PAKETVERARBEITENDE SYSTEME – ALGORITHMEN UND ARCHITEKTUREN FÜR HOHE VERARBEITUNGSGESCHWINDIGKEITEN

Dissertation
zur
Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)
der Fakultät für Informatik und Elektrotechnik
der Universität Rostock

vorgelegt von
Widiger, Harald, geb. am 23.08.1978 in Halle (Saale)
aus Rostock
URN: urn:nbn:de:gbv:28-diss2009-0144-6

Rostock, den 23.11.2008

Gutachter:	Prof. Dr. Dirk Timmermann,Universität Rostock
	Prof. Dr. Volker Kühn,Universität Rostock
	Prof. Dr. Rolf Ernst,Technische Universität Braunschweig
Tag der Verteidigung:	28. Mai 2009

Danksagung

An dieser Stelle möchte ich in erster Linie meiner Familie danken, die mich auf meinem nunmehr schon 23 Jahre währenden Bildungsweg begleitet und stets aufopfernd unterstützt hat. Ohne sie wäre es nicht möglich gewesen, diese wissenschaftliche Arbeit zu verfassen.

Zu großem Dank bin ich auch meinem Doktorvater Professor Timmermann verpflichtet, der es erst ermöglicht hat, am Institut für Angewandte Mikroelektronik und Datentechnik zu arbeiten und zu promovieren. Er sorgte stets dafür, dass mir der notwendige Freiraum blieb, um diese Arbeit zu beenden und dass auch in Zeiten von Tiefs und Schwierigkeiten das Ziel nicht aus den Augen verlogen wurde.

Dank gilt selbstverständlich auch Andreas Tockhorn, Peter Danielis und Barbara Morawietz, die diese Arbeit Korrektur gelesen haben, sowie allen an dieser Arbeit beteiligten Kollegen und Studenten. Dabei sind insbesondere die Mitglieder meiner Forschungsgruppe Stephan Kubisch, Daniel Duchow und Thomas Bahls zu nennen.

Schließlich möchte ich noch allen Mitarbeitern des Instituts für Angewandte Mikroelektronik und Datentechnik dafür danken in dieser freundlichen und netten Atmosphäre arbeiten zu können.

Inhalt

Abkürzungen ..... **IX**
1 Einleitung **1**
2 Grundlagen **5**
2.1 Paketverarbeitende Systeme 5
2.1.1 Netzwerktheoretische Grundlagen 6
2.1.2 Paketverarbeitung 11
2.1.3 Paketklassifizierung 14
2.2 Grundlagen des Hashings 18
2.2.1 Hashing 19
2.2.2 Anwendungsbereiche von Hashfunktionen 22
2.3 Grundlagen evolutionärer Algorithmen 24
2.3.1 Genetische Algorithmen 25
2.3.2 Evolvable Hardware 32
3 Architekturen für paketverarbeitende Systeme **35**
3.1 Paketverarbeitende Systeme 35
3.1.1 Prozessorarchitekturen für Softwarelösungen 36
3.1.2 Existierende Speziallösungen für ASICs 38
3.1.3 Lösungen auf FPGA-Basis 40
3.1.4 Zusammenfassung 43
3.2 Paketklassifizierung 44
3.2.1 Softwarelösungen 44
3.2.2 Existierende Hardwarelösungen 46
3.2.3 Hashing als Mittel der Paketklassifizierung 53
3.2.4 Caching zur Paketklassifizierung 63
4 Paketverarbeitende Systeme im Teilnehmerzugangsnetzwerk **65**
4.1 Funktionale und architektonische Anforderungen 65
4.2 Architektur funktionaler Module 66
4.2.1 Framebuffer mit Parser 68
4.2.2 Controller und Arbiter 69
4.2.3 Speichersystem 71
4.2.4 Ausführungseinheit 73
4.2.5 Zusammenfassung 73
4.3 Beispiele paketverarbeitender Systeme im TZN 74
4.3.1 MPLS User Network Interface 74
4.3.2 Traffic Manager 77
4.3.3 Mac Address Translation (MAT) 80
4.3.4 Leistungsfähigkeit der funktionalen Module 82

4.3.5 IPclip ... 85
4.4 Kombination funktionaler Module ... 97
4.4.1 Pipelining am Beispiel von MATMUNI ... 98
4.4.2 Blockorientierte Verarbeitung ... 101
4.4.3 Networks-on-Chip ... 103
4.5 Zusammenfassung ... 104
5 Evolvierbare Paketklassifizierung in Hardware ... 105
5.1 Ein evolvierbarer Hash-basierter Paketklassifizierer ... 105
5.1.1 Gesamtsystem des evolvierbaren Paketklassifizierers ... 105
5.1.2 Datenpfad ... 107
5.1.3 Kontrollpfad ... 111
5.1.4 Verschränkung von Daten- und Kontrollpfad ... 111
5.2 Die evolvierbare Hashfunktion ... 113
5.2.1 Zwei Multiplexer mit XOR-Verknüpfung ... 115
5.2.2 Zwei Multiplexer in zwei Stufen XOR-Verknüpft ... 116
5.2.3 Drei Multiplexer mit XOR-Verknüpfung ... 117
5.2.4 Multiplexer mit variabler Logikverknüpfung ... 117
5.2.5 Evolvierbare CRC32-Berechnung ... 118
5.2.6 Evolvable H_3 Hashing ... 118
5.3 Der eingesetzte genetische Algorithmus ... 119
5.3.1 Fitnessfunktion und Abbruchkriterium ... 120
5.3.2 Variationsoperatoren ... 121
5.4 Leistungsvergleich verschiedener Hashfunktionen ... 122
5.4.1 Testdaten ... 122
5.4.2 Merkmale der verschiedenen Hashfunktionen ... 123
5.4.3 Performance der Hashfunktionen ... 125
5.4.4 Alternative Schlüssel ... 128
5.4.5 Veränderliche Schlüsselmengen ... 129
5.5 Zusammenfassung ... 131
6 Systemoptimierungen ... 133
6.1 Beschleunigung der Fitnessevaluierung (ET, PFE, MI) ... 133
6.1.1 Vorzeitiger Abbruch (ET) ... 136
6.1.2 Speicherverschränkung (MI) ... 138
6.1.3 Parallele Fitnessevaluierung (PFE) ... 139
6.1.4 Kombination der Maßnahmen ... 141
6.2 Parameter des Genetischen Algorithmus ... 144
6.2.1 Populationsgröße ... 144
6.2.2 Selektionsdruck ... 147
6.2.3 Mutationsrate ... 149

6.2.4 Kombination der Maßnahmen ... 153
6.3 Smart Initialization ... 154
6.4 Wiederholt reinitialisierter GA ... 155
6.5 Caching im Datenpfad ... 157
6.5.1 Architektur des Caches ... 158
6.5.2 Experimentelle Ergebnisse ... 160
6.6 Zusammenfassung der sinnvollsten Optimierungen ... 164
7 Zusammenfassung der Ergebnisse ... 165
7.1 Paketverarbeitung ... 165
7.2 Paketklassifizierung ... 166
7.3 Ausblick ... 167
Abbildungen ... 169
Tabellen ... 176
Literatur ... 177
Anhang ... 193
Anhang A ASIC vs. FPGA Größenverhältnisse ... 193
Anhang B TM Farbmarkierungen ... 193
Anhang C IPclip-Option ... 196
Anhang D Abbildungen zu Kapitel 5 ... 198
Anhang E Abbildungen zu Kapitel 6 ... 203

