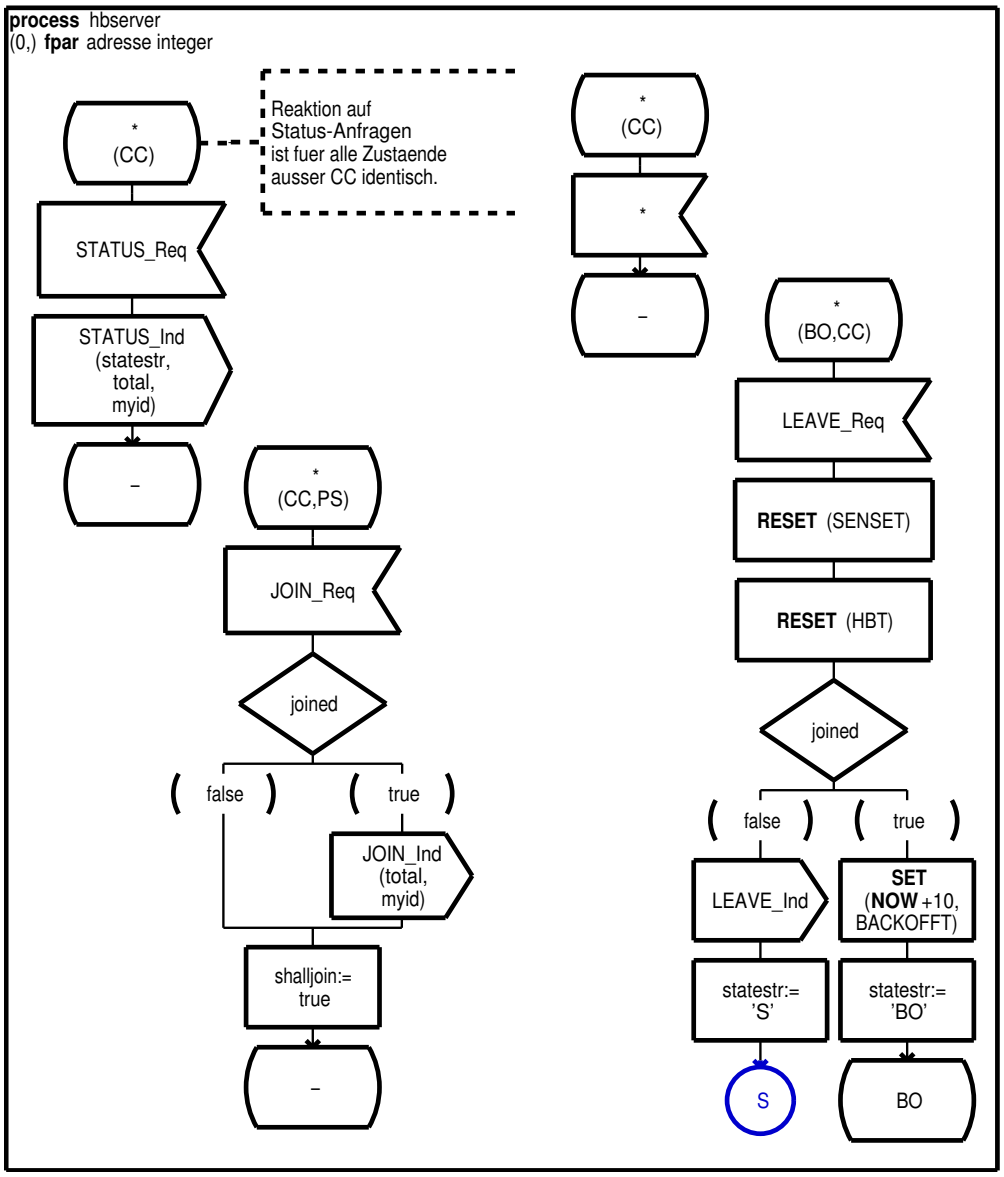


Abbildung B.17: Zusätzliche Nachrichten für mehrere Zustände

Der Zustand “CC” ist ausgeschlossen, da er immer nur durchlaufen wird und anschließend spontan in den Zustand “WB” gewechselt wird. Im SDL-Modell würde es ohnehin keinen Unterschied machen, ob im Zustand “CC” oder erst in einem darauf folgenden Zustand

auf eintreffende Signale reagiert wird, da hier Nachrichtenwarteschlangen mit unendlicher Kapazität angenommen werden (vgl. [81, S.222ff]). Auf den Ausschluß von Zustand "CC" hätte somit auch bei allen drei Signalen verzichtet werden können, da das Signal im Folgezustand verarbeitet wird. Bei einer Implementation kann dies jedoch einen erheblichen Unterschied ausmachen. Sind Berechnungen in einem Zustand sehr zeitaufwendig, könnten zwischenzeitlich Nachrichtenpuffer überlaufen und somit Nachrichten verloren werden. Der Ausschluß von "CC" modelliert diesen potentiellen Nachrichtenverlust somit direkt, da angedeutet wird, daß im Zustand "CC" nicht auf Nachrichten reagiert werden kann.

Schließlich werden in allen Zuständen all diejenigen Nachrichten, für die nicht explizit eine Verarbeitungsvorschrift angegeben wurde, vernichtet. Dieses Konstrukt ist zur Vervollständigung der Spezifikation erforderlich.

B.2.4 Prozeßstruktur (Admin-Prozeß)

Der "Admin"-Prozeß dient zur einfacheren Initialisierung einer bestimmten Anzahl von Instanzen für die Durchführung von Protokollsimulationen an Modellen, die automatisch aus der SDL-Spezifikation erstellt werden können. Der "Admin"-Prozeß wartet hierzu im Zustand "WAITUSER" auf eine Nachricht vom Typ "ANZ", mit der die Anzahl der zu startenden Protokollinstanzen angegeben werden kann.

Daraufhin startet "Admin" nacheinander die geforderte Anzahl von Protokollprozessen, teilt diesen als ihren Namen eine fortlaufende ID mit und meldet die Protokollinstanzen mit ihrer SDL-inhärenten Prozeßkennung beim "Netz"-Prozeß an (Nachricht "NEWPR"). Alternativ könnte der Benutzer die entsprechenden Kommandos auch im Zuge der Simulationsausführung manuell eingeben. Dieser manuelle Vorgang wäre jedoch fehleranfällig und könnte zu Fehlinterpretationen der Simulationsergebnisse führen. Der "Admin"-Prozeß vereinfacht das Starten der gewünschten Anzahl an Instanzen und deren Anmeldung beim "Broadcast"-Prozeß somit erheblich und führt diese Aufgabe garantiert fehlerfrei durch. Der Prozeß ist in Abbildung B.18 dargestellt.

B.2.5 Diverse Hilfsfunktionen

In der Beschreibung der einzelnen Zustände wurden diverse Hilfsfunktionen verwendet, die bisher nur informal erwähnt wurden. Für eine Validierung anhand eines Modells darf dieses Modell keine informalen Anteile mehr beinhalten. Die einzelnen Prozeduren und Makros[6] werden daher an dieser Stelle formal spezifiziert.

Für die Modellierung des Protokolls wurden Prozeduren für klar umrissene Teilaufgaben spezifiziert. Makros (Textersetzungen) wurden verwendet, um zusätzlich an bestimmten Stellen die Lesbarkeit durch eine Abstraktion von Einzelanweisungen zu erhöhen.

[6] Auch bei SDL ist ein Makro eine Textersetzung. Dieses Konzept wurde durch entsprechende Symbole sogar in die graphische Notation übernommen.

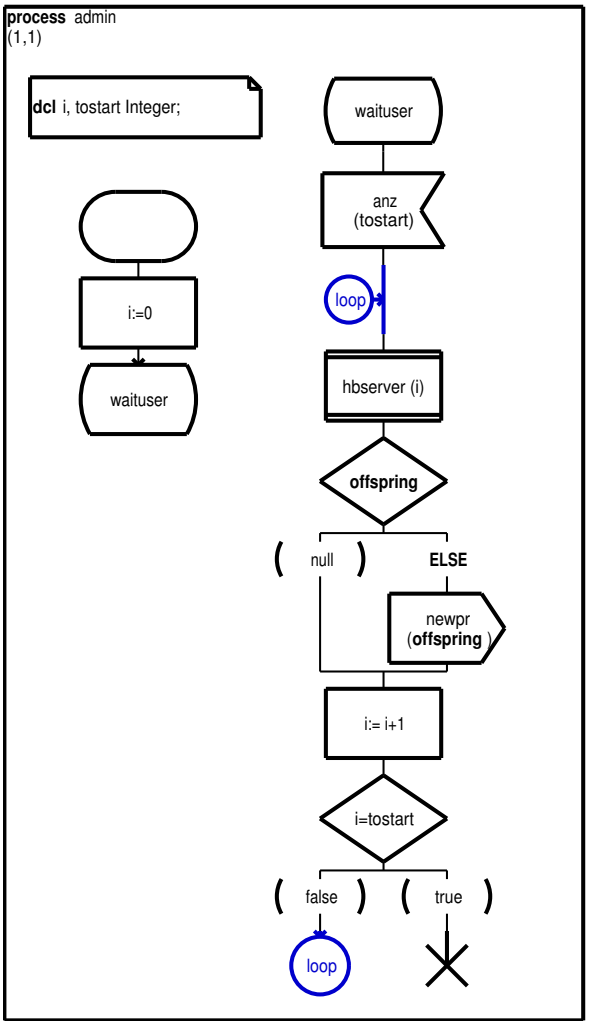


Abbildung B.18: Prozeß: Admin

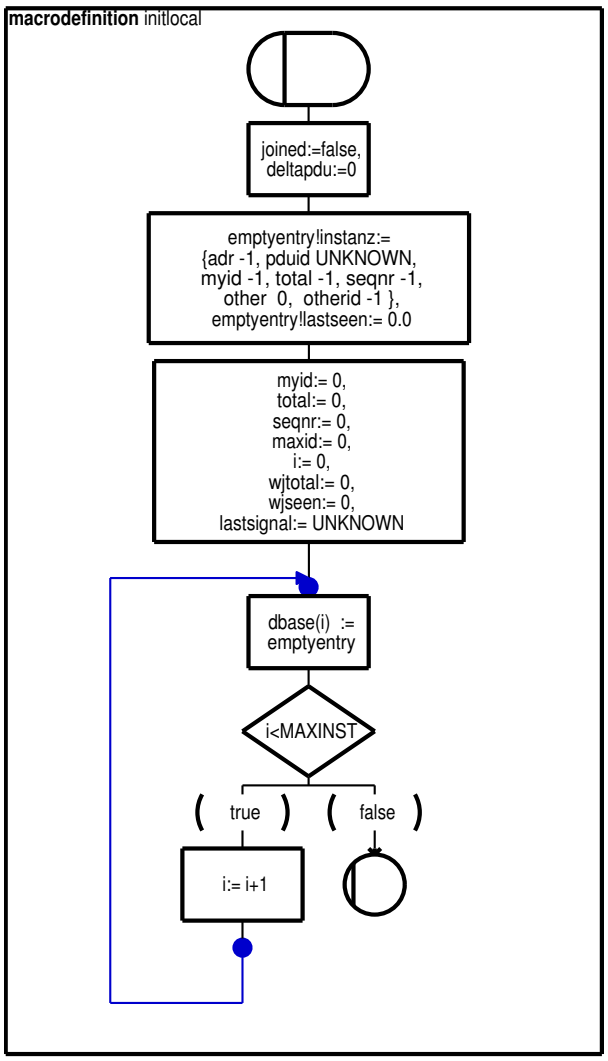


Abbildung B.19: Makro: Vorbelegung von lokalen Variablen

- **Makro “initlocal”**

Im Startzustand werden unter dem Kommentar-Task “INIT” einige Initialisierungsschritte in Makros gekapselt, die für das Verständnis der Protokollmaschine nicht wichtig sind. Für Simulation und Validation ist die genaue Spezifizierung jedoch erforderlich. Das Makro “initlocal” ist in Abbildung B.19 dargestellt. Neben der Vorbelegung skalarer Daten wird im wesentlichen die lokale Gruppenliste in einer Schleife gelöscht. Die einzelnen Felder der Liste mit Prozessorkennungen werden mit einem zuvor initialisierten leeren Eintrag “EMPTYENTRY” überschrieben. Die maximale Anzahl zu verwaltender Instanzen ist implementationsabhängig, da hier das genaue PDU-Format berücksichtigt werden muß. Die in der Simulation verwendete Anzahl ist in der Konstanten (SDL: synonym) “MAXINST”

abgelegt und relativ klein gewählt. Bei einer realistischen Wahl von ca. 90 Einträgen[7] würde allein diese Initialisierung bei der Simulation eine Unmenge an Textausgaben erzeugen. "MAXINST" ist daher nicht protokollinhärent, sondern simulationsspezifisch gewählt.

• **Makro "initpdu"**

Dieses Makro hat ganz ähnliche Aufgaben wie das Makro "INITLOCAL". Die allgemeinen, instanzspezifischen Daten werden in einen Platzhalter ("TEMPLATE") für eine PDU eingetragen. Danach wird dieses Template mit einer frischen Sequenznummer auf drei verschiedene Variablen für Nachrichten kopiert. Dadurch ist es möglich, nach Aufruf dieses Makros, eine in jeder Hinsicht aktualisierte Dateneinheit zu versenden und die Typauswahl bis zum Schluß zu verzögern. Das Makro (vgl. Abbildung B.20) wird daher in fast allen Zuständen der Protokollmaschine jeweils vor dem Senden einer Nachricht an das Medium aufgerufen.

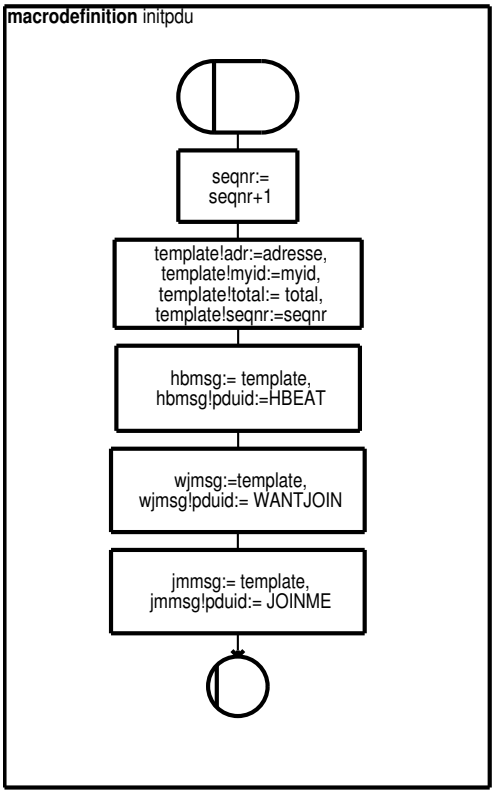


Abbildung B.20: Makro: Vorbelegung von PDUs

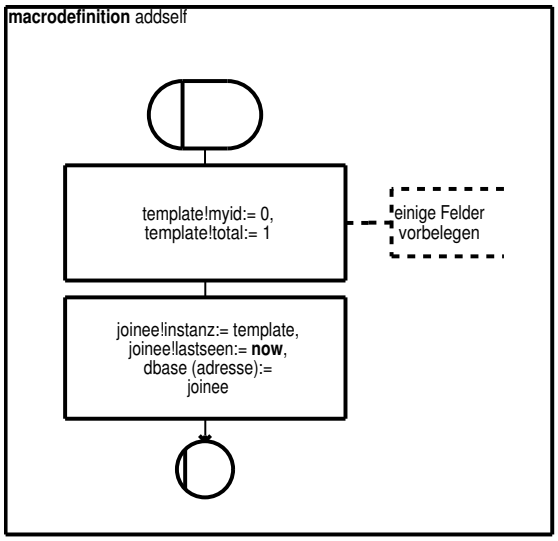


Abbildung B.21: Makro: Lokale Instanz in Gruppenliste eintragen

[7] Das PDU-Format aus Abschnitt 7.2.5.3 erlaubt unter Berücksichtigung der maximalen Rahmenlänge einer Ethernet PDU eine Anzahl von 92 Gruppenmitgliedern in der Gruppenliste.

• **Makro “addself”**

Das Makro “addself” trägt die lokale Instanz in die eigene Gruppenliste ein. Die Position in der Gruppenliste ist eindeutig durch die eigene Adresse festgelegt (vgl. Abbildung B.21).

• **Prozedur “getpdutypes”**

Die Prozedur “getpdutypes” erhält als Eingabeparameter den PDU-Typ einer eingetroffenen Nachricht. Es wird geprüft, ob es sich um einen bekannten Nachrichtentyp handelt oder um eine unbekannte Nachricht. Im Fall einer unbekannten Nachricht wird der “Pseudo-PDU-Typ” “UNKNOWN” zurückgeliefert, andernfalls der erkannte PDU-Typ (vgl. Abbildung B.22).

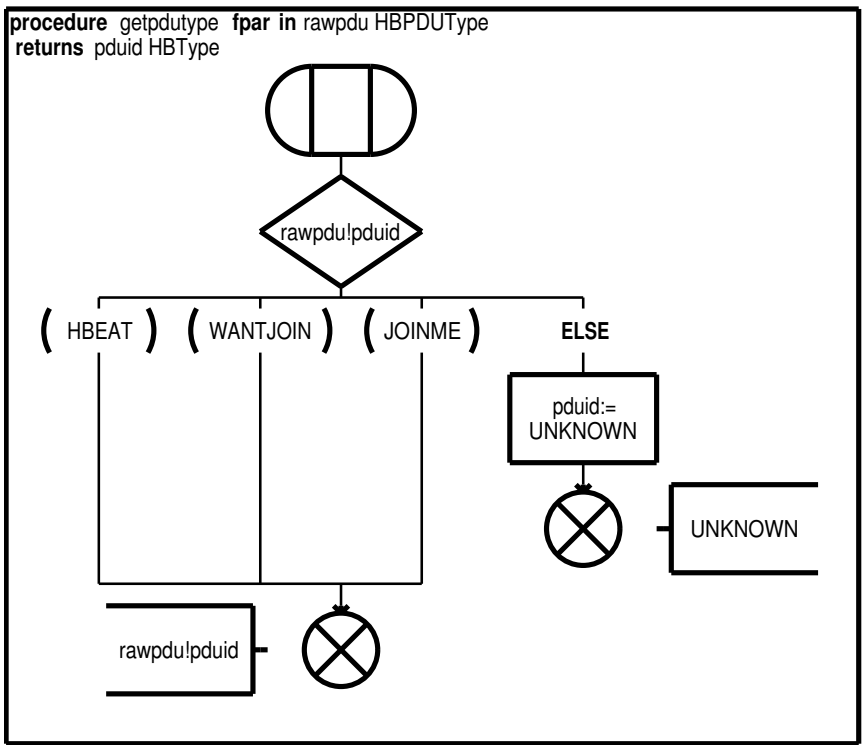


Abbildung B.22: Prozedur: PDU-Typ prüfen

An dieser Stelle wird eine wesentliche Einschränkung bei der formalen Modellierung eines Protokolls deutlich. Da SDL und viele andere formale Modellierungssprachen auf erweiterten endlichen Automaten beruhen, müssen alle vom Protokoll zu verarbeitenden

Nachrichten a priori bekannt sein.[8] Die Modellannahme ist in diesem Fall also, daß alle nicht bekannten Nachrichten auf den "Pseudo-PDU-Typ" "UNKNOWN" abgebildet werden. Diese Annahme ist in der Praxis jedoch durch nichts gerechtfertigt. Viele der in der Praxis auftretenden Probleme beruhen gerade auf falsch interpretierten Nachrichten.