Abkürzungen

ADSL	Asynchronous DSL
AE	Ausführungseinheit
AIA	Additional Information Adder
AIR	Additional Information Remover
AN	Access Network
ARPA	Advanced Research Projects Agency
ASIC	Application Specific Integrated Circuit
ATM	Asynchronous Transfer Mode
BGP	Border Gateway Protocol
BIR	Burst Information Rate
BIR-C	BIR-Counter
BPON	Broadband PON
BRAM	Block RAM
BRAS	Broadband Remote Access Server
CA	Controller & Arbiter
CAM	Content Addressable Memory
CA-RAM	Conent Addressable RAM
CIDR	Classless Inter-domain Routing
CIO	Context Information Output
CIR	Committed Information Rate
CIR-C	CIR-Counter
CLB	Configurable Logic Block
CLIP	Calling Line Identification Presentation
CMAC	Customer MAC
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DEI	Drop Eligable Indicator
DDR	Double Data Rate
DEI	Drop Elegibility Indicator
DHCP	Dynamic Host Configuration Protocol
DKIM	Domain Key Identified Mail
DRAM	Dynamic RAM
DSCP	Differential Services Code Point
DSL	Digital Subscriber Line
DSLAM	DSL Access Multiplexer

DST-IP	Destination IP
DST-MAC	Destination MAC
EA	Evolutionärer Algorithmus
EDF	Earliest Deadline First
EHW	Evolvable Hardware
ELLF	Enhanced Least Laxity First
EP	Evolutionäres Programmieren
EPC	Evolvable Packet Classifier
GEPON	Gigabit Ethernet PON
ES	Evolutionsstrategie
ET	Early Termination
FB	Framebuffer
FCS	Frame Check Sequence
FEX	Field Extractor
FIFO	First In First Out
FM	Funktionales Modul
FMO	Field Modification Unit
FPGA	Field Progammable Gate Array
FPAA	Field Programmable Analog Array
FPTA	Field Programmable Transistor Array
FPX	Field Programmable Port Extender
FSM	Finite State Machine
GA	Genetischer Algorithmus
GLI	Geospacial Location Information
GP	Genetische Programmierung
GPON	Gigabit-capable PON
GPS	Global Positioning System
HA	Hardware-Assist
IEEE	Institute of Electrical Electronics Engineers
IFG	Inter Frame Gap
IO	In Out
IP	Internet Protocol
IPclip	IP-Calling Line Identification Presentation
IPoE	Internet Protocol over Ethernet
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ISDN	Integrated Service Digital Network
ISO	International Stadardisation Organisation
ISP	Internet Service Provider

LAN	Local Area Network
LDP	Label Distribution Protocol
LE	Logic Element
LER	Label Edge Router
LFSR	Linear Feedback Shift Register
LLF	Least Laxity First
LoST	Location-to-Service-Translation
LMP	Longest Matching Prefix
LSP	Label Switch Path
LSR	Label Switch Router
LUT	Look up Table
MAC	Medium Access Control
MAM	IPoE MTU Adaptation Module
MAN	Metropolitan Area Network
MAT	MAC Address Translation
MI	Memory Interleaving
MPLS	Multi Protocol Label Switching
MPLS-UNI	MPLS-User Network Interface
MTA	Mail Transfer Agent
MTU	Maximum Transmission Unit
NID	Network Interface Device
NIDS	Network Intrusion Detection System
NoC	Network-on-Chip
OI	Ortsinformation
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
OVM	Option Verification Module
PAM	PPPoE MTU Adaptation Module
PC	Personal Computer
PFE	Parallel Fitness Evaluation
PLD	Programmable Logic Device
PMAC	Provider MAC
PON	Passive Optical Network
PPE	Programmable Processing Engine
PPP	Point-to-Point Protocol
PPPoE	Point-to-Point Protocol over Ethernet
PSM	Programmable State Machine
QoE	Quality-of-Experience
QoS	Quality-of-Service

RAD	Reprogrammable Application Device
RAM	Random Access Memory
RARP	Reverse ARP
RIP	Routing Information Protocol
RISC	Reduced Instructino Set Computing
RTC	Run-To-Completion
SCA	Subscriber Catchment Area
SDRAM	Synchronous Dynamic Random Access Memory
SI	Smart Initialization
SIP	Session Initialisation Protocol
SM	Smart Initialisation
sMAT	simplified MAT
SMP	Symetric Multiprocessor
SMTP	Simple Mail Transfer Protocol
SoC	System-on-Chip
SRAM	Static RAM
SRC-IP	Source IP
SRC-MAC	Source MAC
SVLAN	Service VLAN
TbW	Trust-by-Wire
TCAM	Ternary CAM
TCP	Transmission Control Protocol
TLV	Type-Lenght-Value
TM	Traffic Manager
TZN	Teilnehmerzugangsnetzwerk
UBE	Uncolicited Bulk E-Mail
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VDSL	Very High Speed DSL
VLAN	Virtual LAN
WAN	Wide Area Network
WRGA	Wiederholt reinitialisierter GA

1 Einleitung

Die Gültigkeit von „Moore's Law", leidlich und weidlich zitiert in unzähligen Veröffentlichungen, ist auch der Grund für die in dieser Arbeit vorgestellten Entwicklungen. Moore postulierte bereits 1965, dass sich die Integrationsdichte von integrierten Schaltkreisen alle 12 bis 18 Monate verdoppelt. Gleiches gilt seit über 40 Jahren auch für Bandbreiten in der Datenkommunikation. Mittlerweile stehen auch privaten Haushalten Bandbreiten von bis zu 100 MBit/s (VDSL2) zur Verfügung.

Der starke Anstieg der zur Verfügung stehenden Bandbreiten stellt für Internet Service Provider (ISPs) eine große Herausforderung dar, denn diese großen Datenmengen müssen auf den Kommunikationsstrecken verarbeitet werden. Um dies zu ermöglichen, verschieben ISPs zunehmend Funktionalitäten aus Kernnetzen in Teilnehmerzugangsnetzwerke (TZNs). Dort ist deshalb bei ansteigenden Bandbreiten zunehmender funktionaler Umfang zu bewältigen. Um dieser anspruchsvollen Aufgabe gerecht zu werden, sind paketverarbeitende Systeme notwendig, deren Leistungsfähigkeit und Funktionsumfang bei der Erforschung der aktuellen Kommunikationstechnik eine herausragende Rolle spielt. Es ist ein herausforderndes Problem, Paketkommunikation schritthaltend mit der wachsenden Bandbreite und funktionalen Diversifikation zu ermöglichen. Zwei Aufgaben sind dabei zu bewältigen:

Paketverarbeitung

Die eigentliche Verarbeitung der Datenpakete (Manipulation, Weiterleitung, Verwurf, Sortierung von Paketen) erfordert hohe Durchsatzraten, eine Verarbeitung in „wirespeed". Es zeigt sich, dass Netzwerkprozessoren zunehmend nicht in der Lage sind, die steigenden Anforderungen in Bezug auf den Datendurchsatz zu befriedigen. Auf Application Specific Integrated Circuits (ASICs) basierende Hardwarelösung sind andererseits nicht flexibel genug, um neue und sich ändernde Funktionen zu realisieren. Deshalb wurde im Rahmen dieser Arbeit eine Hardwarearchitektur für Field Programmable Gate Arrays (FPGAs) entwickelt, die leistungsfähig genug ist, um die aktuellen Anforderungen im Bereich von TZNs voll zu erfüllen. Gleichzeitig ist sie flexibel genug, um die unterschiedlichsten Funktionen zu realisieren. Auf Basis dieser Architektur wurden verschiedene Funktionen zur Anwendung im TZN entwickelt, prototypisch implementiert und evaluiert. Dazu gehören Funktionalitäten zur MAC Address Translation, dem Traffic Management und ein Multiprotocol Label Switching User Network Interface.