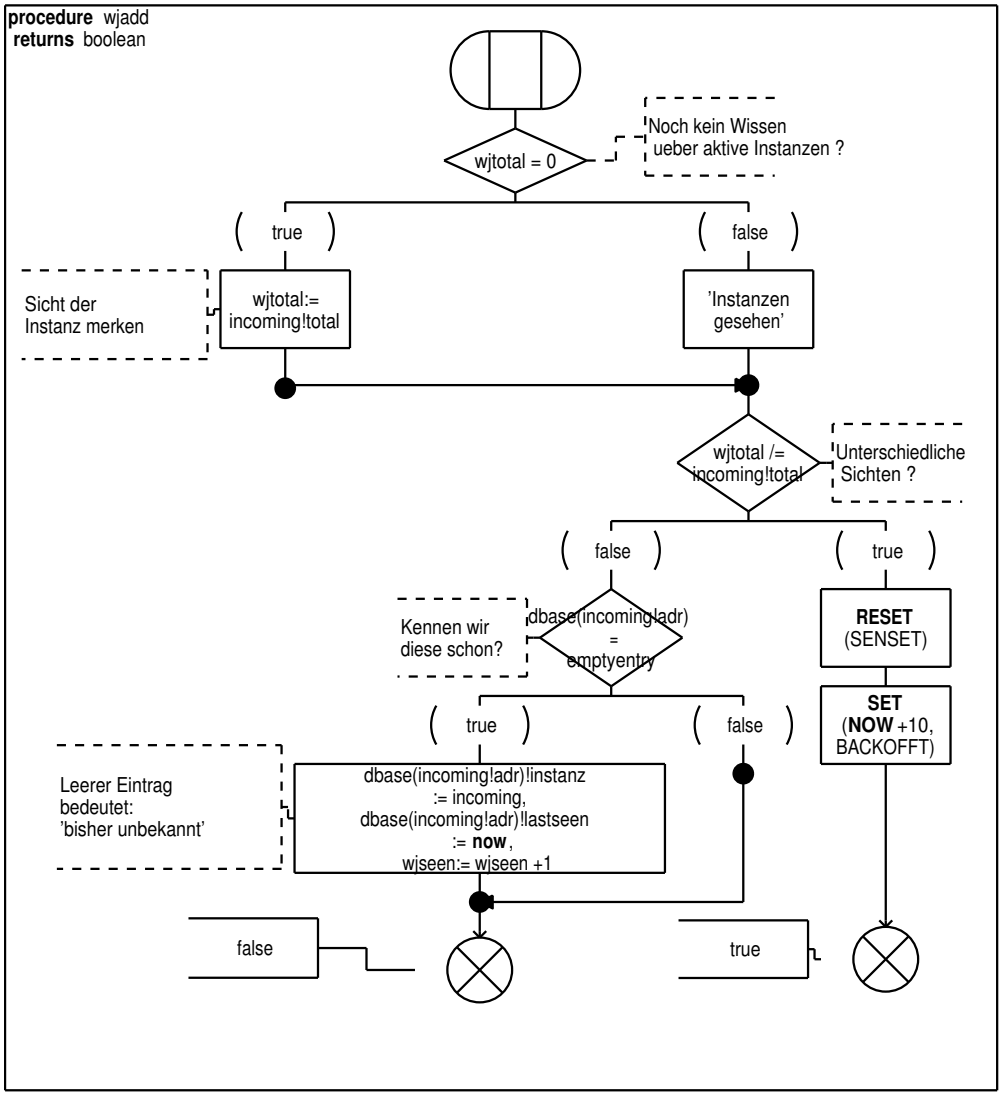


Abbildung B.23: Prozedur: Erstellen der lokalen Gruppenliste, Konsens/Dissens-Erkennung

[8] Formal sind die Nachrichten Eingabesymbole des endlichen Automaten. Die Menge dieser Eingabesymbole muß endlich sein. Im allgemeinen Fall muß ein robustes Protokoll jedoch in der Lage sein, eine beliebige Anzahl unbekannter Nachrichten zu erkennen.

• **Prozedur “wjadd”**

Die Prozedur “wjadd” wird in den Zuständen “WJS” und “HBR” aufgerufen, falls eine HB-Nachricht einer anderen Instanz eintrifft. Die Prozedur “wjadd” prüft, ob unter den anderen Gruppenmitgliedern Konsens herrscht, indem sie die in jeder HB-Nachricht enthaltene Gesamtanzahl der Gruppenmitglieder vergleicht. Ist diese Gesamtanzahl mit der bereits aus einer anderen Nachricht empfangenen Anzahl identisch, herrscht noch Konsens (wjtotal = incoming!total). Bei Dissens bereitet die Prozedur die Timer für einen Durchlauf durch den Wartezustand vor[9] und teilt dies durch den Rückgabewert (“true”) dem aufrufenden Prozeß mit (vgl. Abbildung B.23).

Herrscht Konsens, muß geprüft werden, ob der Absender der eingetroffenen HB-Nachricht bereits bekannt ist. Dazu wird dessen Name in der lokalen Gruppenliste gesucht. Ist an dieser Stelle ein leerer Eintrag, dann wurde eine neue Instanz entdeckt. Die Instanz wird in die lokale Gruppenliste eingetragen und die Anzahl der bisher entdeckten Instanzen aktualisiert. Die Prozedur terminiert mit dem Rückgabewert “false”, solange aus ihrer Sicht Konsens herrscht.

• **Prozedur “instdb”**

Diese Prozedur trägt eine neue Instanz in die Gruppenliste ein. Sie wird normalerweise als Reaktion auf eine “JoinMe”-Nachricht eines Bewerbers von den Gruppenmitgliedern aufgerufen, um den neuen Bewerber in die lokalen Gruppenlisten einzutragen (vgl. Abbildung B.24).

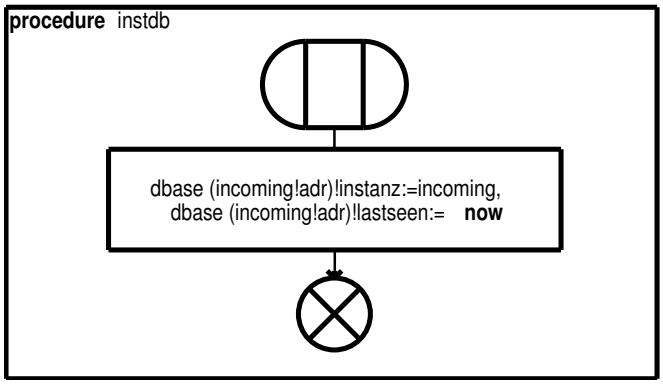


Abbildung B.24: Prozedur: Eintragen einer neuen Instanz in die Gruppenliste

[9] Auch in diesem Fall muß mit der Modellierung dem formalen Modell von SDL genügt werden. Es ist aus der Prozedur kein direkter Übergang in den “BACKOFF”-Zustand möglich, da diese Zustände aus der Prozedur nicht erreichbar sind. Die Prozedur kann eigene Zustandsübergänge initiieren, jedoch nicht den Kontrollfluß der aufrufenden Prozedur verändern.

• **Prozedur “refresh-entry”**

Die Prozedur “refresh-entry” (vgl. Abbildung B.25) wird im Zustand “WB” für jede eingetroffene HB-Nachricht aufgerufen. Sie sucht den Sender in der lokalen Gruppenliste und aktualisiert dessen Zeitstempel. Damit weiß die lokale Instanz, wann sie die andere Instanz zuletzt gesehen hat. Diese Information wird periodisch durch die Prozedur “chk-timeout” überprüft (s.u.).

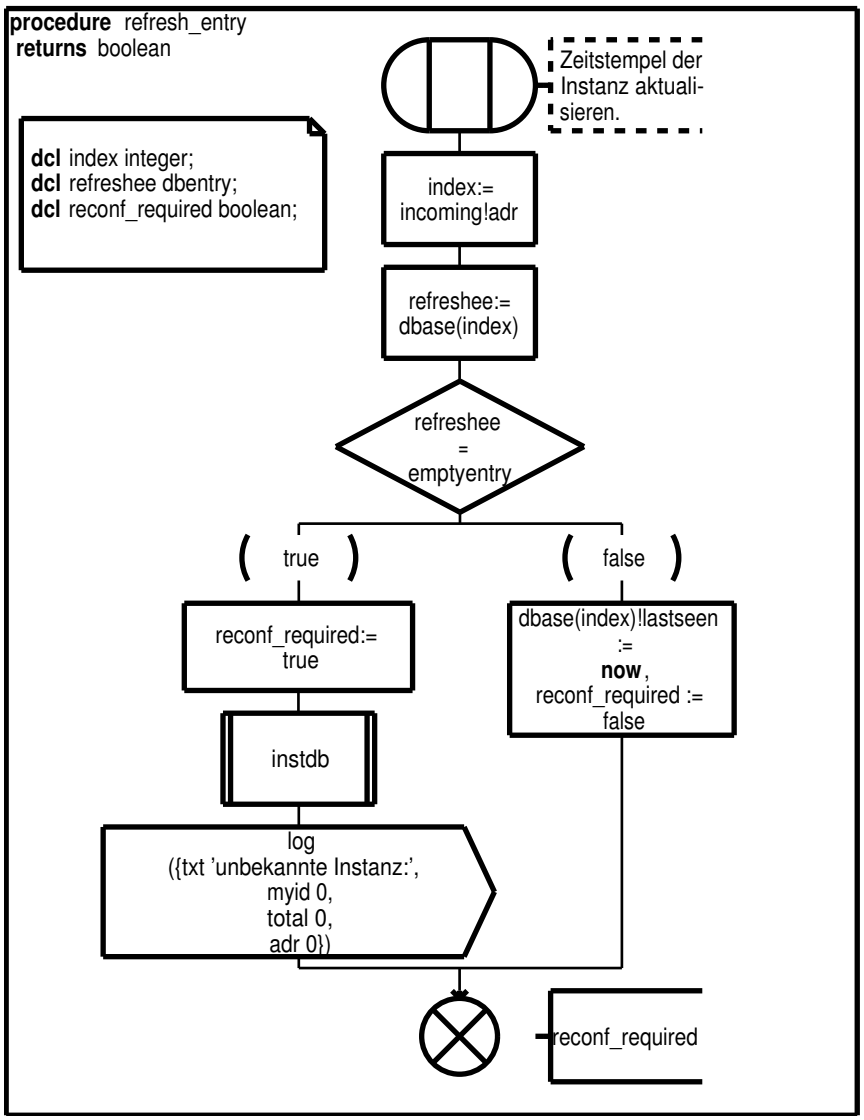


Abbildung B.25: Prozedur: Aktualisieren der Zeitstempel bei Empfang einer HB-Nachricht, nachträgliches Eintragen eines neuen Bewerbers

Zur Erhöhung der Robustheit wird zusätzlich immer geprüft, ob der Absender bereits bekannt ist. Normalerweise muß er aufgrund einer “JoinMe”-Nachricht bereits bekannt sein (in die lokale Gruppenliste eingetragen sein). Da das GMP jedoch mit Nachrichtenverlusten rechnen muß, könnte auch eine “JoinMe”-Nachricht verloren gehen. Da jedoch die notwendigen Informationen für die Aktualisierung der lokalen Gruppenliste auch in jeder HB-Nachricht vorhanden sind, kann die lokale Instanz eine unbekannte Instanz sofort bei Empfang der HB-Nachricht in die Gruppenliste eintragen. Die Prozedur meldet über den Rückgabewert “true” (reconf_required=true), daß die lokale Gruppenliste anschließend geprüft werden muß.

• Prozedur “chk-timeout”

Im Zustand “WB” wird periodisch bei Ablauf des Timers “HBT” diese Prozedur aufgerufen, um zu prüfen, ob sich eine Instanz zu lange nicht mehr gemeldet hat. Dazu werden die gültigen Einträge der lokalen Gruppenliste nacheinander selektiert und die dort eingetragenen Zeitstempel der letzten empfangenen HB-Nachricht (Prozedur: “refresh-entry”) mit der aktuellen Zeit verglichen. Ist die Zeitdifferenz größer als “MAXTIMEOUT” (MAXTIMEOUT = $HB_{intervall} * HB_{maxlost}$), wird der Eintrag gelöscht und im Rückgabewert vermerkt, daß eine Neuberechnung der Gruppenliste erforderlich ist (reconf_required=true, vgl. Abbildung B.26). Die lokale Instanz muß beim Absuchen der Gruppenliste zusätzlich erkennen, ob sie ihren eigenen Eintrag selektiert hat (loopvar = adresse). In diesem Fall darf der Zeitstempel nicht verglichen werden, da die lokale Instanz keine HB-Nachrichten an sich selbst sendet und somit der Zeitstempel ohne Bedeutung ist.

• Prozedur “calcconf”

Falls eine Änderung an der Gruppenzusammensetzung erkannt wurde, muß die neue Gesamtanzahl aller Gruppenmitglieder berechnet werden. Außerdem kann sich durch ein neues oder ausgeschiedenes Mitglied die Position der lokalen Instanz innerhalb der Gruppe verändern. Die Prozedur “calcconf” berechnet diese beiden Werte neu (vgl. Abbildung B.27). Dazu wird die lokale (möglicherweise durch “chk_timeout” oder “refresh_entry” veränderte) Gruppenliste der Reihe nach durchsucht. Über einen Zähler “SUM” wird die Anzahl der gültigen Einträge aufaddiert. Gleichzeitig wird geprüft, ob der gerade selektierte Eintrag die lokale Instanz beschreibt. Ist dies der Fall (loopval = adresse), wird der aktuelle Zählwert zusätzlich in der Variablen “ID” vermerkt. Nach dem Durchsuchen der gesamten Liste befindet sich demnach die neue Gesamtzahl aller Gruppenmitglieder in “SUM” und die eigene Position innerhalb der Gruppenliste in der Variable “ID”. Mit diesen beiden Werten werden vor dem Verlassen der Prozedur die globalen Variablen “TOTAL” und “MYID” überschrieben, die in den darauf folgenden Nachrichten die Sicht dieser Instanz auf die Gruppenzusammensetzung widerspiegeln.