Ein weitere Funktionalität, die auf Grund ihrer Komplexität allerdings nur teilweise auf der entwickelten Hardwarearchitektur basiert, ist in der Lage, das sogenannte „Trust-by-Wire" in paketorientierte Kommunikationssysteme einzuführen. Damit wird unter anderem das Anbieten von Notrufen mittels des Voice-over-Internet Protocols unterstützt. Auch die Bekämpfung von Phishing und E-Mail-Spam kann so unterstützt werden.

Paketklassifizierung

Als Teilproblem der Datenverarbeitung darf die Paketklassifizierung nicht vernachlässigt werden, denn für jedes zu verarbeitende Paket muss festgestellt werden, auf welche Art und Weise es verarbeitet werden soll. Diese Problematik wird als Paketklassifizierungsproblem bezeichnet. Dessen Komplexität nimmt stark zu und wird von zwei Faktoren bestimmt. Einerseits steigt mit zunehmenden Übertragungsgeschwindigkeiten die Anzahl der Pakete an, die klassifiziert werden müssen. Andererseits steigt auch die Zahl der Dienste an, die angeboten werden sollen. Das erhöht die Größe der Klassifizierungsdatenbanken und folglich auch die Komplexität der Suche. Der Aufwand für die Klassifizierung jedes einzelnen Pakets nimmt zu. In konstanter Zeit müssen also immer mehr Pakete mit Hilfe immer größerer Klassifizierungstabellen klassifiziert werden. Bekannte Algorithmen und Architekturen sind zunehmend nicht in der Lage, mit steigenden Datenraten und steigenden Klassifizierungstabellengrößen Schritt zu halten. Die Entwicklungen im Bereich der Paketklassifizierung, die in dieser Arbeit vorgestellt werden, leisten einen Beitrag zur Lösung dieser Problematik durch die Entwicklung einer Hardware-basierten, evolvierbaren Hashfunktion, welche die Basis eines Paketklassifizierungsalgorithmus darstellt, der in einen evolvierbaren Paketklassifizierer integriert ist.

Um die durchgeführten Entwicklungen vorzustellen, ist diese Arbeit wie folgt aufgebaut:

- Kapitel 2 behandelt die relevanten Grundlagen. Dazu gehören die Grundlagen paketverarbeitender System, des Hashings und genetischer Algorithmen.
- In Kapitel 3 werden der aktuelle Stand der Technik sowie die relevanten Arbeiten sowohl im Bereich der paketverarbeitenden Systeme als auch der Paketklassifizierung betrachtet. Insbesondere auf das Hashing als Mittel der Paketklassifizierung wird genauer eingegangen.
- Auf die entwickelte Architektur für ein funktionales Modul, das als Basis für unterschiedliche Funktionalitäten im Bereich der Paketverarbeitung dienen kann, geht das 4. Kapitel ein. Darin werden außerdem mehrere Beispiele neuer Funktionen im TZN, ihre Hardwareimplementierung und deren Eigenschaften vorgestellt.
- Das 5. Kapitel widmet sich der Vorstellung der entwickelten evolvierbaren Hashfunktion und eines einfachen Hash-basierten Paketklassifizierers. Es werden unterschiedliche Architekturen vorgestellt und evaluiert.
- Bevor die Arbeit durch eine Zusammenfassung in Kapitel 7 abgeschlossen wird, werden in Kapitel 6 mehrere sinnvolle Optimierungen für die evolvierbare Hashfunktion vorgestellt. Mit diesen können sowohl die Leistungsfähigkeit als auch die Rechengeschwindigkeit des verwendeten evolutionären Algorithmus verbessert werden.

Abbildung 1 schlüsselt den Aufbau der Arbeit noch etwas genauer auf und hebt die wesentlichen vom Autor gemachten Beiträge hervor.

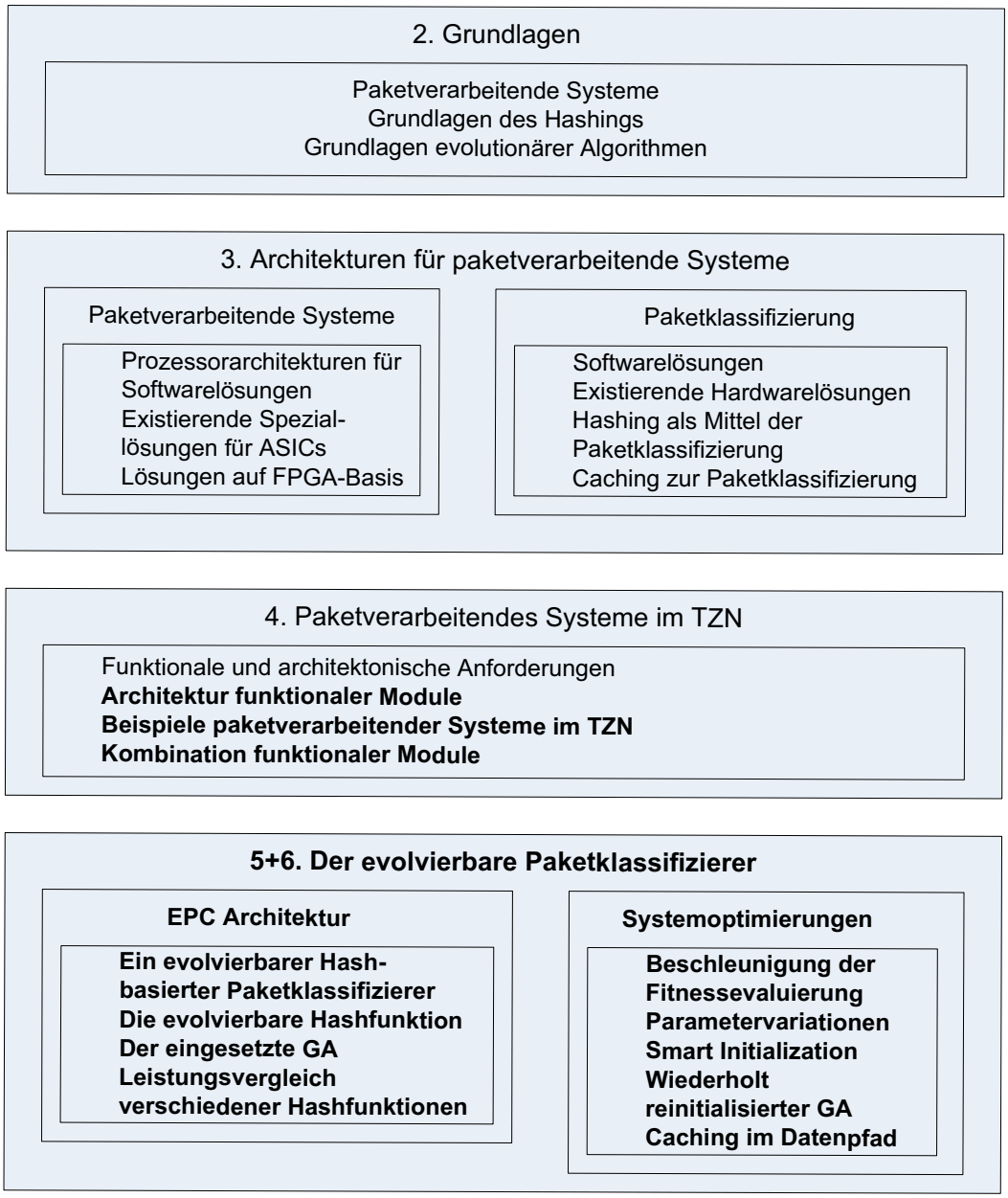


Abbildung 1 - Gliederung der Arbeit. Die Abschnitte mit den wesentlichen eigenen Beiträgen sind fett dargestellt.