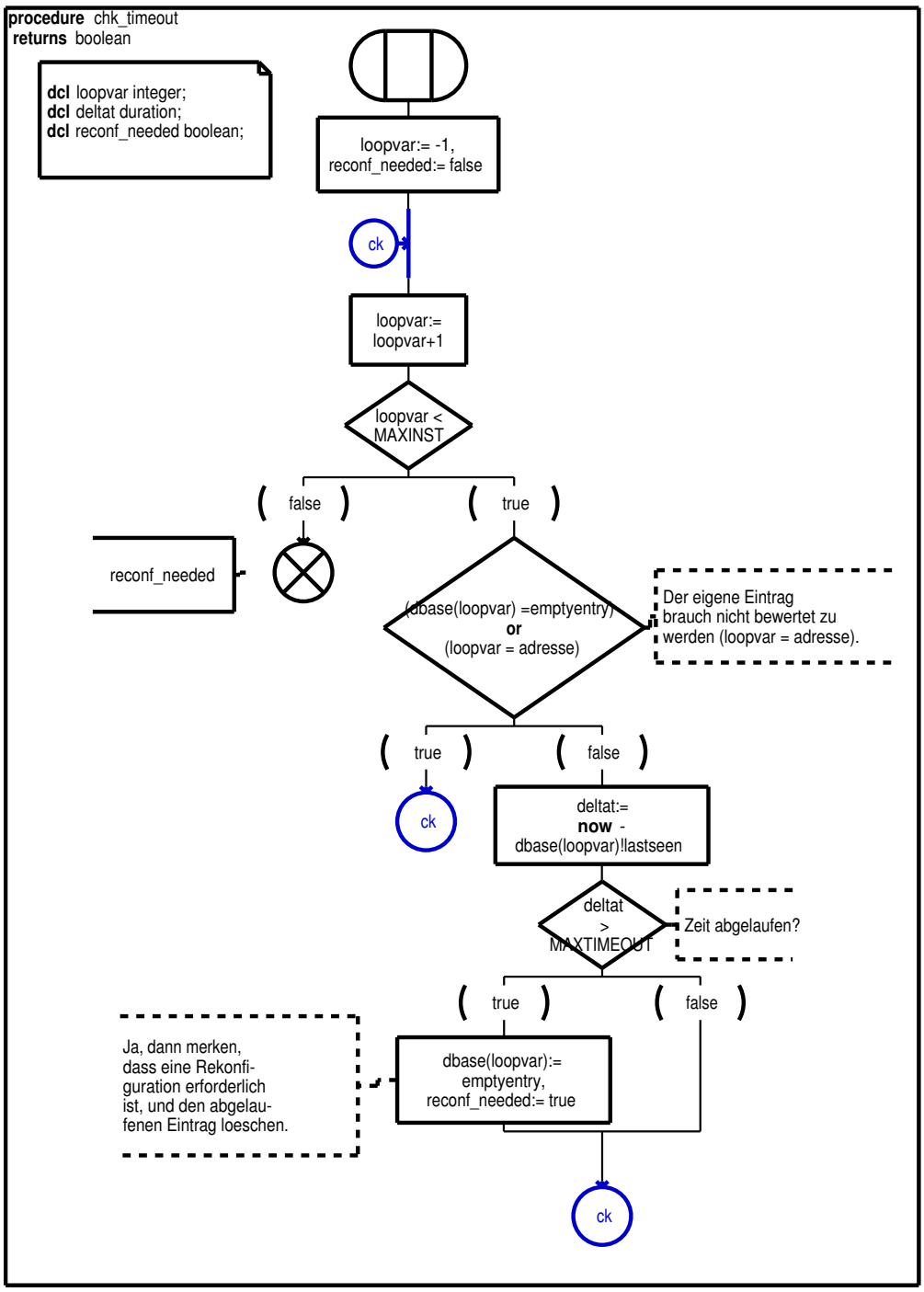


Abbildung B.26: Prozedur: Überprüfen der Zeitstempel in der Gruppenliste

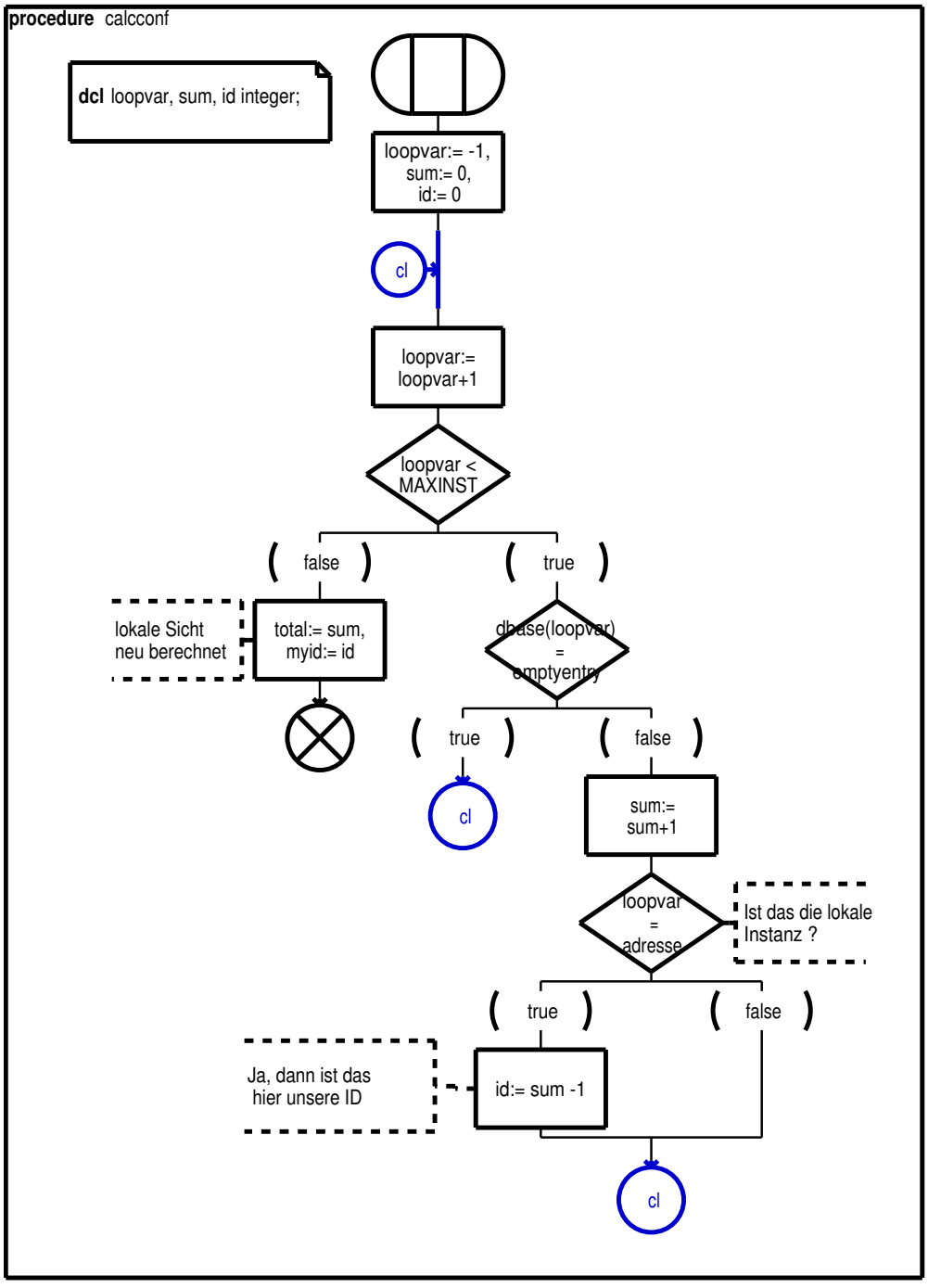


Abbildung B.27: Prozedur: Neuberechnen der Gruppenzusammensetzung

• **Prozedur “reconf”**

Die Prozedur “reconf” wird bei Veränderungen in der lokalen Gruppenliste aufgerufen und teilt dem Benutzer mit einer “STATUS_IND”-Nachricht die neue Gruppenzusammensetzung mit. Die zusätzlich versendete “Logmeldung” trägt die gleichen Informationen wie die “STATUS_IND”-Nachricht. Konfigurationsveränderungen werden somit auch dann protokolliert, falls der Benutzer die “STATUS_IND”-Nachricht ignoriert. In Abbildung B.28 sind zwei verschiedene Modellierungsmöglichkeiten für das Versenden von Nachrichten aufgezeigt. Bei der “Logmeldung” wurde ein eigenes PDU-Format spezifiziert, das eine Selektion der gesendeten Informationen über ihren Namen ermöglicht. Bei der “STATUS_IND”-Nachricht wurde dieses PDU-Format implizit spezifiziert. Die einzelnen Informationen werden direkt in das Signal eingetragen. Das PDU-Format ergibt sich somit implizit als Sequenz dieser Informationen.

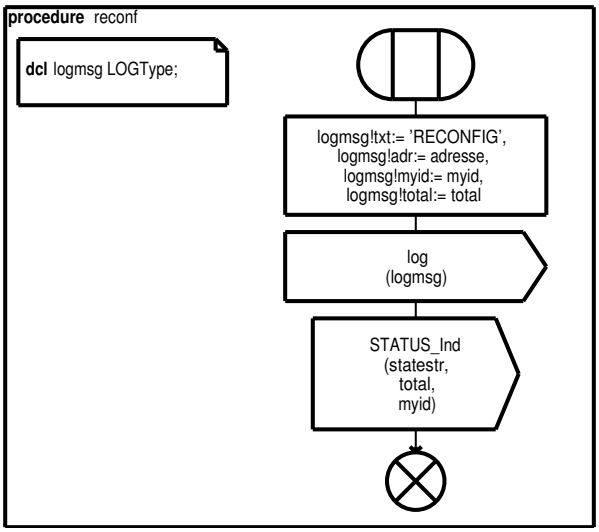


Abbildung B.28: Prozedur: Neue Gruppenzusammensetzung dem Benutzer mitteilen

B.3 Zusammenfassung

In diesem Anhang wurde eine formale Spezifikation des in Kapitel 7 diskutierten, leichtgewichtigen GMPs vorgenommen. Die Spezifikation umfaßt alle wesentlichen Aspekte des Protokolls, so daß die Funktionsweise durch Simulation validiert werden kann. Die Abbildungen 7.5 – 7.7 zeigen beispielsweise zwei Abläufe, die von einem aus der Spezifikation automatisch generierten Simulationsmodell erstellt wurden.

Unerwarteter, zusätzlicher Aufwand war erforderlich, um ein “Broadcast”-Medium zu spezifizieren, das als wesentliche Voraussetzung für das Protokoll in Kapitel 7 benannt wurde.

SDL unterstützt multilaterale Kommunikationsbeziehungen leider nicht durch einen ad hoc verwendbaren Mechanismus. Dieser scheinbare Nachteil (Mehraufwand durch eigene Spezifikation) erweist sich beim zweiten Blick jedoch als Vorteil. Der Vorteil an einer eigenen Spezifikation des Mediums, wie in Abschnitt B.1 geschehen, ist die Tatsache, daß so auch beliebige technische und maliziöse Nachrichtenmanipulationen als Teil des Übertragungsmediums simuliert werden könnten. Es wäre somit denkbar, die in Kapitel 8 diskutierten Angriffe auf das GMP, durch eine Erweiterung der Spezifikation des “Broadcast”-Mediums, ebenfalls zu simulieren. Eine um die zusätzlichen Mechanismen zur Absicherung des Protokolls erweiterte Spezifikation des GMPs (ebenfalls in Kapitel 8 diskutiert), würde dann eine noch detailliertere Simulation der Angriffe und Gegenmaßnahmen ermöglichen.

Anhang C

Durchführung von Messungen und Erweiterungen an Meßwerkzeugen und Gerätetreibern

Die relativ knappe Darstellung der Meßergebnisse aus der Leistungsbewertung der sequentiellen und der parallelen Packet Screen täuschen über den damit verbundenen Aufwand hinweg. Es ist allgemein akzeptiert, Messungen als Grundlage für eine Leistungsbewertung oder zur Validierung/Kalibrierung eines analytischen/simulativen Modells heranzuziehen. Daher ist die ordentliche Durchführung der Messungen und die Auswertung der Meßergebnisse besonders wichtig. Fehler bei der Messung können zu einer falschen Modellbildung oder Modellkalibrierung führen; möglicherweise wird sogar aufgrund systematischer Fehler bei der Meßdatenerhebung ein adäquates Modell verworfen. In diesem Abschnitt soll daher motiviert werden, welche Probleme und Fallstricke speziell bei der meßtechnischen Bewertung von Netzkomponenten auftreten können.

Die hierbei gesammelten Erfahrungen führten zu teilweise ganz erheblichen Erweiterungen an frei verfügbaren Meßwerkzeugen und zur Neuentwicklung von eigenen "Tools" (vgl. [26, 78]).

In Abschnitt C.3 werden die wesentlichen Erweiterungen an den Gerätetreibern vorgestellt. Diese Erweiterungen an "Kernel-Modulen" komplettieren die in Kapitel 9 diskutierte Implementation des Prototypen.

C.1 Fallstricke bei der Meßdatenerfassung

Als extrem aufwendig stellte sich die Elimination von systematischen Effekten durch folgende Fehlerquellen heraus.

C.1.1 Fehlerquelle: "Plug and Play"-Netzanschluß

Die schematischen Darstellungen für die verschiedenen Versuchsaufbauten aus den Abbildungen 3.2 bis 3.5 mußten physikalisch implementiert werden. Je nachdem, ob Untersuchungen zu "shared" oder "switched" Fast Ethernet durchgeführt wurden, mußten zwischen den Meßsystemen und dem SuL entweder Hubs ("Multiport Repeater") oder Switches ("Multiport Bridges") eingesetzt werden.

Nach einem Neustart der Meßsysteme und der SuL (vor jeder Meßreihe) setzt auf allen beteiligten Systemen die automatische Konfigurierung der Fast Ethernet-Netzschnittstellen (kurz FE-Karten) ein. Jede FE-Karte handelt mit der Hub/Switch-Schnittstelle, an die sie mit einem Kabel angeschlossen ist, folgende Parameter aus:

Betriebsmodus Es wird zwischen "Halbduplex" (Switch und Hub) und "Vollduplex" (nur Switch) unterschieden. Beim "Halbduplex"-Betrieb wird für das Senden und Empfangen von Dateneinheiten dasselbe Adernpaar des Kabels benutzt. Die maximale Übertragungsgeschwindigkeit ist dann 100 Mbit/s zwischen den Schnittstellen, und es können Kollisionen auftreten.

Beim "Vollduplex"-Betrieb werden für das Senden- und Empfangen zwei verschiedene Adernpaare des Kabels benutzt. Die maximale Übertragungsgeschwindigkeit ist dann 200 Mbit/s zwischen den Schnittstellen, und es können keine Kollisionen auftreten.

Geschwindigkeit Es wird verhandelt, mit welcher Geschwindigkeit übertragen werden soll. Hubs unterstützen oft nur eine Geschwindigkeit (100 Mbit/s bei FE). Switches hingegen ermöglichen meist auch die Unterscheidung zwischen Ethernet- und Fast Ethernet-Betrieb, so daß zwischen den Geschwindigkeiten 10 vs.100 Mbit/s gewählt werden kann bzw. zwischen 20 und 200 Mbit/s im "Vollduplex"-Betrieb.

Bei der verwendeten Hardware war das Ergebnis dieser Verhandlungen (normalerweise wird die größte gemeinsam unterstützte Geschwindigkeit beider Schnittstellen gewählt) jedoch nicht deterministisch. Obwohl gleichartige FE-Karten in SuL und Meßsystemen verwendet wurden, war das Ergebnis dieser automatischen Konfigurierung nicht immer identisch. Bei mehreren Versuchwiederholungen gleicher Meßreihen trat dadurch ein systematischer Fehler auf. Die automatische Konfigurierung variierte den Versuchsaufbau. Die eigentlich als gleichartige Wiederholungen geplanten Versuche stellten sich als verschiedenartige Versuche heraus. Die Ergebnisse sind somit nicht vergleichbar.

Nach Erkennen dieser Fehlerquelle wurden die Schnittstellen auf allen beteiligten Systemen manuell konfiguriert.

C.1.2 Fehlerquelle: falsche und ungleichmäßige Konfigurierung von Monitor, Lastgenerator und der Systeme unter Last

Da sowohl die Meßsysteme als auch die SuL normale Workstations waren, mußte regelmäßig ein Neustart durchgeführt werden, um wieder einen definierten Systemzustand vor

den Messungen herzustellen. Dies erforderte auch die gleichartige Konfigurierung der automatisch zu startenden Prozesse. Zwei Instanzen einer parallelen Packet Screen sind beispielsweise nur dann identisch, wenn neben der identischen Hardware-Ausstattung auch die gleichen Prozesse während der Messung aktiv sind.

Eine zweite Meßreihe mit identischen Konfigurationskenngrößen ist darüber hinaus nur dann eine Wiederholung einer anderen Meßreihe, wenn ausgeschlossen ist, daß zwischenzeitlich andere Prozesse gestartet werden. Der "cron"-Daemon startet auf typischen UNIX-Workstations beispielsweise zu festgelegten Zeiten automatisch Prozesse für Routine-Aufgaben. Eine genaue Planung der Abläufe auf den beteiligten Systemen ist somit unumgänglich, um nicht tageszeitlich bedingte Schwankungen durch automatisch gestartete Prozesse in die Meßreihen zu bekommen.

Eine Betriebssystemerneuerung (Update oder Patch) ist zu vermeiden, solange danach angefertigte Messungen mit vorher angefertigten Messungen verglichen werden sollen. Durch das Einspielen von Patches können beispielsweise neue Prozesse nach einem Neustart des Rechners gestartet werden und Systemkonfigurationen können verändert sein. Ein Betriebssystem-Update erstellt ein völlig neues System mit anderem Verhalten und anderen Konfigurationsmöglichkeiten. Alle Meßreihen wurden daher mit derselben Betriebssystemversion durchgeführt, um eine Vergleichbarkeit der Meßergebnisse zu gewährleisten.

C.1.3 Fehlerquelle: Störlast im Testnetz

Störlasten im Testnetz können Messungen verfälschen. Es muß somit sichergestellt werden, daß während einer Meßdatenerhebung keine zusätzlichen Dateneinheiten über das selbe Netz versendet werden. Andere Kommunikationsbeziehungen als die für die Messungen erforderlichen sind in der Regel zu vermeiden. Beispielsweise sollten SuL und Meßsysteme während einer Messung keine NFS-Partitionen in das lokale Dateisystem einhängen.

Weitere Störungen können durch aktive Netzkomponenten erfolgen. Beispielsweise mußten auf den verwendeten Switches zwei Protokolle abgeschaltet werden, die eine Störlast erzeugten ("Spanning Tree Protocol" und "Cisco Discovery Protocol").

C.1.4 Fehlerquelle: Anzahl der Prozessoren auf den Meßsystemen

Abhängig vom verwendeten Betriebssystem kann durch die Anzahl der CPUs auf den Systemen ein unangenehmer Seiteneffekt auftreten. Die Meßsysteme waren ursprünglich jeweils mit zwei Prozessoren ausgestattet, um eine möglichst hohe Paketlast generieren und empfangen zu können.

Bei aufeinanderfolgenden Meßreihen stellte sich jedoch heraus, daß der Prozeß des Meßsystems mal auf einer mal auf der anderen CPU zur Ausführung kommt, genauso wie andere

zeitgleich laufende Prozesse. Dieser Umstand führte u.a. zu dem Problem der *Lastschwankungen bei gleichartigen Meßreihen*. Zwar konnte eine Instanz des Lastgenerators, wenn sie einmal gestartet war, eine bestimmte Anzahl von Dateneinheiten hochgradig genau während der Meßwiederholungen reproduzieren (erkennbar an den kleinen Vertrauensintervallen), jedoch generierte eine andere Instanz dieses Prozesses bei einer Wiederholung teilweise eine ganz andere Last (ebenfalls wieder hochgradig genau, vgl. Abbildung C.1).

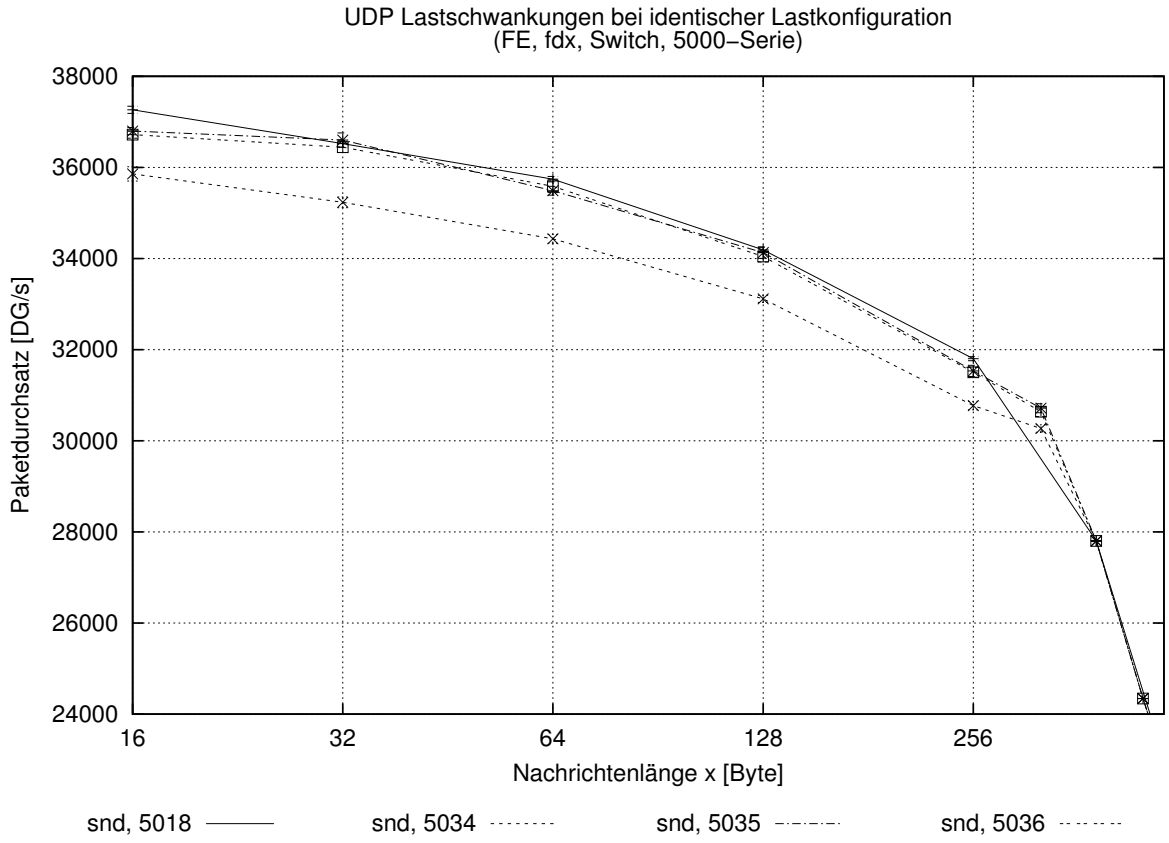


Abbildung C.1: Schwankungen der Sendeleistung bei Versuchswiederholungen

Beispielsweise wurden bei 32 Byte langen Nachrichten in der Meßreihe "5018" 36522 Datagramme pro Sekunde mit einer Standardabweichung von $0,02$ Datagrammen pro Sekunde generiert. In der Wiederholung "5034" hingegen lediglich 35251 Datagramme pro Sekunde, allerdings mit einer ebenfalls sehr geringen Standardabweichung von nur 0.03 bei 30 Wiederholungen zu je 10 Sekunden. Das ist ein Unterschied von ca. 3.5%.

Wünschenswert ist jedoch eine Überlappung der Vertrauensintervalle. Nur so kann man eine "identische" Last annehmen. Dieses Verhalten führte in einem ersten Schritt dazu, daß der Lastgenerator nur noch mit einer CPU bestückt wurde. Es traten jedoch weiterhin zu große Schwankungen auf. In einem nächsten Schritt wurde ein vergleichbares Meßwerkzeug erstellt (programmiert), da nicht ausgeschlossen werden konnte, daß das ursprünglich

eingesetzte Werkzeug für die Schwankungen verantwortlich war. Dies führte ebenfalls zu keinem Erfolg, sondern bestätigte die beobachtbaren Schwankungen in der Sendeleistung. Schließlich wurde das bereits etablierte Meßwerkzeug erweitert, um Leistungsvorgaben in Abhängigkeit von der PDU-Größe zu spezifizieren (s.u.). Hierdurch konnten die generierten Lasten angeglichen werden, so daß sich die Vertrauensintervalle in allen Punkten überschnitten.

C.2 Erweiterungen an den Meßwerkzeugen

Im Verlauf der Meßdatenerfassung stellte sich heraus, daß das ursprünglich ausgewählte Werkzeug "netperf[1]" nicht geeignet war, um alle erforderlichen Meßaufgaben zu ermöglichen.

C.2.1 Statistische Absicherung

Als erstes Problem fiel auf, daß die Vertrauensintervalle nicht ausgegeben wurden. Zwar ermittelt das Tool aufgrund von Wiederholungen, ob ein Meßwert mit einer bestimmten Genauigkeit reproduziert werden kann, die absolute Größe des Vertrauensintervalls wurde jedoch nicht ausgegeben.

Es mußten daher die Statistikfunktionen erweitert werden, so daß das Vertrauensintervall berechnet und als Teil des Meßergebnisses mit ausgegeben wird.

C.2.2 Trennung der Wegwahl für gleichzeitige Kommunikationsbeziehungen

Die Meßwerkzeuge bauen vor jeder Leistungsmessung eine Kontrollverbindung per TCP zwischen Lastgenerator und Monitor auf. Über diese Verbindung werden für die anstehenden Messungen die Parameter ausgetauscht und SAP-Adreßinformationen übermittelt. Nach Durchführung einer Messung überträgt der Monitor die Meßergebnisse an den Lastgenerator, der diese zusammen mit seinen Lastdaten ausgibt.

Für die Messungen bei gestörter Kommunikation, beispielsweise bei der Simulation des Ausfalls einer Packet Screen-Instanz in Abschnitt 6.1.3, ist es erforderlich, die Konfigurationsverbindung über ein separates Netz aufzubauen, so daß diese Dateneinheiten nicht von der (simulierten) Störung beeinflußt werden. "Netperf" (der Lastgenerator) und "Netserver" (der Monitor) mußten daher um entsprechende Funktionen erweitert werden, die die alternative Wegwahl für die Konfigurationsverbindungen implementieren.

[1] Frei erhältlich unter `http://www.netperf.org/`

C.2.3 Synchronisation mehrerer Meßsysteme

Für die Messungen nach dem in Abbildung 3.5 dargestellten Schema mußten umfangreiche Erweiterungen vorgenommen werden. Messungen mit mehr als einem Lastgenerator waren erforderlich, da die Parallelisierung bereits mit drei Instanzen (mit vier Instanzen bei Dual-CPU-Lastgeneratoren) einen so hohen Durchsatz ermöglichte, daß ein einzelner Monitor den Engpaß bei der Messung darstellte.

Dazu ist es erforderlich, daß mehrere Lastgeneratoren von einem zentralen Kontrollprozeß (Koordinator) gesteuert werden, der die Lastgeneratoren vor jeder Wiederholung synchronisiert und nach der Meßdatenerfassung die Meßergebnisse von allen beteiligten Meßsystemen einsammelt. Folgender Ablauf wurde daher für den gleichzeitigen Einsatz mehrerer Meßsysteme implementiert:

1. Ein Koordinator wird auf einem unabhängigen Meßsystem installiert und erwartet eine vorgegebene Anzahl an Kontrollverbindungen von Lastgeneratoren.
2. Die Lastgeneratoren bauen vor der Durchführung einer Messung eine zusätzliche Verbindung zu dem Koordinator auf, melden ihre Bereitschaft und warten mit der Durchführung der Messung auf ein Startsignal vom Koordinator.
3. Der Koordinator sendet erst dann ein Startsignal an alle Lastgeneratoren, wenn sich die vorgegebene Anzahl an Clients bei ihm gemeldet hat. Das Startsignal veranlaßt die zeitgleiche Lastgenerierung auf allen Generatoren.
4. Die Lastgeneratoren ermitteln die Meßergebnisse von den Monitoren über die individuellen Kontrollverbindungen. Sind weitere Iterationen durchzuführen, melden sie anschließend wieder ihre Bereitschaft an den Koordinator. Jede Iteration wird somit neu synchronisiert, so daß ein Auseinanderlaufen der Lastgenerierung bei mehreren Wiederholungen verhindert wird.
5. Sind alle Iterationen durchgeführt, berechnen die Lastgeneratoren die Leistungswerte und senden diese Meßergebnisse an den Koordinator. Der Koordinator berechnet das gemeinsame Meßergebnis und protokolliert es für die spätere Auswertung.
6. Sind weitere Meßdaten zu erheben, wird wieder bei Schritt (1) begonnen.

In Abbildung C.2 ist der Ablauf für zwei zu synchronisierende Lastgeneratoren dargestellt. Diese beginnen unabhängig voneinander mit dem Aufbau einer Kontrollverbindung zu ihrem jeweiligen Monitor und teilen dem Monitor die Parameter für die durchzuführende Messung mit. In der Abbildung ist dies durch die Nachrichten "CNTRL(TST-SPEC)" an die Umgebung angedeutet. Danach bauen sie eine Verbindung zum Koordinator auf, der im Zustand "ACCEPT_CLIENTS" den weiteren Meßablauf so lange verzögert, bis die vorgegebene Anzahl an Lastgeneratoren Verbindungen ("SYNC") aufgebaut haben. Das Start-Signal ("START_MEASURE") stößt die Lastgenerierung durch die Generatoren quasi-synchron

(auf wenige Millisekunden genau) an. Die Lastgenerierung führt zu einem hohen Nachrichtenaufkommen zwischen Generator und Monitor, das in der Abbildung weggelassen bzw. in der Prozedur "MEASURE()" gekapselt wurde. Die Monitore senden ihre lokalen Ergebnisse an die Generatoren ("CNTRL (RESULT)"), die ihrerseits die Ergebnisse an den Koordinator senden ("MRESULT (RESULT)"). Der Koordinator sammelt die Einzelergebnisse im Zustand "GET_RESULTS" ein, speichert ein Gesamtergebnis und wechselt wieder in den Zustand "ACCEPT_CLIENTS". Die Lastgeneratoren können sich nun erneut mit dem Koordinator synchronisieren.

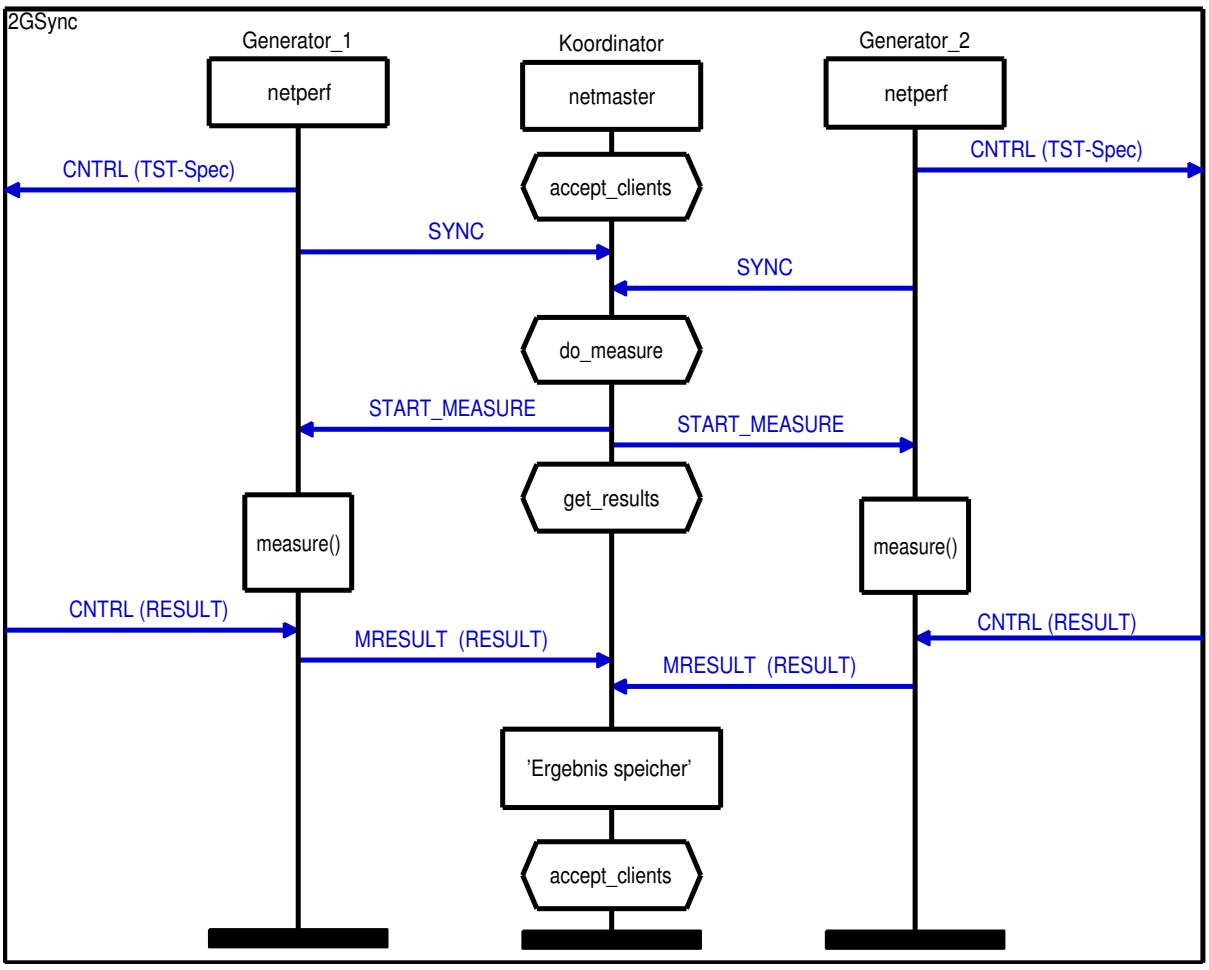


Abbildung C.2: Synchronisation von zwei Lastgeneratoren

C.2.4 Leistungsvorgaben

Das in Abschnitt C.1.4 diskutierte Phänomen der nicht reproduzierbaren Last bei Versuchswiederholungen erforderte weitere Modifikationen an den Meßwerkzeugen.

Aus einer großen Anzahl von Wiederholungen, bei denen die Ausreißer bei den Meßwerten für jeden Meßpunkt (also pro SDU-Länge) manuell aussortiert wurden, konnte ein Erwartungswert der generierten Leistung in Abhängigkeit von der Paketgröße berechnet werden. Die Lastgeneratoren wurden daraufhin folgendermaßen erweitert:

- Zu jedem Meßpunkt wird überprüft, ob die generierte Last bei der ersten Iteration in einem Toleranzbereich von $\pm 1\%$ um den Erwartungswert liegt.
- Ist die generierte Last im Toleranzbereich, wird die Messung mit der konfigurierten Anzahl an Iterationen durchgeführt und die Ergebnisse (Durchsatz mit Vertrauensintervall) werden berechnet und protokolliert.
- Ist die generierte Last der ersten Wiederholung nicht im Toleranzbereich, beendet sich der Lastgenerator. Ein Kontrollprozeß startet den Lastgenerator dann erneut, um die gescheiterte Meßdatenerfassung für den betroffenen Meßpunkt zu wiederholen.
- Um eine Termination zu garantieren, ist die maximale Anzahl an Neustarts des Generators durch den Kontrollprozeß ebenfalls konfigurierbar. Gelingt dennoch keine Lastgenerierung entsprechend den Vorgaben, wird die Meßdatenerfassung für diesen einen Meßpunkt abgebrochen und mit dem folgenden Meßpunkt fortgefahren.

Dieses Verfahren beruht auf der Beobachtung, daß ein einmal gestarteter Lastgenerator ("netperf"-Prozeß) mit sehr hoher Präzision die gleiche Last bei allen Iterationen generiert (kleine Vertrauensintervalle), eine neue Instanz des Lastgenerators (neuer "netperf"-Prozeß) u.U. jedoch eine ganz andere Last generiert, so daß sich die berechneten Vertrauensbereiche nicht überschneiden, wie in Abbildung C.1 zu erkennen ist. Durch einen Neustart des Lastgenerator-Prozesses kann somit versucht werden, *zufällig* die vorgegebene Last zu generieren.

Das Verfahren hat sich bewährt. Es war durch diese Änderungen möglich, die großen Schwankungen bei der generierten Last bei aufeinanderfolgenden Meßreihen so zu verringern, daß sich die berechneten Vertrauensintervalle überschnitten. Abbildung C.3 zeigt die mit diesem Verfahren generierte Last für einen Teil der Durchsatzmessungen, die zu den Werten in Tabelle 4.4 führten. Man erkennt die Überschneidung der Vertrauensintervalle und die geringen absoluten Abweichungen.[2]

C.3 Erweiterungen an den Gerätetreibern

Der Prototyp wurde für Fast Ethernet-Netze implementiert. Im Abschnitt 42 wurde bereits darauf hingewiesen, daß zum Aktivieren des parallelen Selektionsalgorithmus einige

[2] Die Spannbreite beträgt maximal 250 DG/s gegenüber mehr als 1000 DG/s bei Messungen ohne Leistungsvorgaben.

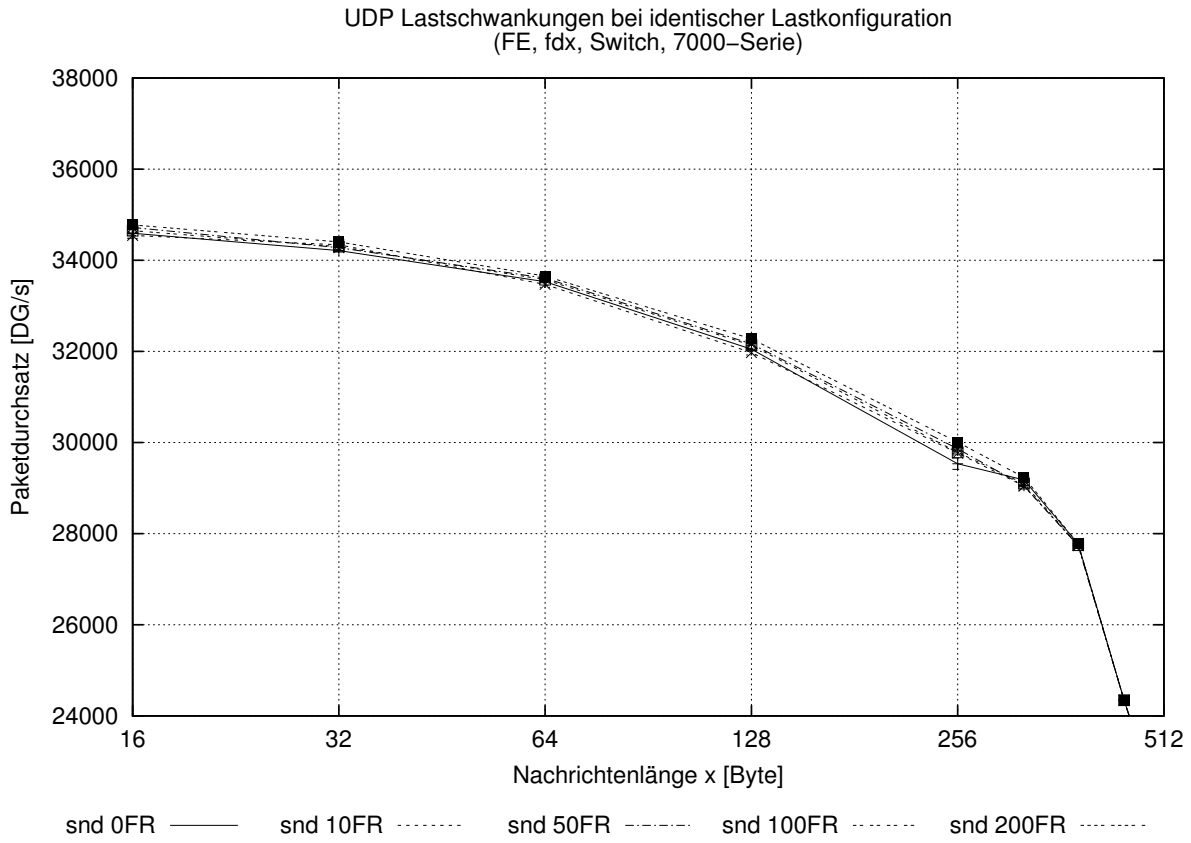


Abbildung C.3: Akzeptable Lastschwankungen bei Versuchswiederholungen

Erweiterungen an den Gerätetreibern erforderlich sind. In diesem Abschnitt sollen kurz alle zusätzlichen Erweiterungen diskutiert werden, die aus funktionalen Gründen und aus Sicherheitsgründen vorgenommen wurden. Die Erweiterungen mußten unter der Randbedingung erfolgen, daß die bereits vorhandenen Funktionen im Gerätetreiber den anderen Anwendungen weiterhin zur Verfügung stehen. Es durften demnach nur kompatible Erweiterungen vorgenommen werden.

C.3.1 Erweiterungen für die verteilte Selektion

Um die in Abschnitt 42 diskutierte, verteilte Selektion zu ermöglichen, mußten folgende Erweiterungen an den Fast Ethernet-Gerätetreibern vorgenommen werden:

- Implementieren zusätzlicher "Input/Output (IO)"-Kommandos zum Ein- und Ausschalten der verteilten Selektion und zum Abfragen bzw. Löschen von Statistiken bezüglich der verteilten Selektion.

- Implementieren des eigentlichen Selektionsalgorithmus in der vorhandenen Empfangsfunktion des Gerätetreibers.

Für die Reaktion auf die zusätzlichen IO-Kommandos war es lediglich erforderlich, die normale Verarbeitungsfunktion für die Bearbeitung von STREAM-IO-Kommandos um die selbstdefinierten Befehle derart zu erweitern, daß hierfür auf eigene Verarbeitungsfunktionen verwiesen wird. Da für die verbesserte parallele Packet Screen die Verteilung der Pakete für die anschließende, dezentrale Selektion über eine "Multicast"-Adressierung erreicht wird, sollte das Setzen dieser "Multicast"-Adresse auch im Zuge der Konfigurierung des Selektionsalgorithmus erfolgen. Der neue Aufruf zum Setzen der Anzahl der parallelen Instanzen und der lokalen Instanz-ID wurde daher um einen Parameter für die zu verwendende "Multicast"-Adresse erweitert.

Der Selektionsalgorithmus wird durch die eben beschriebenen zusätzlichen IO-Kommandos ein- und ausgeschaltet. Zu beachten ist dabei, daß lediglich auf diejenigen Dateneinheiten die parallele Selektion anzuwenden ist, die über die hierfür konfigurierte "Multicast"-Adresse empfangen werden. Somit ist es weiterhin möglich, normale IP-Kommunikationsbeziehungen zum Rechner aufzubauen, die nicht durch die Selektion gestört werden. Dies ist beispielsweise für die dezentrale Administration der parallelen Packet Screen erforderlich.

Der folgende C-Code stellt die Erweiterung für die Selektion im Geräte-Treiber dar. Eine Ethernet-PDU ist zuvor empfangen worden und der Zeiger `ehp` ("Ethernet Header Pointer") zeigt auf den PDU-Kopf. Ob überhaupt eine Selektion durchgeführt werden soll, ist abhängig davon, ob:

- es sich um eine IP-Dateneinheit handelt (Vergleich in Zeile 10),
- der "Multicast"-Empfang und somit die verteilte Selektion aktiviert ist (Zeile 11),
- die verwendete Ethernet-Adresse mit der konfigurierten "Multicast"-Adresse übereinstimmt (Zeile 12),
- ein IP-Modul geladen wurde, an das IP-Dateneinheiten gesendet werden können (Zeile 13, Kurzform für (ipq != NULL)).