2 Grundlagen

Diese Arbeit beschäftigt sich mit den zwei Teilproblemen paketverarbeitender Systeme. Das ist zum einen die Paketverarbeitung als solche und zum anderen die Klassifizierung von Paketen. Die Klassifizierung ist mit Hilfe von Hashfunktionen gelöst, die durch einen evolutionären Algorithmus optimiert werden. Im Folgenden wird deshalb auf die Grundlagen paketverarbeitender Systeme, die Grundlagen des Hashings und die Grundlagen evolutionärer Algorithmen eingegangen.

2.1 Paketverarbeitende Systeme

Moderne Kommunikationsnetze sind paketorientierte Netze. Im Gegensatz zu verbindungsorientierten Netzen, wie dem Telefonnetz, werden Informationen verbindungslos in Form von Datenpaketen übertragen. Demzufolge sind alle Systeme auf einer solchen Kommunikationsstrecke paketverarbeitende Systeme. Der Nutzen solcher Systeme ist maßgeblich die Menge und die Qualität der Dienste bestimmt, die angeboten werden können. In der modernen Datenkommunikation spielt deshalb der Begriff der Quality-of-Service (QoS) eine zunehmend wichtigere Rolle. Je mehr Dienste angeboten werden, desto größer ist der Nutzen sowohl für Kunden als auch für ISPs. Gleiches gilt für die Qualität der Dienste. Je besser sie umgesetzt werden, desto größer ist die QoS. Die Aufgabe eines paketverarbeitenden Systems ist einfach zu umreißen: Verarbeite (manipuliere, versende, speichere, ordne) alle ankommenden Pakete auf eine adäquate Weise.

Die bekanntesten Vertreter paketverarbeitender Systeme sind Router. Sie bilden die Knotenpunkte der Kommunikation in Weitverkehrsnetzen (Wide Area Networks – WANs), die regionale, nationale und internationale Netzwerke miteinander verbinden [Hal05]. Die Hauptaufgabe eines Routers ist es, alle eingehenden Datenpakete an einen von mehreren physikalischen Ausgangports weiterzuleiten. Damit wird ein effizienter und schneller Punkt-zu-Punkt Datentransport über große Strecken zwischen zwei Kommunikationspartnern ermöglicht. Router transportieren Datenpakete zwischen Netzwerken [Awe99]. Datenpakete bestehen im Prinzip aus zwei Teilen: dem Paketheader und den tatsächlichen Nutzdaten. Im einfachsten Fall trifft ein Router die Entscheidung, welchem Ausgang ein Paket zugewiesen werden soll, anhand bestimmter Felder des Paketheaders. Bei komplexeren Aufgaben ist es auch möglich, dass auch Teile der Nutzdaten in die Entscheidungsfindung einbezogen werden. Zusammenfassend ist festzustellen, dass für jedes eingehende Paket Berechnungen durchgeführt werden müssen, um es angemessen behandeln zu können; sei es die Manipulation des Pakets selbst oder das Finden des richtigen Ausgangsports. Die Problematik, jedes einzelne Paket korrekt einzuordnen und behandeln zu müssen, wird als Paketklassifizierungsproblem bezeichnet [Var05].

2.1.1 Netzwerktheoretische Grundlagen

2.1.1.1 ISO-OSI Referenzmodell

In der Kommunikationstechnik haben sich für verschiedene Anwendungsbereiche die unterschiedlichsten Kommunikationsprotokolle etabliert. Das birgt die Gefahr von Inkompatibilitäten bei der Zusammenarbeit verschiedener Anwendungen auf verschiedenen Rechnersystemen in verschiedenen Netzwerken. Um solche Probleme zu verhindern, hat die International Organization for Standardization (ISO) 1983 das Open Systems Interconnection (OSI) Referenzmodel [Zim80] standardisiert. Dieses Modell definiert sieben verschiedene Kommunikationsschichten (Abbildung 2). Jede Schicht bietet der übergeordneten Schicht den Dienst des Datentransportes an. Auf der oberen Ebene nutzt ein Kommunikationsprotokoll diesen Dienst, um Daten an einen Kommunikationspartner zu übertragen. Ein Kommunikationsprotokoll umfasst eine Menge von Regeln, die die Übermittlung von Daten zwischen zwei oder mehreren Entitäten (z. B. Computern oder Prozessen) im gesamten Netz steuert [JW02]. Dazu definiert das Protokoll Operationen und den Aufbau von Nachrichten [PD03].

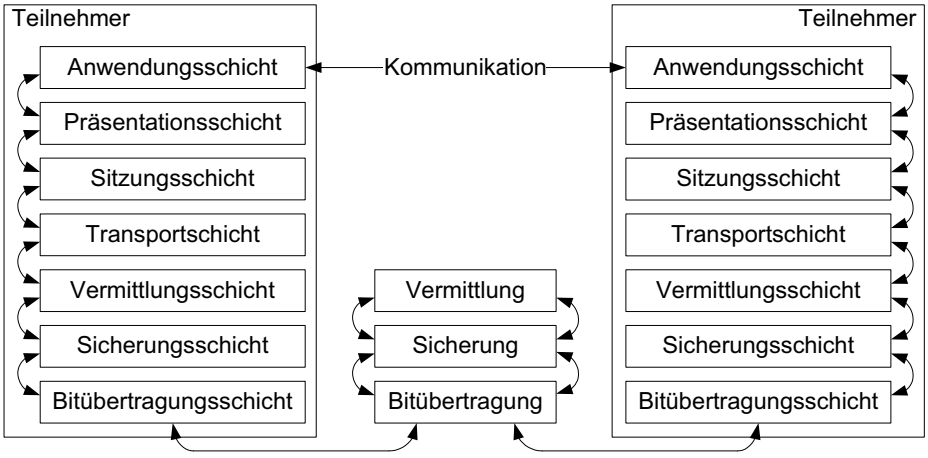


Abbildung 2 - Kommunikationsbeziehung zwischen zwei Teilnehmern über einen dazwischen liegenden Router aus Sicht des ISO-OSI Schichtenmodells.

Das Protokoll einer Schicht übergibt seine Daten an die darunter liegende Schicht, welche dann die übernommenen Informationen in das dort gültige Protokoll einbettet. Das heißt, den Daten werden sogenannte Headerinformationen voran- und Trailerinformationen nachgestellt. Eine Kommunikation zwischen zwei Entitäten findet immer auf der gleichen horizontalen Ebene statt. Beispielsweise kommunizieren die Transportschichten zweier Teilnehmer miteinander. Dies geschieht durch die Dienstnutzung der untergeordneten Protokolle von Netzwerk-, Vermittlungs- und Bitübertragungsschicht. Die einzelnen Schichten haben verschiedene Aufgaben:

- Die Bitübertragungsschicht (engl. *physical layer*) steuert die Übertragung von einzelnen Datenbits über einen Kommunikationskanal. Das können z. B.

Lichtimpulse bei optischer Datenübertragung oder Spannungen bei elektrischer Datenübertragung sein.

- Die Sicherungsschicht (engl. *data link layer*) hat die Aufgabe, einen Frame, welcher eine Menge von Bits darstellt, an einen anderen Teilnehmer zu übertragen. Beide Teilnehmer sind über das physikalische Medium direkt miteinander verbunden. Dabei können Mechanismen zur Flusskontrolle (z. B. Stop-and-Wait), Fehlererkennung bzw. -korrektur (z. B. Cyclic Redundancy Check – CRC [IEE05]) oder Sequenzierung (z. B. beim Asynchronous Transfer Mode – ATM [MS98]) zum Einsatz kommen.
- Die Vermittlungsschicht (engl. *network layer*) hat die Aufgabe, für einen Nutzer ihres Dienstes einen Kanal im Netzwerk bereitzustellen. Die Funktionalitäten der Vermittlungsschicht umfassen vor allem die Wegfindung (Routing) und das Weiterleiten von Datenpaketen zwischen Netzwerken.
- Die Transportschicht (engl. *transport layer*) hat die Aufgabe, eine verlässliche Ende-zu-Ende Verbindung zwischen zwei Teilnehmern bereitzustellen. Ihre Funktionen umfassen Flusskontrolle, Fehlererkennung und –korrektur sowie die Sicherstellung, dass Reihenfolgen von Paketen eingehalten werden.
- Die Sitzungsschicht (engl. *session layer*) befasst sich mit der Koordinierung und Synchronisierung eines Dialoges zweier Teilnehmer. Sie legt unter anderem fest, ob der Kommunikationskanal im Simplex-, Halbduplex- oder Vollduplexmodus betrieben wird.
- Die Präsentationsschicht (engl. *presentation layer*) hat Aufgaben im Bereich der Darstellung von Informationen. Dazu gehört z. B., ob ein Ganzzahlwert durch 16 oder 32 Bit darzustellen ist oder ob eine Übertragung von Daten mit dem höchst- oder niederwertigsten Bit zuerst zu erfolgen hat.
- Die Anwendungsschicht (engl. *application layer*) als höchste Schicht bietet Anwendungen die Möglichkeit, sich mit anderen Anwendungen auf entfernten Systemen zu verbinden und Daten auszutauschen.

2.1.1.2 Netzwerkprotokolle

Die Internet-Architektur, die nach ihren am häufigsten verwendeten Protokollen auch TCP/IP-Architektur genannt wird, bildet das ISO-OSI Referenzmodell nicht exakt ab. Die oberen drei Schichten des ISO-OSI Modells sind hier zur Anwendungsschicht zusammengefasst. Transport-, Vermittlungs- und Sicherungsschicht werden unter anderem durch die in Abbildung 3 dargestellten Protokolle implementiert.

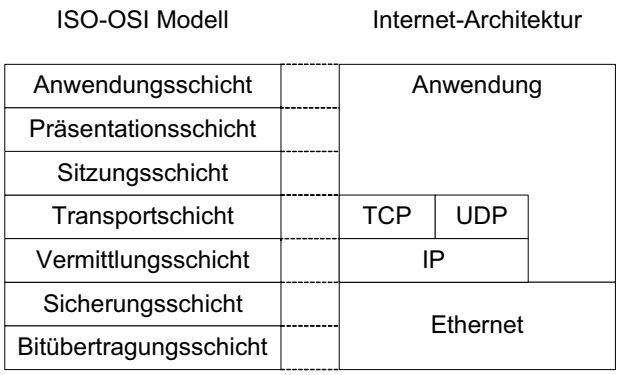


Abbildung 3 - Abbildung des ISO-OSI Modells auf die Internet-Architektur mit Beispielprotokollen.

Wie in Abbildung 4 dargestellt, bestehen alle Protokolle aus einem Nutzdatenteil und einem protokolleigenen Header. Je nachdem auf welcher Protokollschicht man sich bewegt, werden die Datenstrukturen unterschiedlich bezeichnet. Auf Schicht 4 (TCP/UDP) spricht man von Datagrammen. Auf Schicht 3 (IP) wird von Datenpaketen gesprochen. Die Bezeichnung für Daten der Schicht 2 (Ethernet) ist Datenframe. ATM ist als Sicherungsschichtprotokoll heute nahezu bedeutungslos [SGS05].

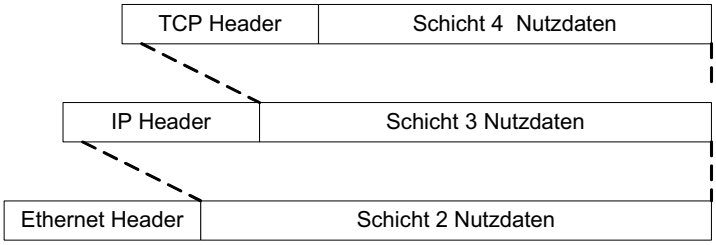


Abbildung 4 - Aufbau der verschiedenen Protokollschichten der Internet-Architektur.

Die wichtigsten und auch für diese Arbeit relevanten Protokolle sind TCP, IP und Ethernet. Weil darum ein grundlegendes Verständnis essentiell ist, sollen sie im Folgenden näher beschrieben werden.

2.1.1.3 Ethernet

Das Ethernet ist die am weitesten verbreitete Technologie zur Implementierung der Sicherungsschicht [DIX82]. Neben der Sicherungsschicht implementiert Ethernet auch die Bitübertragungsschicht. Ethernet-basierte Local Area Networks (LANs) bestehen aus Netzwerkknoten und dem sie verbindenden Übertragungsmedium. Im LAN werden Datenframes übertragen, die zwischen 64 und 1518 Byte lang sein können. Bei den Knoten unterscheidet man das Data Terminal Equipment und das Data Communication Equipment. Das Data Terminal Equipment kann Quelle oder Senke von Datenframes sein. Beispiele sind

Personalcomputer, Workstations oder Drucker. Zum Data Communication Equipment gehören Netzwerkgeräte, die Daten empfangen und weiter versenden. Das sind beispielsweise Repeater, Hubs oder Switches. Als Übertragungsmedium sind Kupferkabel (Twisted Pair, Coaxial) und Lichtwellenleiter spezifiziert. Zurzeit unterstützt Ethernet vier unterschiedliche Datenraten:

- 10 MBit/s – 10BaseT Ethernet (IEEE 802.3)
- 100 MBit/s – Fast Ethernet (IEEE 802.3u)
- 1000 MBit/s – Gigabit Ethernet (IEEE 802.3z)
- 19 GBit/s – 10 Gbps Ethernet (IEEE 802.3ae)

LANs können als Bus- oder Sterntopologie verbaut sein, wobei moderne Netze ausschließlich die Sterntopologie verwenden. Zur Adressierung von Netzwerkgeräten werden in Ethernetframes MAC-Adressen (Medium Access Control) verwendet (Abbildung 5). MAC-Adressen sind 48 Bit breit. Jede Ethernet-fähige Netzwerkkarte besitzt eine eigene MAC-Adresse, die weltweit einmalig ist. Eine Empfänger-MAC (Destination-MAC – DST-MAC) und eine Absender-MAC (Source-MAC – SRC-MAC) definieren die Partner einer Punkt-zu-Punkt Verbindung im Ethernet. Um in einem Ethernet logische (virtuelle) Netzwerke mit unterschiedlichen Nutzern zu bilden, können Virtual LANs (VLANs) verwendet werden [IEE06a]. In einem Ethernet, das VLANs unterstützt, folgt auf SRC- und DST-MAC im Header ein VLAN-Tag, welches insgesamt 4 Byte umfasst.