Diese Bedingungen werden alle in der "if-Abfrage" in den Zeilen 10–13 überprüft.

Treffen alle diese Bedingungen zu, muß der Selektionsalgorithmus aufgerufen werden. Die "if-Abfrage" in Zeile 22 führt diese Selektion durch. Die Funktion get_ip_chksum () liefert die Prüfsumme des IP-Paketes. Geprüft wird, ob dieser Wert modulo der Anzahl der parallelen Instanzen gleich der lokalen Instanz ID ist. Ist dies der Fall, wird die empfangene Ethernet-PDU um den Ethernet-Kopf gekürzt (Zeile 27) und in die Warteschlange des IP-Moduls eingereiht (Zeile 28). Andernfalls wird die empfangene Ethernet-PDU vernichtet (Zeile 39).

In Abhängigkeit davon, ob eine Dateneinheit lokal verarbeitet wird oder eine andere Instanz diese Dateneinheit verarbeiten muß, werden entsprechende Variablen inkrementiert. Diese

Variablen können mit einem der neuen STREAM-IO-Kommandos (s.o.) abgefragt oder auch zurückgesetzt werden. So läßt sich beispielsweise prüfen, ob der Selektionsalgorithmus zu einer ausgeglichenen Belastung der Instanzen führt. Dies ist der Fall, falls gilt:

$$hme_par_num \approx \frac{hme_par_rx}{hme_par_sendup} \tag{C.1}$$

```
/*
 * cb, 21.10.98
 *
 * Hier koennte man pruefen, ob die ether_dhost die
 * von uns verwendete Multicast-Adresse ist. Wenn ja,
 * dann Selektion ausfuehren und wie unten gezeigt an
 * die IP-Queue weiterleiten.
 */

if ((get_ether_type(ehp) == ETHERTYPE_IP) &&
    (hme_hasmc != 0) &&
    !(bcmp(&ehp->ether_dhost, &hme_mc, 6)) &&
    (ipq))
{
  /* Es ist eine IP-Dateneinheit an die von uns
   * verwendete IP-Multicast-Adresse gesendet worden.
   * Wir fuehren die Filterfunktion aus und senden die IP-
   * Dateneinheit ggf. an das IP-Modul weiter.
   */

  hme_par_rx++;
  if ( ((get_ip_chksum (ehp) % hme_par_num) == hme_par_id)
       && canput(ipq->q_next) )
    {
      /* Um diese PDU muss sich diese Instanz kuemmern */
#ifndef DEBUG
      bp->b_rptr += sizeof (struct ether_header);
      (void) putnext(ipq, bp);
#else
      freemsg (bp);
#endif
      hme_par_sendup++;
      return;
    }
  else
    {
      /* Eine der parallelen Screens wird diese PDU verarbeiten */
      hme_par_dropped++;
      freemsg (bp);
```

```
40          return;
        }
    }
```

Natürlich muß das Betrachtungsintervall hinreichend groß sein, so daß eine repräsentative Menge von IP-Dateneinheiten (beispielsweise 10.000) den Selektionsalgorithmus durchlaufen haben.

C.3.2 Erweiterungen an der Flußkontrolle

Im Abschnitt 8.6.5 wurde darauf hingewiesen, daß die Erkennung eines Angriffs und eine entsprechende Reaktion nach einer der vorgestellten Heuristiken aus Abschnitt 8.5.5.2 erfolgen muß. Beim Prototypen ist in Hinblick auf eine optimale Unterstützung des Packetdurchsatzes der parallelen Packet Screen das "Aggressive Ignorieren aller Nachrichten" im Zustand "ATTACK-DETECTED" implementiert worden.

Zum Erkennen des Endes eines Angriffs müssen die pro $HB_{intervall}$ $[sec]$ eintreffenden Dateneinheiten gezählt werden (vgl. Formel 8.18). Bei einem naiven Ansatz würde man daher im Zustand "ATTACK-DETECTED" alle Dateneinheiten empfangen und sie von der HB-Instanz zählen lassen. Daraus ließe sich erkennen, ob der Angriff fortgesetzt wird oder beendet ist.

Dieser Ansatz hat in der Praxis einen erheblichen Nachteil: Die Dateneinheiten müssen alle bis auf die Anwendungsebene transportiert werden (wiederholter Aufruf der *putnext()*-Funktion), nur um die Anzahl der empfangenen Dateneinheiten festzustellen. Dafür ist nicht nur eine erhebliche Menge an Kernel-Speicher erforderlich, sondern es werden auch die STREAM-basierten Flußkontrollmechanismen des Kernels unnötigerweise, in jedem Modul bis zum STREAM-Kopf für alle Dateneinheiten, beansprucht (auch im Gerätetreiber werden diese Funktionen verwendet, vgl. Zeile 23 und 28 im obigen Listing).

Eine bessere Vorgehensweise ist die Erweiterung der vorhandenen Flußkontrolle. Anstelle unerwünschte Dateneinheiten bis auf die Anwendungsebene zu transportieren, werden diese bereits im Gerätetreiber vernichtet. Das Führen von Statistiken kann ebenfalls dem Gerätetreiber überlassen werden. Da derartige Funktionen nicht im Standardgerätetreiber vorhanden sind, mußten auch hierfür neue IO-Kommandos (zum Ein- und Ausschalten der zusätzlichen Flußkontrolle) implementiert werden. Der implementierte Algorithmus arbeitet folgendermaßen:

1. Prüfe für jede Dateneinheit, ob auf den zugeordneten STREAMS die Flußkontrolle aktiviert ist. Falls nicht, dann sende die Dateneinheit ("upstream") zum nächsten Modul.

2. Falls die Flußkontrolle aktiviert ist, prüfe, ob für die zugeordneten STREAMS noch Kredit vorhanden ist. Falls nicht, vernichte die Dateneinheit und aktualisiere die Statistik.

3. Falls noch Kredit vorhanden ist, verringere den Kredit um eins und sende die Dateneinheit (“upstream”) zum nächsten Modul.

Die Anwendung muß nur noch die Flußkontrolle einschalten und periodisch (in diesem Fall alle $HB_{intervall}$ Sekunden) den Kredit erneuern. Der Gerätetreiber liefert in einer Bestätigung für den Erhalt des neuen Kredits die Anzahl der zwischenzeitlich — aufgrund der Flußkontrolle — vernichteten Dateneinheiten an die Anwendung zurück. Die Implementierung ist unten auszugsweise wiedergegeben.

```
/*
 * While holding a reader lock on the linked list of streams structures,
 * attempt to match the address criteria for each stream
 * and pass up the raw M_DATA ("fastpath") or a DL_UNITDATA_IND.
 */

rw_enter(&hmestruplock, RW_READER);

if ((sbp = (*acceptfunc)(hmestrup, hmep, type, dhostp)) == NULL) {
  hmep->hme_ierrors++;
  rw_exit(&hmestruplock);
  freemsg(mp);
  TRACE_0(TR_FAC_BE, TR_BE_SENDUP_END, "hmesendup end");
  return;
}

/*
 * Loop on matching open streams until (*acceptfunc)() returns NULL.
 */
for (; nsbp = (*acceptfunc)(sbp->sb_nextp, hmep, type, dhostp);
     sbp = nsbp){

  /* Die folgenden acht Zeilen und die weiter unten nach "do the last one"
   * angegebenen Zeilen implementieren die ratenbasierte
   * Flusskontrolle als Ergaenzung zur vorhandenen hmesendup()-Funktion.
   */

  if (sbp->sb_flags & HMESRATE){ /* cb, 12.02.2001: ratecontrol? */
    if ((sbp->rate) == 0){
      sbp->rate_dropped++;
      continue; /* no more rate for this STREAM */
    }else{
      /* decrement allowed rate */
      sbp->rate--;
    }
```

```
    } /* rate control checked */

40  /* normale Funktionalitaet von hmesendup() */

    if (canput(sbp->sb_rq->q_next))
      if (nmp = dupmsg(mp)) {
        if ((sbp->sb_flags & HMESFAST) &&
45      !isgroupaddr) {
      nmp->b_rptr +=
        sizeof (struct ether_header);
      (void) putnext(sbp->sb_rq, nmp);
          } else if (sbp->sb_flags & HMESRAW)
50    (void) putnext(sbp->sb_rq, nmp);
          else if ((nmp = hmeaddudind(hmep, nmp, shostp,
              dhostp, type, isgroupaddr)))
      (void) putnext(sbp->sb_rq, nmp);
      } else
55      hmep->hme_allocbfail++;
    else
      hmep->hme_nocanput++;
  }

60  /*
   * Do the last one.
   */

  /* Flusskontrolle fuer den zuletzt gefundenen STREAM */
65
  if (sbp->sb_flags & HMESRATE) /* cb, 12.02.2001 */
      if (sbp->rate == 0){
        sbp->rate_dropped++;
        freemsg(mp); /* no more rate for this STREAM */
70      goto cblab;
      }else{
        /* decrement allowed rate */
        sbp->rate--;
  }
75
  /* weitere Funktionalitaet von hmesendup() */
```

In Zeile 28 wird geprüft, ob die Flußkontrolle auf dem gefundenen STREAM eingeschaltet ist. Falls nicht, wird die normale Verarbeitung der Funktion `hmesendup()` in Zeile 42 aufgenommen. Ist die Rate bei Null angekommen, wird für die Anwendung der STREAM-lokale Zähler `rate_dropped` erhöht (Zeile 30) und der nächste STREAM, der ebenfalls mit

dem SAP der eingetroffenen Dateneinheit verbunden ist, wird gesucht (Zeile 31 verzweigt zu Zeile 20).

Falls die Rate für den STREAM noch nicht abgelaufen ist, wird der Zähler für diese Rate in Zeile 34 verringert und die normale Bearbeitung ab Zeile 42 fortgesetzt. Man beachte, daß diese Überprüfungen stattfinden, bevor die Nachrichten an darauffolgende Module weitergegeben werden (Zeilen 48, 50, 53).

Falls keine Rate mehr vorhanden ist, und kein weiterer STREAM mit demselben SAP verknüpft ist, wird die Dateneinheit bereits unmittelbar nach dem Empfang im Gerätetreiber vernichtet (Zeile 69). Durch diesen Mechanismus wird erreicht, daß Dateneinheiten für eine Anwendung, die die Flußkontrolle aktiviert hat, genauso effizient abgewiesen werden können, als würde der Rechner eine Dateneinheit empfangen, für die es überhaupt keinen SAP gibt (Zeilen 9–15).

Durch diese Erweiterung wird das "Aggressive Ignorieren aller Nachrichten" optimal unterstützt. Im Zustand "ATTACK-DETECTED" kann die Instanz nun folgendermaßen vorgehen:

1. Aktiviere die Flußkontrolle.
2. Setze periodisch alle $HB_{intervall}$-Sekunden die Rate auf 0.
3. Prüfe nach Formel 8.18, ob der Rückgabewert (`rate_dropped`) des Funktionsaufrufs zum Setzen der Rate immer noch eine zu hohe Anzahl an PDUs anzeigt. Falls ja, gehe zu Schritt (2). Falls nein, deaktiviere die Flußkontrolle und wechsle in den Zustand "WAIT-BEAT"; der Angriff ist vorüber.

Durch diesen Algorithmus ist die Belastung durch das Gruppenmanagement auf einer Instanz, die unter dem in Abschnitt 8.5.5 dargestellten DoS-Angriff leidet, *geringer als im normalen Betriebszustand "*WAIT-BEAT*"*. Damit entspricht das Verhalten sowohl der Intuition als auch den Anforderungen an überlebensfähige Systeme, daß im Zuge einer "degradation" – aufgrund eines Angriffs – Betriebsmittel (hier CPU-Zeit und Speicher) frei werden. Ohne diese zusätzlichen Mechanismen für eine effiziente Flußkontrolle wäre das Ziel nicht erreichbar.

C.3.3 Erweiterungen an der Fehlererkennung

Die für den normalen, hardwarenahen Zugriff auf die Gerätetreiber verwendete DLPI-Schnittstelle (vgl. [228]) bietet zwar die Möglichkeit, Statusinformationen über die jeweiligen Netzschnittstellen abzurufen, jedoch fehlen hierbei unverständlicherweise wichtige Informationen, nämlich die Informationen zum Status des physikalischen Netzanschlusses.

Fast alle bekannten Schicht-2-Protokolle ("Data Link Layer") verfügen über Anschluß-Informationen, die Auskunft über die physikalische Verbindung der Schnittstelle zum jeweiligen

Netz geben. Diese Informationen sind somit vorhanden und liegen in der Regel auch als Informationen im Kernel vor (als irgendein Bit eines Statusvektors für jede Netzschnittstelle), werden jedoch nicht als Teil der DLPI-Informationen an den Benutzer geliefert.

Diese Informationen sind jedoch auch für die Fehlererkennung des GMPs beim Einsatz auf der parallelen Packet Screen wichtig. Tritt beispielsweise ein Kabelbruch zwischen der Instanz und dem externen Netz auf, während das GMP jedoch auf der Schnittstelle zum internen Netz betrieben wird, dann würden alle GMP-Instanzen davon ausgehen, daß die vom Kabelbruch betroffene Packet Screen Instanz noch verfügbar ist. Es würde keine Rekonfiguration stattfinden. Die lokalen Instanzen des GMPs müssen daher Zugriff auf die Anschluß-Informationen aller Schnittstellen haben, von denen der verteilte Dienst abhängt. Erkennt eine GMP-Instanz ein Problem mit einer der essentiellen Schnittstellen, dann kann sie in den Zustand "Backoff" wechseln, so daß die anderen Instanzen sie aus der Gruppe ausschließen.

Da diese Informationen nicht als Teil der `DL_INFO_ACK`-Nachricht geliefert werden, mußte ein weiteres IO-Kommando implementiert werden, um diese Informationen abzurufen. Da nur geprüft werden muß, ob eine physikalische Verbindung zum Netz vorliegt oder nicht, reicht hierfür ein Bit pro Schnittstelle aus. Der folgende Auszug aus der Funktion `hme_handle_fwl()`, die mehrere der zusätzlichen IO-Kommandos bearbeitet, zeigt die Implementierung:

```
/* Der folgende Auszug ist Teil der Funktion hme_handle_fwl(),
 * die einen Grossteil der zusaetzlichen IO-Kommandos implementiert.
 * Die auszufuehrende Aktion wird mit dem Parameter "get" uebergeben,
 * der anzeigt, ob Informationen abgefragt oder gesetzt werden sollen.
 */

switch (get){
 case 1: /* get infos */
   {
     /* Benutzer will Infos lesen, also keine
      * Ueberpruefung notwendig.
      */
     bcopy (&hme_mc, &hme_mc_fwl->mcaddr, 6);
     hme_mc_fwl->hme_par_num = hme_par_num;
     hme_mc_fwl->hme_par_id = hme_par_id;
     hme_mc_fwl->hme_par_rx = hme_par_rx;
     hme_mc_fwl->hme_par_sendup = hme_par_sendup;
     hme_mc_fwl->hme_par_dropped = hme_par_dropped;
     hme_mc_fwl->hme_par_tx = hme_par_tx;
     hme_mc_fwl->hme_enablemcarp = hme_hasmc;
     /* cb, 12.07.2000 :
      * pruefen, welche Instanz physikalisch angeschlossen ist,
      * d.h. ob der physikalische Link "up/down" ist.
      */
     hme_par_linkup = 0;
```

```
        hmep = hmeup;
        while (hmep){
          hmeifmask = HMEUP0 ■ (ddi_get_instance(hmep->hme_dip));
          mutex_enter(&hmelock);
30        (void) hme_mii_read(hmep, HME_PHY_BMSR, &stat);
          mutex_exit(&hmelock);
          if (stat & PHY_BMSR_LNKSTS){
            hme_par_linkup |= hmeifmask;
          }
35        /* naechstes IF auswaehlen */
          hmep = hmep->hme_nextp;
        }
        hme_mc_fwl->hme_par_linkup = hme_par_linkup;
        miocack (wq, mp, HME_MC_FWL_T_SIZE, 0);
40      return;
      }
```

In dem Block von Zeile 13–20 werden die Statistiken und die aktuellen Konfigurationsparameter (Anzahl paralleler Instanzen, Gesamtzahl der Instanzen) des verteilten Selektionsalgorithmus ausgelesen. Die Variable `hme_par_linkup` (Zeile 25) speichert das Bitfeld mit den Informationen, ob die jeweiligen Schnittstellen mit dem Netz verbunden sind. Dazu wird in der Schleife von Zeile 27–37 die verkettete Liste mit Informationen bezüglich der Netzschnittstellen des Typs "hme" (Fast Ethernet-Schnittstelle) durchlaufen und das entsprechende Status-Bit abgerufen (Zeile 30, `HME_PHY_BMSR`). Die Funktion `hme_mii_read()` liest dazu die Register des jeweiligen Ethernet-Chipsatzes aus. Zu beachten ist noch, daß unabhängig davon, an welcher Position die Schnittstellen-Instanz in der verketteten Liste eingehängt ist, das entsprechende Bit immer an derselben Stelle im Bitvektor gesetzt wird. Dies wird durch die Abfrage des "Physical Point of Attachment (PPA)", der die Schnittstellen-Instanz auch nach einem Rechnerneustart eindeutig identifiziert[3], erreicht (Zeile 28). Ist die GMP-Software einmal konfiguriert, kann sie sich somit darauf verlassen, daß die bei der Konfiguration verwendeten PPA-Bezeichner auch immer dieselbe Schnittstelle identifizieren. Nur so kann sichergestellt werden, daß die oben diskutierte Überwachung der Hardware durch die GMP-Instanz auch tatsächlich die für den Dienst essentiellen[4] Schnittstellen überwacht. Ist die Schnittstelle an das Netz angeschlossen, wird schließlich in Zeile 33 das dem PPA entsprechende Bit im Bitvektor gesetzt. Dieser Vek-

[3] Dies ist eine sehr wichtige Eigenschaft. Ist ein Rechner beispielsweise mit vier Netzkarten des selben Typs bestückt (hme0,...,hme3), dann sollte die GMP-Instanz nach einer einmaligen Konfigurierung (beispielsweise für die Schnittstelle hme2) immer diesen PPA (PPA = 2) vorfinden können, auch dann, wenn eine weitere Schnittstelle eingebaut oder eine bei der Konfigurierung vorhandene Schnittstelle ausgebaut wird.

[4] Dies ist eine typische "depends on"-Relation. Der "Packet Screen"-Dienst auf jeder Instanz hängt davon ab, daß die Schnittstellen zu den Netzen, zwischen denen gefiltert wird, funktionieren. Alle anderen Netzschnittstellen im Rechner sind für diesen Dienst nicht relevant.

tor wird als Teil der eigens für die neuen IO-Kommandos definierten Informationsstruktur (`HME_MC_FWL_T`) an die Anwendung gesendet (Zeile 38, 39).

C.4 Zusammenfassung

Der Aufwand für eine ordentliche Meßdatenerfassung in verteilten Systemen ist sehr hoch. Es müssen zum Teil sehr systemspezifische Konfigurationsschritte unternommen werden, um systematische Effekte auszuschließen, die zu Verfälschungen der Ergebnisse führen können.