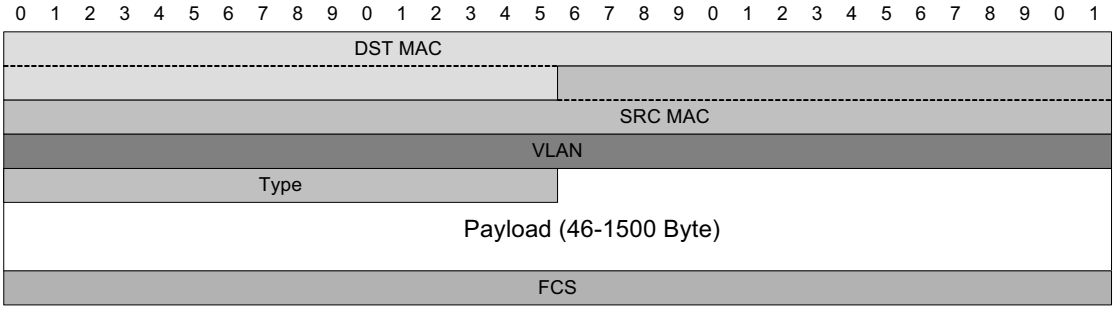


Abbildung 5 - Aufbau eines Ethernetframes.

2.1.1.4 Internet Protocol

Das Internet Protocol (IP) [Usc81b], hervorgegangen aus dem Advanced Research Projects Agency Network (ARPANet) [FC74], ist ein Protokoll der Vermittlungsschicht. Als solches enthält es Funktionalitäten zur Weiterleitung von Datenpaketen und vor allem zur Wegfindung (Routing). Daten werden zwischen Netzwerkgeräten ausgetauscht. Dies geschieht auf Basis von IP-Adressen. Eine IP-Adresse identifiziert im Allgemeinen ein Netzwerkgerät eindeutig. Bestimmte Netzwerkgeräte wie Router können Mitglieder mehrerer Netzwerke sein. Folglich besitzen sie in jedem dieser Netzwerke auch eine Netzwerkadresse. In diesem Fall identifiziert die IP-Adresse eher die Netzwerkschnittstelle als das Netzwerkgerät [PD03]. In der weit

verbreiteten Version 4 des Protokolls (IPv4) sind dies 32 Bit breite Adressen, bei denen jedes Byte in dezimaler Form mit einem Punkt getrennt dargestellt wird (z. B. 139.30.201.5). Damit können theoretisch ca. 4 Milliarden unterschiedliche Netzwerkgeräte adressiert werden. IP-Pakete der Version 4 bestehen aus einem mindestens 20 Byte langen Header, der unter anderem Informationen zum gekapselten Transportschichtprotokoll und zur Fragmentierung von Paketen enthält (Abbildung 6). Er beinhaltet außerdem SRC- und DST-IP zur Adressierung von Datenquelle und –senke. Ein IP-Paket ist zwischen 46 und 1500 Byte lang.

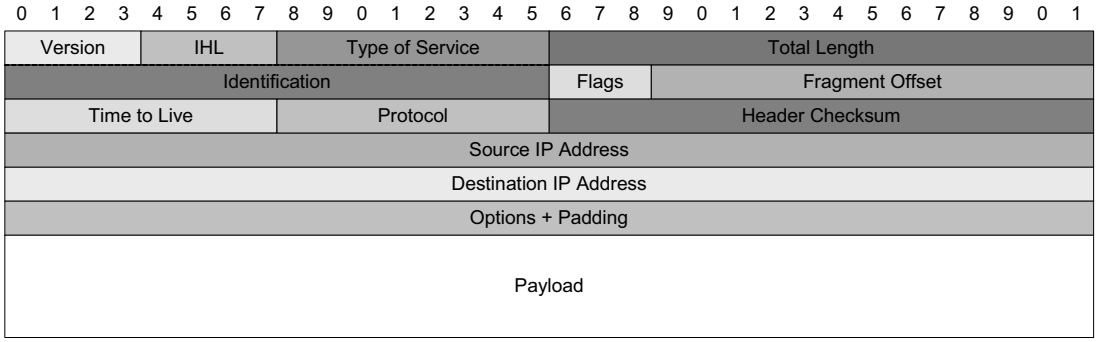


Abbildung 6 - Aufbau eines IPv4-Pakets.

Da die Zahl von 4 Milliarden bei IPv4 für die zukünftig zu erwartende Teilnehmerzahl unzureichend ist, wird zunehmend auf das IP-Protokoll der Version 6 (IPv6) umgestellt. Bei IPv6 stehen jedem Gerät 128 Bit breite IP-Adressen zur Verfügung. Das sind mehr als $3{,}4 \cdot 10^{38}$ Adressen. Es gibt unterschiedliche Protokolle, die das Routing implementieren. Dazu gehören z. B. RIP [Hed88], OSPF [Moy94] und BGP [Rek+06].

2.1.1.5 Transmission Control Protocol

Das Transmission Control Protocol (TCP) [Usc81a] bildet in der TCP/IP-Architektur die Transportschicht ab. Es handelt sich um ein verbindungsorientiertes Protokoll zur Übertragung von Datenströmen zwischen Anwendungen. Das Protokoll stellt den Transport der Daten von einem Teilnehmer zum anderen sicher. Sollten Datagramme bei der Übertragung verloren gehen, veranlasst das Protokoll eine erneute Übertragung. Dies wird durch die Vergabe von Sequenznummern erreicht. Kommt ein Datagramm mit einer bestimmten Sequenznummer nicht beim Empfänger an bzw. erreicht die Empfangsbestätigung den Sender nicht, findet ein erneutes Versenden statt. Auf der Ebene von TCP kommunizieren nicht die Netzwerkgeräte miteinadner, da auf einem Gerät mehrere Netzwerkanwendungen gleichzeitig laufen können. Deshalb reicht die Identifikation über die IP-Adresse nicht aus. Für das TCP-Protokoll stehen sogenannte Source- und Destination-Ports zur Verfügung. Dabei handelt es sich im Unterschied zu physikalischen Ports von Netzwerkgeräten um logische Ports als Teil des Protokolls. Beispielsweise nutzen Internetbrowser den Port 0x80 zur Kommunikation über das

Anwendungsprotokoll HTTP. Mittels des Tupels aus {SRC-IP, DST-IP, SRC-Port, DST-Port} ist die Kommunikationsverbindung zwischen zwei Anwendungen eindeutig identifizierbar.

TCP ist ein zustandsbehaftetes Protokoll. Die Kommunikation verläuft über eine Initialisierungsphase, die eigentliche Datenübertragung und den Abbau der Kommunikationsverbindung.