Die bekannten Werkzeuge zur Meßdatenerfassung haben teilweise erhebliche funktionale Defizite. Ohne die Verfügbarkeit der Quelltexte wäre daher eine Anpassung an individuelle Hardware und an individuelle Meßexperimente nicht möglich gewesen. Die Verfügbarkeit der Quelltexte ist somit nicht nur für die Erstellung eines überlebensfähigen Systems unbedingt erforderlich, sondern auch für die begleitende oder abschließende Leistungsevaluierung.

Für den Prototypen einer überlebensfähigen Packet Screen waren auch einige Modifikationen im "Kernel" derjenigen Rechner erforderlich, auf die Packet Screen Instanzen plaziert wurden. Es wurde ein effizienter Selektionsalgorithmus im Gerätetreiber implementiert, um möglichst wenig Aufwand für nicht-selektierte Dateneinheiten zu betreiben. Außerdem wurde die Erkennung von Problemen bzgl. des Netzanschlusses im Gerätetreiber implementiert, um die Fehlererkennung des Gruppenmanagement-Protokolls zu unterstützen. Abgerundet werden die Erweiterungen an den Fast Ethernet-Gerätetreibern durch eine sehr effiziente Flußkontrolle, die es ermöglicht, DoS-Angriffe auf das GMP aufwandsarm zu überstehen.

Für die vergleichende Bewertung der Packet Screen – in ihren verschiedenen Entwicklungsphasen bis hin zur überlebensfähigen Packet Screen – war es erforderlich, mehrere Meßwerkzeuge automatisch zu synchronisieren, um eine ausreichend hohe Last generieren zu können. Die Trennung von Kontroll- und Lastkommunikationsbeziehung zwischen Lastgenerator und Monitor, sowie die Möglichkeit zur Vorgabe und automatischen Kontrolle von Lastkenngrößen bei der Durchführung von Messungen, waren notwendige Erweiterungen, um mit den vorhandenen Meßwerkzeugen alle erforderlichen Messungen durchführen zu können.

Abkürzungsverzeichnis

ATM:	Asynchronous Transfer Mode
AAL:	ATM Adaption Layer
AIX:	UNIX-Derivat der Firma IBM
ARP:	Address Resolution Protocol
ASIC:	Application Specific Integrated Circuit
ATM:	Asynchronous Transfer Mode
BAN:	Burrows, Abadi, Needham
BNF:	Backus-Naur-Form
BSD:	Berkeley Software Distribution
CERT:	Computer Emergency Response Team
COTS:	Commercial-of-the-Shelf
CPU:	Central Processing Unit
CSMA/CD:	Carrier Sense Multiple Access with Collision Detection
DG:	Datagramm
DIN:	Deutsche Industrie Norm
DLPI:	Data Link Provider Interface
DMA:	Direct Memory Access
DMZ:	Demilitarized Zone
DNS:	Domain Name System
DoS:	Denial of Service
EA:	Empfängeradresse
EDV:	Elektronische Datenverarbeitung
ELF:	Executable and Linking Format
EP:	Empfangsportnummer
FDDI:	Fiber Distributed Data Interface
FE:	Fast Ethernet
FR:	Forwarding Rate
FTP:	File Transfer Protocol
GMP:	Gruppenmanagement-Protokoll
GM:	Gruppenmanagement
GNY:	Gong, Needham, Yahalom
HB:	Heart Beat
HSN:	High Speed Network
HTML:	HyperText Markup Language
HTTP:	HyperText Transfer Protocol
IBM:	International Business Machines

ICMP: Internet Control Message Protocol

IDS: Intrusion Detection System

IEEE: Institute of Electrical and Electronics Engineers

IETF: Internet Engineering Task Force

IFG: Inter-Frame Gap

IF: Interface

IO: Input/Output

IPX: Internetwork Packet Exchange

IP: Internet Protocol

ISDN: Integrated Services Digital Network

ISO: International Organization for Standardization

KByte: Kilobyte

LAF: Log Availability Filter

LAN: Local Area Network

LINUX: UNIX-Derivat

MAC: Message Authentication Code **oder** Media Access Control

Mbit/s: Megabit pro Sekunde

MD: Message Digest

MFR: Maximum Frame Rate

MIMD: Multiple Instruction Streams, Multiple Data Streams

MSC: Message Sequence Chart

MTU: Maximum Transfer Unit

OSIRM: Open Systems Interconnection Reference Model

OSI: Open Systems Interconnection

OSPF: Open Shortest Path First

OS: Operating System

PCI: Peripheral Component Interconnection **oder** Protocol Control Information

PDU: Protocol Data Unit

PID: Processor Identification **oder** Process Identification

POSIX: Portable Operating System Interface

PPA: Physical Point of Attachment

PPS: Packets per Second

PS: Packet Screen

RFC: Request for Comments

RIP: Route Information Protocol

RPC: Remote Procedure Call

SAP: Service Access Point

SA: Senderadresse

SDL: Specification and Description Language

SDU: Service Data Unit

SIGNAL: Typbezeichnung für Nachrichten bei SDL

SPEC: Standard Performance Evaluation Corporation

SPX: Sequenced Packet Exchange

SP:	Sendeportnummer
TCP:	Transmission Control Protocol
TIMER:	Zeitgeber
TP:	Trace Point
UDP:	User Datagram Protocol
UNIX:	Klasse von Betriebssystemen
URL:	Universal Resource Locator
UTP:	Unshielded Twisted Pair
VPN:	Virtual Private Network
WAN:	Wide Area Network
WWW:	World Wide Web

Literaturverzeichnis

[1] Abadi, M.; Lamport, L. “Composing Specifications”, SRC Research Report 66, Digital Systems Research Center (SRC), October 1990 2.3, 8.1, 1

[2] Agouros, K. “Aktive Webinhalte und Schutzmöglichkeiten”, Tagungsunterlagen 6. Workshop des “DFN-CERT, Zentrum für sichere Netzdienste GmbH” und des Projektes “PCA im DFN” “Sicherheit in vernetzten Systemen”, 17./18. März 1999, Hamburg, in DFN-Bericht Nr. 87 10.3.2

[3] Aguilera, M. K.; Chen, W.; Toueg, S. “A Timeout-Free Failure Detector for Quiescent Reliable Communication”, Proc. 11th Intern. Workshop on Distributed Algorithms, WDAG’97, Saarbrücken, Germany, September 1997 6.5, 6.6

[4] Aguilera, M. K.; Chen, W.; Toueg, S. “Failure Detection and Consensus in the Crash-Recovery Model”, Proc. 12th Intern. Symp. on Distributed Computing, DISC’98, pp. 231–245, Andros, Greece, September 1998 7.1, 7.2.3

[5] Aguilera, M. K.; Chen, W.; Toueg, S. “Using the heartbeat failure detector for quiescent reliable communication and consensus in patitionalble networks”, Theoretical Computer Science 220, pp. 3–30, June 1990 6.5, 6.6, 7.1, 7.2.3

[6] Aho, A. V.; Sethi, R.; Ullman, J. D. “Compilerbau”, Teil 1, Addison-Wesley GmbH, 1988 5.2.4

[7] Alexander, D. S.; Arbaugh, W. A.; Keromytis, A. D.; Muir, S.; Smith, J. M. “Secure Quality of Service Handling (SQoSH)”, IEEE Communications Magazine, vol. 38, no. 4, pp. 106–112, Apr. 2000 87

[8] Anceaume, E. “A Lightweight Solution to Uniform Atomic Broadcast for Asynchronous Systems”, Proc. 27th Annual Intern. Symp. on Fault-Tolerant Computing, FTCS-27, pp. 292–301, Seattle, Washington, USA, June 1997 7.1

[9] Anderson, R. J. “Why Cryptosystems Fail”, Communications of the ACM, Vol. 37, No. 11, pp. 32–40, November 1994 100

[10] Apple Developer Connection "Understanding PCI Performance", Technical Note No. 1008 (TN1008), Apple Computer, Inc., October 1995 (`http://www.devworld.apple.com/technotes/tn/tn1008.html`) 4.1

[11] Arona, A.; Bruschi, D.; Rosti, E. "Adding Availability to Log Services of Untrusted Machines", in Proc. "15th Annual Computer Security Applications Conference" (ACSAC'99), pp. 199–206, Phoenix, Arizona, USA, December 1999 10.2.4

[12] Atkinson, R. "IP Encapsulating Security Payload (ESP)", Request for Comments: 1827, Network Working Group, August 1995 8.5.5, 10.2.2

[13] Aura, T.; Nikander, P. "Stateless Connections", Proc. Information and Communications Security, First Intern. Conf., ICICS'97, pp. 87–97, November 1997, Beijing, China 87

[14] Awerbuch, B.; Ostrovsky, R. "Memory-Efficient and Self-Stabilizing Network RESET", Proc. 13th Annual Symp. on Principles of Distributed Computing, pp. 254–263, Los Angeles, CA, USA, August 1994

[15] Awerbuch, B.; Patt-Shamir, B.; Varghese, G. "Self-Stabilization By Local Checking and Correction", Proc. 32nd Annual Symp. on Foundations of Computer Science, pp. 268–277, San Juan, Puerto Rico, October 1991 6.5

[16] Awerbuch, B.; Patt-Shamir, B.; Varghese, G.; Dolev, S. "Self-Stabilization by Local Checking and Global Reset", Proc. 8th Intern. Workshop on Distributed Algorithms, WDAG'94, pp. 326–339, Terschelling, The Netherlands, September/October 1994 6.5

[17] Awerbuch, B.; Varghese, G. "Distributed Program Checking: a Pradigm for Building Self-stabilizing Distributed Protocols", Proc. 32nd Annual Symp. on Foundations of Computer Science, pp. 258–267, San Juan, Puerto Rico, October 1991 6.5

[18] Badach, A.; Hoffmann, E.; Knauer, O. "High Speed Internetworking. Grundlagen und Konzepte für FDDI und ATM", Addison-Wesley 1994 2.1.1

[19] Barlett, W. (Moderator) "Cyber Threats", panel discussion, Proc. 28th Annual Intern. Symp. on Fault-Tolerant Computing, FTCS-28, pp. 104–106, Munich, Germany, June 1998 2.2

[20] Beauquier, J.; Gradinariu, M.; Johnen, C. "Memory Space Requirements for Self-stabilizing Leader election Protocols", Proc. 18th Annual ACM Symp. on Principles of Distributed Computing, pp. 199–207, Atlanta, GA, USA, May 1999 7.1

[21] Benecke, C. "A Parallel Packet Screen for High Speed Networks", in Proc. "15th Annual Computer Security Applications Conference" (ACSAC'99), pp. 67–74, Phoenix, Arizona, USA, December 1999 33, 4.5.1

[22] Benecke, C. “Entwurf und Realisierung eines parallelen Analysewerkzeuges für Ethernet-basierte Netze”, Diplomarbeit am Fachbereich Informatik, Universität Hamburg, Februar 1996 19, 4.2.1.2, 4.2.2, 4.3.1, 40, 10.2.3

[23] Benecke, C. “Sicherheit in ATM Netzen”, Online Ratgeber des DFN-FWL, Universität Hamburg, Fachbereich Informatik, April 2001 (`http://www.fwl.dfn.de/fwl/ratgeber/ratgeber.html`) 2.1.2, 88, 8.5

[24] Benecke, C.; Ellermann, U. “Nutzung von Kryptographie in Zusammenhang mit Firewalls”’. *Studie des DFN-FWL*, DFN-Bericht Nr. 86. Oktober 1998 (`http://www.fwl.dfn.de/dfn/berichte/db086/`) 2.3.1.1, 2.8, 2.3.1.1, 12, 13, 15, 3, 87, 88, 8.5.1

[25] Benecke, C.; Ellermann, U. “Securing ‘Classical IP over ATM Networks’”, Proc. of 7th USENIX Security Symp., 26-29. Januar 1998, San Antonio, Texas, Januar 1998 2.1.2

[26] Benecke, C.; Ellermann, U. “Tools für Durchsatzmessungen an Proxies: Entwurf und Realisierung am Beispiel von HTTP-Client und -Server”, MMB-Arbeitsgespräch: Leistungs-, Zuverlässigkeits- und Verläßlichkeitsbewertung von Kommunikationsnetzen und verteilten Systemen, Hamburg, 5./6. März 1998 17, 3.2.2.1, C

[27] Berners-Lee, T.; Fielding, R.; Frystyk, H. “Hypertext Transfer Protocol – HTTP/1.0”, Network Working Group, Request for Comments: 1945, May 1996 17

[28] Birman, K. P.; Joseph, T. A. “Reliable Communication in the Presence of Failures”, ACM Transactions on Computer Systems, vol. 5, no. 1, pp. 47–76, February 1987 6.5

[29] Black, U. D. “Data link protocols”, Prentice-Hall, New Jersey, 1993 3.10

[30] Bloomer, J. “Power Programming with RPC”, O’Reilly 1992 5.2.1

[31] Bodei, C.; Degano, P. “Security Analysis using Flow Logics”, Bulletin of the European Association for Theoretical Computer Science, no. 70, pp. 112–129, Feb. 2000 8

[32] Boneh, D.; Durfee, G.; Franklin, M. “Lower Bounds for Multicast Message Authentication”, in Pfitzmann, B. (Ed.) EUROCRYPT 2001, LNCS 2045, pp. 437–452, Springer-Verlag 2001 8.5.1

[33] Boyd, C.; Mao, W. “On a Limitation of BAN Logic”, in Proc. “Advances in Cryptology–EUROCRYPT’93”, Workshop on the Theory and Application of Cryptographic Techniques, Lofthus, Norway, May 1993 81

[34] Brackin, S. H. “Evaluating and Improving Protocol Analysis by Automatic Proof”, Proc. of Computer Security Foundations Workshop XI, Rockport, Massachusetts, USA, June 1998

[35] Brackin, S. H. “Using Checkable Types in Automatic Protocol Analysis”, in Proc. “15th Annual Computer Security Applications Conference (ACSAC99)”, Phoenix, Arizona, USA, Dezember 1999 8.4.3

[36] Bradner, S. (Ed.) “Benchmarking Terminology for Network Interconnection Devices”, Request for Comments: 1242, Network Working Group, July 1991 3

[37] Bradner, S.; McQuaid, J. “Benchmarking Methodology for Network Interconnect Devices”, Request for Comments: 2544, Network Working Group, March 1999 3.2, 3, 3.2.2.2

[38] Brauch, P. “FBI warnt vor Schutzgeld-Erpressung im Internet”, Heise Online News, März 2001, (`http://www.heise.de/newsticker/data/pab-09.03.01-000/`) 1

[39] Brusil, P. J.; Johnson, L. A.; Steeble, E. F. “Emerging Security Testing, Evaluation, and Validation. The Key to Enhancing Consumer Trust in Security-Enhanced Products”, in [184], pp. 347–381

[40] Burns, A.; McDermind, J.A.; Dobson, J.E. “On the Meaning of Safety and Security”, Technical Report Series, No. 382, University of Newcastle upon Tyne, May 1992 6

[41] Burrows, M.; Abadi, M.; Needham, R. “A Logic of Authentication”, SRC Research Report 39, Digital Systems Research Center (SRC), February 1989 81, 8.4.3, 87, 92, A.2.1

[42] CERT Coordination Center “Ongoing Network Monitoring Attacks”, CERT Advisory CA-94:01, 3. Februar 1994
(`ftp://ftp.cert.dfn.de/pub/csir/cert/cert_advisories/CA-94:01.network.monitoring.attacks`) 78

[43] CERT Coordination Center “IP Spoofing Attacks and Hijacked Terminal Connections”, CERT Advisory CA-95:01, 23. Januar 1995,
(`ftp://ftp.cert.dfn.de/pub/csir/cert/cert_advisories/CA-95:01.IP.spoofing`) 78

[44] Chandra, T. D.; Toueg, S. “Unreliable Failure Detectors for Asynchronous Systems”, Proc. 10th Annual ACM Symp. on Principles of Distributed Computing, pp. 325–340, Montreal, Quebec, Canada, August 1991 6.5, 6.6, 7.2.3

[45] Chang, J.-M.; Maxemchuk, N. F. “Reliable Broadcast Protocols”, ACM Transactions on Computer Systems, vol. 2, no. 3, pp. 251–273, August 1984 7.1

[46] Chapman, D. B. & Zwicky, E. D. “Building Internet Firewalls”, O’Reilly 195 2.3.1.1, 2.3.1.1

[47] Cheswick, W. R.; Bellovin, S. M. “Firewalls and Internet Security: repelling the wily hacker”, Addison-Wesley, 1994 2.3.1.1, 2.3.1.1

[48] Cheung, S.; Levitt, K. N. “Protecting Routing Infrastructures from Denial of Service Using Cooperative Intrusion Detection”, Proc. New Security Paradigms Workshop 1997, pp. 94–106, Langdale, Cumbria, UK, in ACM SIGSAC, vol. 15, no. 3 8.4.4

[49] Clark, J. A.; Jacob, J. L. “Searching for a Solution: Engineering Tradeoffs and the Evolution of Provably Secure Protocols”, in Proc. “IEEE Sympos. on Security and Privacy”, pp. 82–95, Oakland, CA, May 2000 83

[50] Clegg, M.; Marzullo, K. “A Low-Cost Processor Group Membership Protocol for a Hard Real-Time Distributed System”, Proc. Real-Time Systems Symp., pp. 90–98, San Francisco, CA, USA 6.5

[51] Comer, D. E. “Internetworking with TCP/IP Volume I. Principles, Protocols, and Architecture”, 2nd Edition, Prentice-Hall 1991 7.2.5.1

[52] Comer, D. E. & Stevens, D. L. “Internetworking with TCP/IP Volume III. Client-Server Programming and Applications”, Prentice-Hall 1992 5.2.1

[53] Coulouris, G.; Dollimore, J.; Kindberg, T. “Distributed Systems. Concepts and Design”, 2nd Edition, 3rd impression, Addison-Wesley, 1995 5.2

[54] Cristian, F. “Abstractions for Fault-Tolerance”, Invited Paper, in Duncan, K; Krueger, K. (eds.) “Linkage and Developing Countries – Information Processing ‘94”, Proc. of the IFIP 13th World Computer Congress, Hamburg, Germany, 28 August - 2 September, Elsevier Science B. V., 1994 6.4, 6.5

[55] Cristian, F. “On the Semantics of Group Communication”, Proc. Formal Techniques in Real-Time and Fault-Tolerant Systems, pp. 1–21, 4th Intern. Symp., Uppsala, Sweden, September 1996 6.3.2

[56] Cristian, F. “Reaching Agreement on Processor Group Membership in Synchronous Distributed Systems”, Research Report RJ 5964 (59426), IBM Reseach Division (Ed.), March 1988 6.5

[57] Cristian, F. “Synchronous Atomic Broadcast for Redundant Broadcast Channels”, Research Report RJ 7203 (67682), IBM Reseach Division (Ed.), December 1989 6.5, 7.1

[58] Cristian, F. “Understanding Fault-Tolerant Distributed Systems”, Communications of the ACM, vol. 34, no. 2, pp. 56–78, February 1991 6.2, 63, 6.4, 7.2.3.2

[59] Cristian, F.; Fetzer, C. “The Timed Asynchronous Distributed System Model”, IEEE Transactions on Parallel and Distributed Systems, vol. 10, no. 6, pp. 642-657, June 1999 6.6, 68, 7.2.5.2, 7.2.5.2, 75

[60] DFN-CERT (Hrsg.) “Leitfaden zur Absicherung von Rechnersystemen in Netzen”, Version 1.0, Zentrum für sichere Netzdienste GmbH, Juli 2000 (`ftp://ftp.cert.dfn.de/pub/docs/leitfaden/leitfaden.ps.gz`) 13, 106