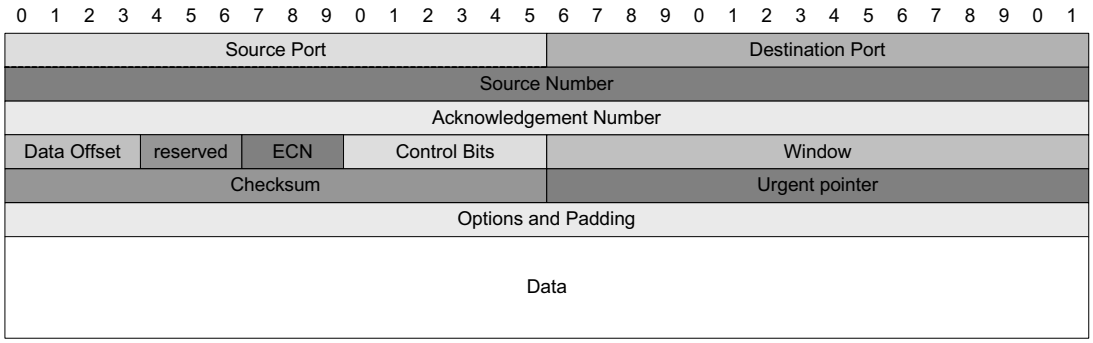


Abbildung 7 - Aufbau eines TCP-Datagramms.

2.1.2 Paketverarbeitung

Bei der Verarbeitung von Paketen können grundsätzlich zwei Klassen von Operationen auf Paketen unterschieden werden: Datenpfadoperationen und Kontrollpfadoperationen [Wil01].

Kontrollpfadoperationen benötigen im Allgemeinen weniger Rechenleistung als Datenpfadoperationen. Sie treten seltener auf, sind zeitlich zumeist unkritisch und haben keine Verbindung zum Datenpfad. Beispiele für Kontrollpfadoperationen sind die Konfiguration und die Verwaltung des paketverarbeitenden Systems, die Verarbeitung bestimmter im Datenverkehr nur sehr selten vorkommender Protokolle oder die Behandlung von Ausnahmen und Sonderfällen. Diese erfordern zwar unter Umständen spezielle, rechenintensive Maßnahmen, kommen aber so selten vor, dass deren Verarbeitung nicht ins Gewicht fällt.

Datenpfadoperationen haben weitaus höhere Anforderungen an die Leistung eines paketverarbeitenden Systems. Im Datenpfad müssen die Pakete in großer Zahl und mit höchster Geschwindigkeit verarbeitet werden. Die Anforderungen lauten demzufolge Verarbeitung mit niedrigster Latenz und Realisierung höchster Datendurchsätze (wirespeed).

Aus diesen Anforderungen folgt, dass Datenpfadoperationen für eine Hardwareimplementierung prädestiniert sind. Auf der anderen Seite hat der Kontrollpfad geringe zeitliche jedoch teilweise komplexe funktionale Anforderungen. Aus diesem Grund ist er bevorzugt in Software zu realisieren. Die Realisierung kann auf Netzwerkprozessoren oder Universalprozessoren geschehen.

Auch unterschiedliche Netzwerke und Systembereiche haben verschiedene Anforderungen an paketverarbeitende Systeme. Man kann die Paketverarbeitung grundsätzlich im Weitverkehrsnetz (Wide Area Network – WAN), im lokalen Netz (Local Area Network – LAN)

und im Teilnehmerzugangsbereich (Access Network – AN) unterscheiden. Abbildung 8 illustriert die Zusammenhänge zwischen den Netzen.

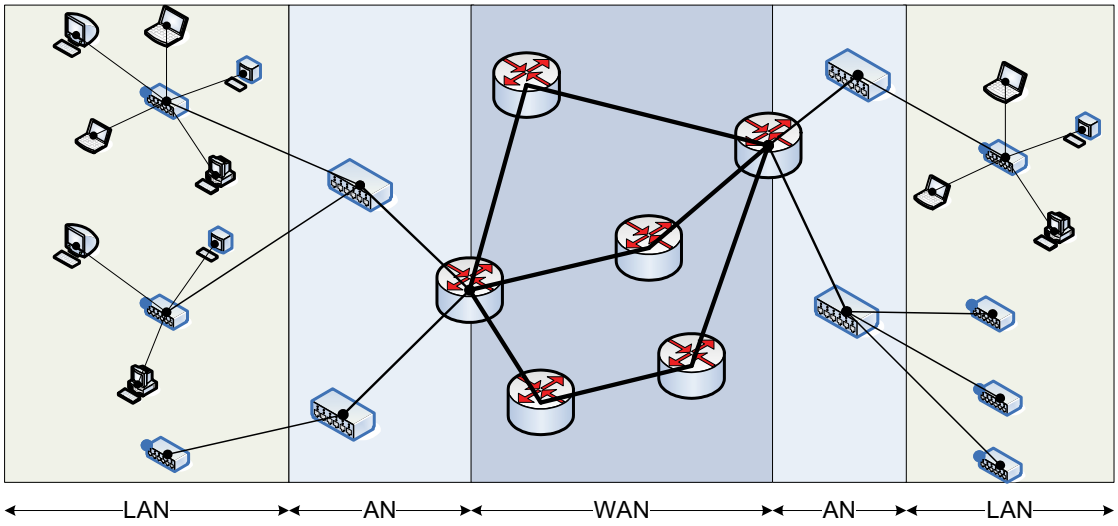


Abbildung 8 - Netzwerkstruktur des Internets.

2.1.2.1 Paketverarbeitung im LAN

In lokalen Netzwerken sind räumlich nahe Kommunikationsteilnehmer miteinander verbunden. Die am weitesten verbreitete Technologie zur Verbindung ist das Ethernet, welches auf ein geteiltes Übertragungsmedium setzt. Als Zugriffsverfahren in einem geteilten Medium wird CSMA/CD (Carrier Sense Multipla Access with Collision Detection) verwendet. Dabei kann es durch die Teilung des Mediums zu Kollisionen durch gleichzeitigen Zugriff kommen, weshalb maximal 1024 Teilnehmer angeschlossen sein dürfen. Moderne Netze nutzen „switched LANs“. Bei diesen sind alle Teilnehmer jeweils einzeln an Switches angeschlossen, wodurch erreicht wird, dass das Übertragungsmedium nicht mehr geteilt werden muss. In switched LANs ist deshalb eine kollisionsfreie Datenübertragung möglich [JW02]. Switches haben dabei die Aufgabe, alle ankommende Datenframes ausschließlich an den richtigen Ausgang weiter zu versenden. Das geschieht auf Basis der DST-MAC des Ethernetheaders. Paketverarbeitung im LAN bedeutet folglich die Klassifizierung des Pakets auf Basis seiner DST-MAC und das Versenden desselben an den korrekten pysikalischen Ausgangsport.

2.1.2.2 Paketverarbeitung im Teilnehmerzugangsbereich

Der Teilnehmerzugangsbereich stellt den Übergang vom LAN zum WAN dar. An dieser Stelle aggregieren Internet Service Provider (ISPs) die lokalen Netzwerke von Unternehmen und die Breit- oder Schmalbandanschlüsse privater Haushalte. DSL-Anschlüsse von Privatnutzern werden beispielsweise mittels DSL Access Multiplexern (DSLAMs) terminiert, aggregiert und über einen MultiGigabit Anschluss dem WAN zugeführt (Abbildung 9).

DSLAMs sind verhältnismäßig große Systeme, die mehrere Tausend Nutzer mit dem Breitbandnetz des ISPs verbinden.