[61] DIN Deutsches Institut für Normen e.V. (Hrsg.) “Informationsverarbeitung 1. Normen über Grundbegriffe, Datenübertragung, Schnittstellen”, DIN Taschenbuch 25, 5. Auflage 1981 61

[62] Deitel, H. M. “Operating Systems”. Second Edition, Addison-Wesley, 1990 5

[63] Deitel, H. M. “An Introduction to Operating Systems”. Revised First Edition, Addison-Wesley, 1984 5

[64] Delgado Friedrichs, F. “Cracker-Tools am Beispiel”, in [176], S. 165–186 2.2.2, 106

[65] Dierks, T.; Allen, C. “The TLS Protocol. Version 1.0”, Request for Comments: 2246, Network Working Group, January 1999 8.5.5

[66] Dijkstra, E. W. “Self-stabilizing Systems in Spite of Distributed Control”, Communications of the ACM, vol. 17, no. 11, November 1974 7.1

[67] Dolev, D.; Dwork, C.; Stockmeyer, L. “On the Minimal Synchronisation Needed for Distributed Consensus”, Journal of the ACM, vol. 34, no. 1, pp. 77-79, January 1987 7

[68] Dolev, D.; Yao, A. C. “On the Security of Public Key Protocols”, IEEE Transactions on Information Theory, Vol. 29, No. 2, March 1983 78, 8.4.2, 86

[69] Eastlake, D. “Domain Name System Security Extensions”, Request for Comments: 2535, Network Working Group, March 1999 10.3.1

[70] Ehlebracht, O. “Management von QoS am Beispiel der Integration von SQoS in das Kommunikationssystem”, Studienarbeit, Universität Hamburg, Fachbereich Informatik, März 2001 87

[71] Ellermann, U. “Firewalls in Hochgeschwindigkeitsnetzen. Untersuchungen zu Firewalls in ATM-Netzen”, Dissertation am Fachbereich Informatik der Universtiät Hamburg, Hamburg 1998 2.3.1.1, 15, 3, 33, 4.3.3, 78, 10.2.1

[72] Ellermann, U. “Netzabsicherung durch Firewalls”, 4. IT-Sicherheitskongreß, 8. bis 11. Mai, Bonn 1995 (`http://www.cert.dfn.de/team/ue/fw/fire/`) 2.3.1.1

[73] Ellermann, U. “Ein Firewall am Fachbereich Informatik”, Universität Hamburg, Dezember 1993 2.3.1.3

[74] Ellermann, U. “Firewalls. Isolations- und Audittechniken zum Schutz von lokalen Computer-Netzen”, DFN-Bericht Nr. 76, September 1994 2.3.1.1

[75] Ellermann, U. “IPv6 and Firewalls”, Proc. of SECURICOM — 14th Intern. Congress on Computer and Communications Security Protection, Paris, Juni 1996 2.3.1.1

[76] Ellermann, U.; Benecke, C. “Firewalls für Hochgeschwindigkeitsnetze”, Deutscher Internet Kongress 1998, 5. und 6. Mai 1998, CongressCenter Messe Frankfurt, Mai 1998 2.3.1.1, 15, 3, 33, 4.3.3, 10.2.1

[77] Ellermann, U.; Benecke, C. “Firewalls for ATM Networks”, in Proc. INFOSEC‘COM, Paris, 4./5. June, 1998 3, 4.3, 10.2.1

[78] Ellermann, U.; Benecke, C. “Firewalls in ATM-Netzen: Vorbericht”, DFN-FWL Forschungsbericht, Hamburg, Februar 1998 3, 25, 4.3.3, 10.2.1, C

[79] Ellision, R. J.; et al. “A Case Study in Survivable Network System Analysis”, Technical Report, CMU/SEI-98-TR-014, ESC-TR-98-014, Software Engineering Institute, Carnegie Mellon University, Pittsburgh 1998 5, 2.2.4, 2.3

[80] Ellision, R. J.; et al. “Survivable Network Systems: An Emerging Discipline”, Technical Report, CMU/SEI-97-TR-013, ESC-TR-97-013, Software Engineering Institute, Carnegie Mellon University, Pittsburgh 1997 2.1, 2.2.2, 2.2.3

[81] Ellsberger, J.; Hogrefe, D.; Sarma, A. “SDL. Formal Object-oriented Language for Communicating Systems”, Prentice Hall Europe 1997 7.2, A, 117

[82] European Computer Manufacturers Association (ECMA) “Security in Open Systems. Data Elements and Service Definitions”, STANDARD ECMA-138, December 1989 2.4

[83] European Computer Manufacturers Association (ECMA) “Security in Open Systems. A Security Framework”, ECMA TR/46, July 1988 2.4

[84] Faltin, N. et al. “An Annotational Extension of Message Sequence Charts to Support Performance Engineering”, in Cavalli, A.; Sarma, A. (eds.) SDL’97: Time for Testing – SDL, MSC and Trends, pp. 307–322, Evry, France, September 1997 109

[85] Fetzer, C.; Cristian, F. “A Fail-Aware Membership Service”, Proc. 16th Symp. on Reliable Distributed Systems, pp. 157–164, Durham, North Carolina, USA, October 1997 68, 76

[86] Fetzer, C.; Cristian, F. “Derivation of Fail-Aware Membership Service Specifications”, Proc. Parallel and Distributed Processing, in Conjunction with the 12th Intern. Parallel Processing Symp. and 9th Symp. on Parallel and Distributed Processing, IPPS/SPDD’98, pp. 644–663, Orlando, FL, USA, March/April 1998 68

[87] Fielding, R. et al. “Hypertext Transfer Protocol – HTTP/1.1”, Network Working Group, Request for Comments: 2068, January 1997 17

[88] Fischer, M. J.; Lynch, N. A.; Paterson, M. S. “Impossibility of Distributed Consensus with One Faulty Process”, Journal of the ACM, vol. 32, no. 2, pp. 374–382, April 1985 6.5, 6.6, 7, 7.1, 7.2.3.2

[89] FitzGerald, N. “What DDoS it all Mean?”, Virus Bulletin, vol. 12, no. 3, pp. 16–17, March 2000 8.4.4

[90] Fraser, B. (Ed.) “Site Security Handbook”. RFC 2196, Network Working Group, 1997 106

[91] Fábrega, F. J.; Herzog, J. C.; Guttman, J. D. “Strand Spaces: Why is a Security Protocol Correct?”, in Proc. “IEEE Symp. on Security and Privacy”, pp. 160–171, Oakland, CA, May 1998 86, A.2.2

[92] Garcia-Molina, H. “Elections in a distributed computing system”, IEEE Transactions on Computers, vol. C-31, no. 1, pp. 48-59, January 1982 7.1

[93] Garfinkel, S.; Spafford, G. “Practical UNIX and Internet Security”, 2nd Edition, O’Reilly & Associates, Inc., 1996 20, 106

[94] Garg, V. K.; Mitchell, J. R. “Implementable Failure Detectors in Asynchronous Systems”, Proc. 18th Conf. on Foundations of Software Technology and Theoretical Computer Science, pp. 158–169, Chennai, India, December 1998 6.5, 7.2.3

[95] Gellert, O.; Goldbach, G. “httpf — Ein filternder Proxy”, Tagungsunterlagen 7. Workshop des “DFN-CERT, Zentrum für sichere Netzdienste GmbH” und des Projektes “PCA im DFN”, 8./9. März 2000, Hamburg, in DFN-Bericht Nr. 90 10.3.2

[96] Gligor, V. D. “On Denial-of-Service in Computer Networks”, Proc. Interna. Conf. on Data Engineering, pp. 608–617, Los Angeles, CA, USA, February 1986 8.4.4

[97] Goldberg, M. W.; Neufeld, G. W.; Ito, M. R. “A Parallel Approach to OSI Connection-Oriented Protocols”, in Proc. of the IFIP WF6.1/WF6.4 Third Intern. Workshop, Stockholm, Sweden, 13-15 May 1992, in Pehrson, B.; Gunningberg, P.; Pink, S. (eds) Protocols for High Speed Networks III, pp. 219–232, North-Holland, 1993 4.2.2, 38, 40

[98] Gong, L. “Securely Replicating Authentication Services”, Proc. 9th Intern. Conf. on Distributed Computing Systems, pp. 85–91, Newport Beach, CA, June 1989 4

[99] Gong, L.; Needham, R.; Yahalom, R. "Reasoning about Belief in Cryptographic Protocols", in Proc. "IEEE Conf. on Research in Security and Privacy", pp. 234–248, Oakland, CA, May 1990 81, 83, 8.4.1.2, 8.4.3, 8.5.1, A.2.1, 111

[100] Gong, L.; Syverson, P. "Fail-Stop Protocols: An Approach to Designing Secure Protocols", in Iyer, R. K.; Morganti, M.; Fuchs, W. K.; Gligor, V. (eds.) "Dependable Computing for Critical Applications 5", pp 79–100, IEEE Computer Society, 1998 88

[101] Gray, J. "The Cost of Messages", Proc. 7th Annual ACM Symp. on Principles of Distributed Computing, pp. 1–7, Toronto, Ontario, Canada, August 1988 7.1

[102] Großklaus, A. "Policy, Vorfallsbearbeitung, Schwachstellenanalyse", in [176], S. 53–68 2.2.4, 2.2.4

[103] Guha, B.; Mukherjee, B. "Network Security via Reverse Engineering of TCP Code: Vulnerability Analysis and Proposed Solutions", IEEE Network, vol. 11, no. 4, pp. 40–48, July/August 1997 12, 7.2.5.1, 8

[104] Haas, M.; Zorn, W. "Methodische Leistungsanalyse von Rechensystemen", Oldenbourg, 1995 3.1, 3.2

[105] Han, S.; Shin, K. G. "Experimental Evaluation of Behavior-Based Failure-Detection Schemes in Real-Time Communication Networks", IEEE Transactions on Parallel and Distributed Systems, pp. 613–626, vol. 10, no. 6, June 1999 7.2.3

[106] Harkins, D.; Carrel, D. "The Internet Key Exchange (IKE)", Request for Comments: 2409, Network Working Group, November 1998 8.5.5

[107] Harris, B.; Hunt, R. "Review: TCP/IP security threats and attack methods", Computer Communications, vol. 22, pp. 885–897, 1999 8, 9.4

[108] Hauser, R. C.; Lee, E. S. "Verification and Modelling of Authentication Protocols", in Proc. "2nd European Symp. on Research in Computer Security (ESORICS92)", Toulouse, France, November 1992 83

[109] Haverkort, B.R. "Performance of Computer Communication Systems. A Model-Based Approach", Wiley & Sons, 1998 3.1

[110] Haverkort, B.R.; Niemegeers, I.G. "A Survey of Performability Modelling Tools", Universiteit Twente, Faculteit der Informatica, Enschede, the Netherlands, September 1989 2.2.4

[111] Heidtmann, K. "Zuverlässigkeitsanalyse technischer Systeme", Teubner-Texte zur Informatik, Band 21, Teubner 1997 6.1.2

[112] Heiss, H.-U. “Prozessorzuteilung in Parallelrechnern”, Reihe Informatik, Band 98, BI-Wissenschaftsverlag, Mannheim, 1994 4.2.1.2

[113] Heitmeyer, C. “Applying Practical Formal Methods to the Specification and Analysis of Security Properties”, in Gorodetski, V.I.; Skormin, V.A.; Popyack, L.J. (eds.) Information Assurance in Computer Networks, LNCS 2052, pp. 84–89, Springer-Verlag 2001 86

[114] Hennessy, J. L.; Patterson, D. A. “Computer architecture: a quantitative approach”, 2nd Edition, Morgan Kaufmann, San Francisco, 1996 4.2.1.1

[115] Hinde, S. “Live Was Simple Then”, Computer & Secutity, vol. 19 (2000), no. 8, pp. 222–229 12

[116] Hinton, H. M. “Composing Partially-Specified Systems”, Proc. IEEE Symp. on Security and Privacy, pp. 27–37, Oakland, CA, USA, May 1998 2.3, 8.1

[117] Hinton, H. M. “Under-Specification, Composition and Emergent Properties”, Proc. New Security Paradigms Workshop 1997, pp. 83–93, Langdale, Cumbria, UK, in: ACM SIGSAC, vol. 15, no. 3 2.3, 8.1

[118] Hogrefe, D. “Estelle, LOTOS und SDL. Standard-Spezifikationssprachen für verteilte Systeme”, Springer-Verlag 1989 7.2, A

[119] Holst, S. “Absicherung von Netzdiensten am Beispiel des ATMARP-Dienstes”, Diplomarbeit am Fachbereich Informatik, Universität Hamburg, in Vorbereitung 2.1.2

[120] Holzmann, G. J. “Design and Validation of Computer Protocols” Prentice Hall, New Jersey, 1991 71, 7.3.1, 8.4.2

[121] Intern. Telecommunication Union (ITU-T) “CCITT Specification and Description Language (SDL)”, ITU-T Recommendation Z.100, March 1993 7.2, A

[122] Intern. Telecommunication Union (ITU-T) “Message Sequence Chart (MSC)”, ITU-T Recommendation Z.120, Oktober 1996 7.2, A, A.2

[123] Intern. Telecommunication Union (ITU-T) “SDL Methodology Guidelines. SDL Bibliography”, ITU-T Recommendation Z.100 – Appendices I and II, March 1993 7.2, A

[124] Intern. Telecommunication Union (ITU-T) “SDL combined with ASN.1 (SDL/ASN.1)”, ITU-T Recommendation Z.105, March 1995 7.2, A

[125] Intern. Telecommunication Union (ITU-T) “SDL+ methodology: Use of MSC and SDL (with ASN.1)”, ITU-T Recommendation Z.100 – Supplement 1, May 1997 7.2, A, A.2

[126] Intern. Telecommunication Union (ITU-T) “Specifikation and Description Language (SDL)”, ITU-T Recommendation Z.100 – Annex F1, March 1993 A.1.2

[127] Jahanian, F.; Fakhouri, S.; Rajkumar, R. “Processor Group Membership Protocols: Specification, Design and Implementation”, Proc. 12th Symp. on Reliable Distributed Systems, pp. 2–11, Princeton, New Jersey, USA, October 1993 7.1

[128] Jain, R. “FDDI Handbook. High-Speed Networking Using Fiber and Other Media”, Addison-Wesley 1994 3, 7.2.2

[129] Jain, R. “The Art of Computer Systems Performance Analysis. Techniques for Experimental Design, Measurement, Simulation, and Modeling”, Wiley & Sons, 1991 3.1

[130] Joseph, M. K. “Architectural Issues in Fault-Tolerant, Secure Computing Systems”, Computer Science Department, University of California, Los Angeles, Report CSD-880047, September 1988 6.4, 6.5

[131] Katz, S.; Lincoln, P.; Rushby, J. “Low-overhead Time-Triggered Group Membership”, Proc. 11th Intern. Workshop on Distributed Algorithms, WDAG’97, pp. 155–169, Saarbrücken, Germany, September 1997 7.1

[132] Katz, S.; Perry, K. J. “Self-stabilizing Extensions for Message-passing Systems”, Proc. 9th Annual ACM Symp. on Principles of Distributed Computing, pp. 91–101, Quebec-City, Quebec, Canada, August 1990 6.5

[133] Kerber, L. “Scenario-Based Performance Evaluation of SDL/MSC-Specified Systems”, in Dumpke, R. et al. (eds.) Performance Engineering, LNCS 2047, pp. 185–201, Springer-Verlag 2001 109

[134] Kerner, H. “Rechnernetze nach OSI”, 3. Auflage, Addison-Wesley, Bonn, 1995 4.2.2, 6.4

[135] Kernighan, B. W.; Ritchie, D. M. “The C Programming Language. ANSI C”, 2nd edition, Prentice-Hall 1988 9.1

[136] Kihlstrom, K. P.; Moser, L. E.; Melliar-Smith, P. M. “The SecureRing Protocols for Securing Group Communications”, 31st Hawaii Intern. Conf. on System Sciences, pp. 317–326, Kohala Coast, HI, USA, January 1998 7.1

[137] Kim, K. H. et al. “An Efficient Decentralised Approach to Processor-Group Membership Maintenace in Real-Time LAN Systems: The PRHB/ED Scheme”, Proc. 11th Symp. on Reliable Distributed Systems, pp. 74–83, Houston, Texas, USA, October 1992 7.1

[138] Kossakowski, K.-P. “Information Technology Incident Response Capabilities”, Dissertation am Fachbereich Informatik der Universtiät Hamburg, Hamburg 2000 1, 2.2.2, 2.2.4

[139] Kossakowski, K.-P. “Virtuelle Private Netzwerke: Einsatzmöglichkeiten und Techniken”, 5. DFN-CERT Workshop 1998 “Sicherheit in vernetzten Systemen”, 4./5. März 1998, Hamburg, in DFN-Bericht Nr. 85, Mai 1998 12, 15

[140] Krawczyk, H.; Bellare, M.; Canetti, R. “HMAC: Keyed-Hashing for Message Authentication”, Network Working Group, Request for Comments: 2104, February 1997 8.5.1, 8.6.1

[141] Kyas, O. “Sicherheit im Internet”, 2. Auflage, Intern. Thomson Publishing, Bonn 1998

[142] Ladkin, P. B. “Causal Reasoning about Aircraft Accidents”, Proc. Computer Safety, Reliability, and Security (SAFECOMP 2000), pp. 344–360, Rotterdam, Netherlands, October 2000 5

[143] Lamport, L. “The weak Byzantine generals problem”, Journal of the ACM, vol 30, no. 3, pp. 668–676, July 1983 7.1

[144] Lamport, L.; Merz, S. “Specifying and Verifying Fault-Tolerant Systems”, Proc. 3rd Inter. Symp. on Formal Techniques in Real Time and Fault Tolerant Systems, pp. 41–76, Vol. 863 of LNCS, Lübeck, Germany, September 1994 2.2.4

[145] Lampson, B.; Abadi, M.; Borrows, M. “Authentication in Distributed Systems: Theory and Practice”, ACM Transactions on Computer Systems, vol. 10, no. 4, pp. 265–310, November 1992 109

[146] Langendörfer, H. “Leistungsanalyse von Rechensystemen. Messen, Modellieren, Simulation”, Hanser 1992 3.2

[147] Laprie, J.-C. “Dependable Computing and Fault Tolerance: Concepts and Terminology”, Proc. 15th Annual Intern. Sympos. on Fault-Tolerant Computing, Ann Arbor, Michigan, USA, June 1995 63, 3, 1

[148] Laubach, M. “Classical IP and ARP over ATM”. RFC 1577, Network Working Group, Januar 1994 2.1.2

[149] Laubach, M; Halpern, J. “Classical IP and ARP over ATM”. RFC 2225, Network Working Group, April 1998 2.1.2

[150] Leiwo, J.; Zheng, Y. “A Method to Implement a Denial of Service Protection Base”, Proc. Information Security and Privacy, Second Australasian Conf., ACISP’97, pp. 90–101, Sydney, Australia, July 1997 8.4.4

[151] Lessig, A. "'Active Content' und mobiler Code", in [176], S. 125–144 10.3.2

[152] Lewine, D. A. "POSIX Programmer's Guide. Writing Portable UNIX Programs", O'Reilly 1994 9.1

[153] Liedtke, J.; Islam, N.; Jaeger, T. "Preventing Denial-of-Service Attacks on a μ-Kernel for WebOSes", 6st Workshop on Hot Topics in Operating Systems, pp. 73–79, Cape Cod, MA, USA, May 1997 8.4.4

[154] Linden van der, P. "Expert C Programming. Deep C Secrets", SunSoft Press, Mountain View, 1994 2.2.4, 9.6.2

[155] Linger, R.C.; Mead, N.R.; Lipson, H.F. "Requirements Definition for Survivable Network Systems", Proc. Int. Conf. on Requirements Engineering, Colorado Springs, CO, USA, pp. 14–23, April 6-10, 1998 5, 2.2.2, 2.2.4

[156] Lowe, G. "Breaking and Fixing the Needham-Schroeder Public-key Protocol using FDR", in Proc. "2nd Intern. Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)", pp 146–166, Passau, Germany, March 1996 8.4.2

[157] Malkin, G. "RIP Version 2 Protocol Analysis", Network Working Group, Request for Comments: 1387, January 1993 12

[158] Mandeville, R. "Benchmarking Terminology for LAN Switching Devices", Request for Comments: 2285, Network Working Group, February 1998 3.2.1.1, 3, 3.2.2.3, 1, 2

[159] Mandeville, R.; Perser, J. "Benchmarking Methodology for LAN Switching Devices", Request for Comments: 2889, Network Working Group, August 2000 3

[160] Mao, W. "An Augmentation of BAN-Like Logics", in Prooceedings "8th IEEE Computer Security Foundations Workshop", pp. 44–56, Kenmare, Ireland, June 1995 83, A.2.1

[161] Mao, W.; Boyd, C. "Towards Formal Analysis of Security Protocols", in Proc. "6th Computer Security Foundations Workshop", New Hampshire, USA, June 1993 81, 83, A.2.1

[162] McCanne, S.; Jacobson, V. "The BSD Packet Filter: A New Architecture for User-level Packet Capture", Proc. USENIX conference, pp. 259–269, January 25–29, 1993, San Diego, CA, USA 2

[163] Meadows, C. "A Formal Framework and Evaluation Method for Network Denial of Service", Proc. 12th IEEE Computer Security Foundations Workshop, pp. 4–13, Mardano, Italy, June 1999 8.4.4, 87, 88, A.2.3, A.2.3, 109

[164] Meadows, C. “Open Issues in Formal Methods for Cryptographic Protocol Analysis”, Proc. DARPA Information Survivability Conf. and Exposition, DISCEX’00, pp. 237–250, January 2000, Hilton Head, SC, USA 2.2, 87

[165] Meadows, C. “The NRL Protocol Analyzer: An Overview”, Journal of Logic Programming, Vol. 26, No. 2, pp. 113–131, February 1996 8.4.2

[166] Meadows, C. “Three Paradigms in Computer Security”, Proc. New Security Paradigms Workshop 1997, pp. 34–37, Langdale, Cumbria, UK, in ACM SIGSAC, vol. 15, no. 3

[167] Meadows, C.; McLean, J. “Security and Dependability: Then and Now”, Proc. Computer Security, Dependability, and Assurance: From Needs to Solutions, pp. 166-170, York, UK & Williamsburg, VA, USA, July & November 1998 2.2

[168] Melliar-Smith, P. M.; Moser, L. E. “Fault-Tolerant Distributed Systems Based on Broadcast Communication”, Proc. 9th Intern. Conference on Distributed Computing Systems, pp. 129–134, Newport Beach, CA, USA, June 1989 6.5, 7.1

[169] Mishra, S.; Fetzer, C.; Cristian, F. “The Timewheel Group Membership Protocol”, Proc. Parallel and Distributed Processing, in Conjunction with the 12th Intern. Parallel Processing Symp. and 9th Symp. on Parallel and Distributed Processing, IPPS/SPDD’98, pp. 664–680, Orlando, FL, USA, March/April 1998 68, 7.1, 76

[170] Mishra, S.; Schlichting, R. D. “Abstractions for constructing dependable distributed systems”. Technical Report 92-19, Dept of Computer Science, University of Arizona, Tucson, AZ, 1992 6.4, 6.5

[171] Mitschele-Thiel, A.; Müller-Clostermann, B. “Performance engineering of SDL/MSC systems”, Computer Networks, Vol. 31, No. 17, pp. 1801–1815, June 1999 109

[172] Mockapetris, P. “Domain Names - Concepts and Facilities”, Request for Comments: 1034, Network Working Group, November 1987 10.3.1

[173] Mockapetris, P. “Domain Names - Implementation and Specification”, Request for Comments: 1035, Network Working Group, November 1987 10.3.1

[174] Moorsel, A.P.A. “Performability Evaluation Concepts and Techniques”, Dissertation, Universiteit Twente, Faculteit der Informatica, Enschede, the Netherlands, December 1993 2.2.4

[175] Moy, J. “OSPF Version 2”, Network Working Group, Request for Comments: 1387, January 1993 12

[176] Mück, H.-J.; Benecke, C.; Kelm, S. (Hrsg.) “Sicherheit in vernetzten Systemen”, Bericht 224 (FBI-HH-B-224/00), Fachbereich Informatik, Universität Hamburg, Februar 2000 64, 102, 151

[177] Nelson, R. “Integrating Formalism and Pragmatism: Architektural Security”, Proc. New Security Paradigms Workshop 1997, pp. 1–4, Langdale, Cumbria, UK, in ACM SIGSAC, vol. 15, no. 3

[178] Nessett, D.; Humenn, P. “The Multilayer Firewall”, Proc. Symp. on Network and Distributed System Security, San Diego, CA, USA, March 1998

[179] Neumann, P. G. “A System-Oriented Perspective of Survivability”, position statement, Information Survivability Workshop - ISW ’98, Orlando, Florida USA, October 28-30, 1998 2.2.2

[180] Neumann, P. G. “Inside Risk. The Big Picture”, Communications of the ACM, vol. 40, no. 5, pp. 136, May 1997 7

[181] Neumann, P. G. “Practical Architectures for Survivable Systems and Networks. Phase-Two Final Report”, SRI Intern., June 2000 2.2.1, 2.2, 2.2.2, 2.2.4, 2.2.4

[182] Newman, D. “Benchmarking Terminology for Firewall Performance”, Request for Comments: 2647, Network Working Group, August 1999

[183] Oram, A.; Talbott, S. “Managing Projects with make”, 2nd edition, O’Reilly, Dezember 1991 9.2.2

[184] Osso, R. (Ed.) “Handbook of Emerging Communications Technologies. The Next Decade”, CRC Press LLC, 2000 39

[185] Parks, T. M.; Kassay, D. A.; Weinstein, C. J. “Vulnerabilities of Reliable Multicast Protocols”, Proc. IEEE Military Communications Conference, MILCOM’98, pp. 934–938, Boston, MA, USA, October 1998 7.1

[186] Paxson, V. “Bro: A System for Detecting Network Intruders in Real-Time”, Computer Networks, Vol. 31, No. 23–24, pp. 2435–2463, December 1999 10.2.3

[187] Pfleeger, C. P. “Security in Computing”, 2nd Edition, Prentice-Hall, New Jersey, 1997 2.2.3, 2.2.4, 2.2.4, A.2.3

[188] Plessel, C.; Wilde, E. “Dienstbare Geister. Server-Side-Techniken im Überblick”, IX, Magazin für professionelle Informationstechnik, Nr. 3/2001, S. 88-93, März 2001 10.3.2

[189] Plummer, D. C. “An Ethernet Address Resolution Protocol”, Request for Comments: 826, Network Working Group, November 1982 1

[190] Postel, J. “Internet Protocol”, Information Sciences Institute, University of Southern California, Request for Comments: 791, September 1981 7.2.5.3

[191] Postel, J. “Transmission Control Protocol”, Information Sciences Institute, University of Southern California, Request for Comments: 793, September 1981 7.2.5.3

[192] Preneel, B. “Hash Functions and MAC Algorithms Based on Block Ciphers” in Darnell, M. (Ed.) Proc. Cryptography and Coding, Cirencester, UK, Dec. 1997 8.5.1

[193] Ramakrishnan, K. K.; Yang, H. “The Ethernet Capture Effect: Analysis and Solution”, Proc. 19th Conference on Local Computer Networks, pp. 228–240, Minneapolis, Minn, USA, October 1994 1

[194] Ranum, M. J. “Intrusion Detection and Network Forensics”, Tutorial T4, “16th Annual Computer Security Applications Conference” (ACSAC’00), New Orleans, LA, USA, December 2000 10.2.3

[195] Reiter, M.; Birman, K.; Gong, L. “Integrating Security in a Group Oriented Distributed System.”, in Proc. “IEEE Computer Society Symp. on Research in Security and Privacy”, pp. 18–32, Oakland, CA, USA, May 1992 7.1

[196] Reiter, M. K. “A Secure Group Membership Protocol”, IEEE Transactions on Software Engineering, vol. 22, no. 1, pp. 31–42, January 1996 7.1

[197] Reiter, M. K.; Birman, K. P.; van Renesse, R. “A Security Architecture for Fault-Tolerant Systems”, ACM Transactions on Computer Systems, vol. 12, no. 4, pp. 340–371, November 1994 7.1

[198] Rivest, R. “The MD5 Message-Digest Algorithm”, Network Working Group, Request for Comments: 1321, April 1992 8.5.1, 8.6.1

[199] Rodeh, O.; Birman, K. P.; Dolev, D. “The Architecture and Performance of Security Protocols in the Ensemble Group Communication System”, Technical Report: TR2000-1822, Cornell University, Computer Science, October 2000 7.1

[200] Rodrigues, L.; Veríssimo, P.; Rufino, J. “A Low-Level Processor Group Membership Protocol for LANS”, Proc. 13th Intern. Conf. on Distributed Computing Systems, pp. 541–550, Pittsburgh, Pennsylvania, USA, May 1993 7.1

[201] Russel, R. “Linux IPCHAINS-HOWTO”, v1.0.8, Online (`http://www.linux.org/docs/ldp/howto/IPCHAINS-HOWTO.html`) 4.3.7

[202] Ryan, P. Y. A.; Schneider, S. A. “The Modelling and Analysis of Security Protocols: the CSP Approach”, Addison-Wesley, 2001 8.4.3

[203] Schneeweiß, W.G. “Grundbegriffe für praktische Zuverlässigkeitsanalysen”, Datakontext-Verlag, Köln 1981 6.1.2

[204] Schneeweiß, W.G. “Zuverlässigkeitstechnik: von den Komponenten zum System”, Nr. 1, 2. Ausg., Datakontext, 1992 2

[205] Schneider, F. B. “Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial”, ACM Computing Surveys, vol. 22, no. 2, pp. 299–319, December 1990 6.2.1.2, 5, 6.4, 6.5, 7.1

[206] Schneier, B. “Applied Cryptographie. Protocols, Algorithms, and Source Code in C”, 2nd Edition, John Wiley & Sons, 1996 8.5.1

[207] Schuba, C. L. et al. “Analysis of a Denial of Service Attack on TCP”, IEEE Symp. on Security and Privacy, pp. 208–223, Oakland, CA, USA, May 1997 12, 7.2.5.1, 8, 8.4.4

[208] Schudel, G.; Wood, B. “Adversary Work Factor as a Metric for Information Assurance”, Proc. New Security Paradigm Workshop, pp. 23–30, Ballycotton, Ireland, September 2000

[209] Schwartau, W. “Surviving Denial of Service”, Computer & Security, Vol. 18, No. 2, pp. 124–133, 1999 12, 8.4.4

[210] Seifert, R. “The Effect of Ethernet Behavior on Networks using High-Performance Workstations and Servers”, Technical Report, Networks and Communications Consulting, 1995
(`http://wwwhost.utexas.edu/ethernet/papers.hmtl`) 1

[211] Shambroom, W. D. “Use of Protocol Validation and Verification Techniques in the Design of a Fault-Tolerant Computer Architecture”, 23rd Annual Intern. Symp. on Fault-Tolerant Computing, pp. 636–640, Toulouse, France, June 1993 2.2.4

[212] Shouhuai, X.; Zhang, G.; Zhu, H. “On the Properties of Cryptographic Protocols and the Weaknesses of the BAN-like logics”, ACM SIGOPS, Operating Systems Review, Vol. 31, No. 4, pp. 12–23, October 1997 81

[213] Singhai, A.; Lim, S.-B.; Radia, S. R. “The SunSCALAR Framework for Internet Servers”, Proc. 28th Annual Intern. Symp. on Fault-Tolerant Computing, FTCS-28, pp. 108–117, Munich, Germany, June 1998 10

[214] Skillicorn, D. B. “A Taxonomy for Computer Architectures”, IEEE Computer, vol. 21, no. 11, pp. 46–57, November 1988 4.2.1.1

[215] Soh, B. C.; Young, S. “Distributed computing: an experimental investigation of a malicious denial-of-service applet”, Computer Communications, vol. 21, pp. 670–674, 1998 8.4.4

[216] Song, D. X. “Athena: a New Efficient Automatic Checker for Security Protocol Analysis”, Proc. 12th IEEE Comp. Security Foundations Workshop, pp. 192–202, Mordano, Italy, June 1999

[217] Song, D. X.; Berezin, S.; Perrig, A. “Athena: a novel approach to efficient automatic security protocol analysis”, Journal of Computer Security, 2000

[218] Song, J. H.; Li, L.; Kappler, P.; Freimark, R.; Kozlik, G. T. “An Open Solution to Fault-Tolerant Ethernet: Design, Prototyping, and Evaluation”, Proc. 18th IEEE Intern. Performance, Computing, and Communications Conf., pp. 461–468, Scottsdale, Arizona, USA, February 1999 6.2.1.1

[219] Stallings, W. “High-Speed Networks: TCP/IP and ATM Design Principles”, Prentice Hall 1998 7.2.2

[220] Stevens, W. R. “UNIX Network Programming. Networking APIs: Sockets and XTI”, 2nd Edition, Prentice-Hall, New Jersey 1997 106

[221] Stieler, W. “Cracker stehlen Microsoft-Sourcecode”, Heise Online News, Oktober 2000, (`http://www.heise.de/newsticker/data/wst-27.10.00-001/`) 1

[222] Stoller, S. D. “Leader Election in Asynchronous Distributed Systems”, IEEE Transactions on Computers, vol. 49, no. 3, pp. 283–284, March 2000 7.1

[223] Sun Microsystems “Programming Utilities Guide. Solaris 2.6”, Part No. 802-5880-10, Revision A, Mountain View, 1997 9.2.2

[224] Sun Microsystems “STREAMS Programming Guide. Solaris 2.6”, Part No. 802-5893-10, Revision A, Mountain View, 1997 4.3.8, 9.1

[225] Syverson, P. “The Use of Logic in the Analysis of Cryptographic Protocols”, in Proc. “IEEE Conf. on Research in Security and Privacy”, pp. 156–170, Oakland, CA, May 1991 81, A.2.1

[226] Terplan, K. “Communication Networks Management”, 2nd Edition, Prentice-Hall, New Jersey, 1992 2.1

[227] Trivedi, K. S.; Malhotra, M. “Reliability and Performability Techniques and Tools: A Survey”, in Walke, B. and Spaniol, O. (Hrsg.) Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen, S. 27–48, Aachen, September 1993 2.2.4

[228] UNIX Intern. Inc., OSI Work Group “Data Link Provider Interface Specification”, Revision 2.0.0, August 1991 9.1, 2, C.3.3

[229] Vahalia, U. “UNIX Internals. The new Frontiers”, Prentice-Hall, New Jersey 1996 26, 4.3.8, 17, 9.1

[230] Varadarajan, S.; Chiueh, T. “Automatic Fault Detection and Recovery in Real Time Switched Ethernet Networks”, Proc. 18th Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM’99, pp. 161–169, New York, NY, USA, March 1999 6.5

[231] Varghese, G. “Self-stabilization by Counter Flushing”, Proc. 13th Annual ACM Symp. on Principles of Distributed Computing, pp. 244–253, Los Angeles, CA, USA, August 1994 7.1

[232] Viega, J.; Bloch, J. T.; Kohno, T.; McGraw, G. “ITS4: A Static Vulnerability Scanner for C and C++ Code”, Proc. “16th Annual Computer Security Applications Conference” (ACSAC’00), pp. 257–267, New Orleans, LA, USA, December 2000 106

[233] Voydock, V. L.; Kent, S. T. “Security Mechanisms in High-Level Network Protocols”, Computing Surveys, vol. 15, no. 2, pp. 135–171, June 1983 109

[234] Vázquez-Gómez, J. “Multidomain security”, Computer & Security, Vol. 13, No. 2, pp. 161–184, April 1994 2.3.1.1

[235] Wang, F.; Ramamrithan, K.; Stankovic, J.A. “Determining Redundancy Levels for Fault Tolerant Real-Time Systems”, IEEE Transactions on Computers, Vol. 44, No. 2, pp. 292–301, February 1995 2.2.4

[236] Wellington, B. “Domain Name System Security (DNSSEC) Signing Authority”, Request for Comments: 3008, Network Working Group, November 2000 10.3.1

[237] Wellington, B. “Secure Domain Name System (DNS) Dynamic Update”, Request for Comments: 3007, Network Working Group, November 2000 10.3.1

[238] West-Brown, M. J. et al. “Handbook for Computer Security Incident Response Teams (CSIRTs)”. *HANDBOOK, CMU/SEI-98-HB-001*, Dezember 1998 106

[239] Wolfinger, B. E.; Zaddach, M.; Bai, G.; Heidtmann, K. D. “Modelling of Primary and Secondary Load in the Internet”, Bericht 227 (FBI-HH-B-227/00), Fachbereich Informatik, Universität Hamburg, November 2000 3.2.2.1

[240] Woodside, C. M.; Franks, G. “Alternative Software Architectures for Parallel Protocol Execution with Synchronous IPC”, IEEE/ACM Transactions on Networking, Vol. 1, No. 2, pp. 187–186, April 1993 4.2.2, 4.3.1

[241] Xu, J.; Singhal, M. “Design of a High-Performance ATM Firewall”, Proc. 5th ACM Conference on Computer and Communication Security, pp. 92–102, San Francisco, CA, USA, November 1998 4.1

[242] Yasinac, A. “Dynamic Analysis of Security Protocols”, Proc. New Security Paradigm Workshop, pp. 77–87, Ballycotton, Ireland, September 2000

[243] Zhang, Y.; Paxson, V. “Detecting Backdoors”, Proc. of the 9th USENIX Security Symp., 14–17 August 2000, Denver, Colorado, USA 10.2.3

[244] Zhang, Y.; Paxson, V. “Detecting Stepping Stones”, Proc. of the 9th USENIX Security Symp., 14–17 August 2000, Denver, Colorado, USA 10.2.3

[245] Zitterbart, M. “Funktionsbezogene Parallelität in transportorientierten Kommunikationsprotokollen”, VDI-Reihe 10, Nr. 183, VDI-Verlag, Düsseldorf, 1991 4.2.2, 4.3.1

[246] Zitterbart, M. “High-Speed Transport Components”, IEEE Network Magazine, Vol. 5, No. 1, pp. 54–63, January 1991 38, 4.4

[247] de Vivo, M.; de Vivo, G. O.; Isern, G. “Internet Security Attacks at the Basic Levels”, Operating Systems Review, ACM SIGOPS, vol. 32, no. 2, pp. 4–15, April 1998 8

[248] de Vivo, M.; de Vivo, G. O.; Koeneke, R.; Isern, G. “Internet Vulnerabilities Related to TCP/IP and T/TCP”, Computer Communication Review, ACM SIGCOMM, vol. 29, no. 1, pp. 81–85, January 1999 8