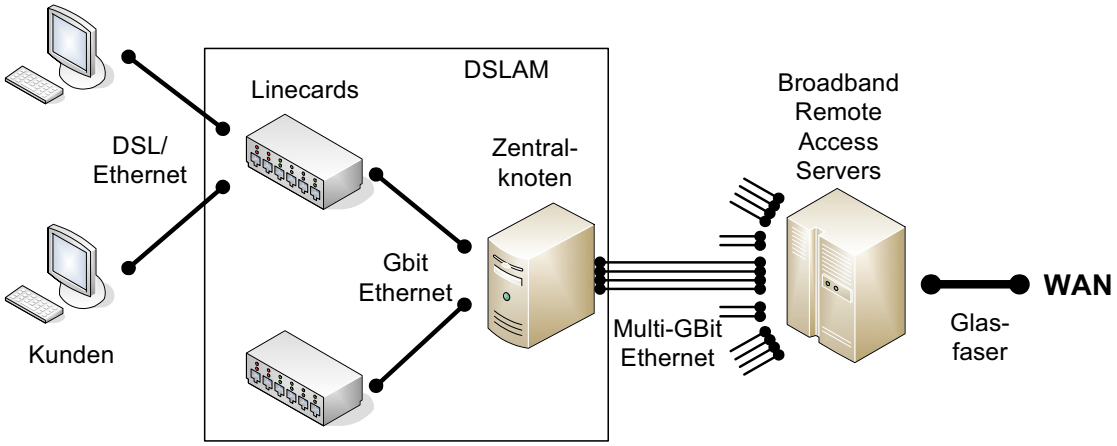


Abbildung 9 - Aufbau eines Teilnehmerzugagnsnetzwerks.

Für neuere optische Netzwerke übernimmt oft das GPON (Gigabit-capable Passive Optical Network) diese Funktion [ITU03]. Andere optische Technologien im Teilnehmerzugang sind z. B. BPON (Broadband PON [ITU05], (G)EPON (Ethernet PON - IEEE 802.3ah [IEE04]) oder aktive optischen Netze. Aus dem Übergang vom LAN in das WAN ergeben sich verschiedene spezielle Aufgaben und Funktionen, die im Teilnehmerzugangsnetz (TZN) ausgeführt bzw. unterstützt werden müssen. Es ist in aktuellen und auch zukünftigen Teilnehmerzugangssystemen eine Vielzahl von nutzerspezifischen Funktionen zu unterstützen. Mit dem Trend, Ethernet auch für die Erste Meile zu verwenden, kommen auch Aufgaben im Bereich der Quality-of-Service hinzu, wie Metering, Policing und Scheduling [DSL03b]. In diesen Bereich fallen Funktionalitäten wie die MAC Address Translation [KWD+06], das Traffic Management [KWD+06] und das Multi Protocol Label Switching (MPLS) [WKD+06].

Zusammen mit der Erweiterung des funktionalen Spektrums erhöhen sich allerdings auch die Datendurchsätze und die Zahl der angeschlossenen Teilnehmer im AN rapide. Einerseits steigt die Bandbreite eines einzelnen angeschlossenen Haushalts (VDSL2 ermöglicht bereits bis zu 100 MBit/s Datendurchsatz). Zum anderen steigt auch die Anzahl der Teilnehmer, die an einen physikalischen Port angeschlossen werden. VDSL2 ermöglicht bei 100 MBit/s eine zufriedenstellende Vernetzung einer kleinen Firma über einen einzigen Anschluss. DSLAMs haben zurzeit eine Bandbreite von 4 GBit/s auf Providerseite. Die Anforderungen werden sich allerdings aus den oben genannten Gründen in Zukunft weiter erhöhen.

Ein vergrößertes funktionales Spektrum und steigende Datenraten sind zwei sich widersprechende Designziele: Hohe Durchsätze und Datenraten sind zu erzielen, wenn man schnelle Hardwarelösungen einsetzt, die bestimmte Aufgabenstellungen sehr effizient lösen können. Funktionale Diversifikation ist jedoch in einem solchen Umfeld nur sehr schwer zu erreichen. Sollen Systeme flexibel sein, um sie für neue Anforderungen anzupassen und ein

breites funktionales Spektrum abzudecken, bieten sich auf den ersten Blick Softwarelösungen auf der Basis von Mikro- oder speziellen Netzwerkprozessoren an. Damit ist, grob gesprochen, praktisch jede Funktion realisierbar. Allein die Performance ist nicht für jeden Anwendungsbereich ausreichend.

2.1.2.3 Paketverarbeitung im WAN

Die vorherrschenden Weitverkehrsnetze sind Internetbackbones von Internet Service Providern. Es handelt sich um Internetworks. Internetworks sind eine beliebige Anzahl miteinander verbundener heterogener Netzwerke [PD03]. In diesen werden Datenpakete geroutet. Das heißt, anhand der Zieladresse des IP-Headers jedes Pakets werden an Knotenpunkten (Routern) Entscheidungen über die Weiterleitung der Pakete getroffen. Es kommen verschiedene Routingprotokolle wie RIP oder BGP zum Einsatz. In moderneren Netzen wird das IP-Routing aber auch schon durch das Multi Protocol Label Switching (MPLS) ersetzt [RVC01, RTF+01]. Hierbei wird jedem Paket, das ein MPLS-fähiges Netzwerk betritt, ein Labelstack zugeordnet. Dieser aus beliebig vielen 24-Bit Labels bestehende Stack definiert, wie das MPLS-fähige Netzwerk zu durchlaufen ist. In jedem Fall treffen Router ihre Entscheidungen immer, nachdem eine Durchführungsregel (entweder für eine bestimmte IP-Adresse oder für ein bestimmtes MPLS-Label) gefunden wurde. Dieses Routing ist die Hauptaufgabe eines paketverarbeitenden Systems im WAN. Dazu muss in jedem Router auf der Übertragungsstrecke eine Entscheidung getroffen werden, wie das Paket weiterzuleiten ist. Das heißt, jedes Paket ist zu bearbeiten. Aufgrund sehr hoher Datenraten von mehreren Gbit/s müssen die Systeme überaus leistungsstark sein. Moderne Router der Firma Cisco – als Beispiel sei der Cisco 12012 Router genannt – unterstützen Datenraten von 60 Gbit/s und mehr [Cis08]. Neben dem Routing muss im WAN auch die Bereitstellung bestimmter Dienstgüten für verschiedene Verkehrsarten unterstützt werden. Das bezeichnet man als Quality-of-Service (QoS). Beispielsweise ist es sinnvoll, Voice-over-IP (VoIP) Daten mit sehr hoher Priorität zu versenden, um die Paketlatenz zu minimieren. Dafür muss in den entsprechenden Geräten der VoIP-Verkehr klassifiziert werden.

2.1.3 Paketklassifizierung

Ein ganz entscheidender Faktor für die Performance eines jeden paketverarbeitenden Systems ist die Leistung des Paketklassifizierers. Dies gilt sowohl im LAN, im WAN wie auch im Teilnehmerzugangsbereich. In allen Fällen muss jedes ankommende Datenpaket überprüft werden. Das Ergebnis dieser Überprüfung oder Klassifizierung bestimmter Paketinformationen definiert, wie jedes Paket verarbeitet wird. Man spricht dabei von der Zuordnung zu Flows. Die Aufgabe, jedes einzelne Paket überprüfen zu müssen, wird als Paketklassifizierungsproblem bezeichnet.

2.1.3.1 Paketklassifizierungsproblem

Wie oben bereits erwähnt, ist es notwendig, jedes einzelne ankommende Datenpaket zu verarbeiten. Das können beispielsweise die Wahl eines Ausgangsports in einem Router oder das Feststellen der zugehörigen Priorität für ein Datenpaket sein. Die Bandbreiten in der Datenkommunikation wachsen rasant. Abbildung 10 stellt das Verkehrsaufkommen am größten europäischen Internetkontenpunkt DE-CIX in Frankfurt dar.

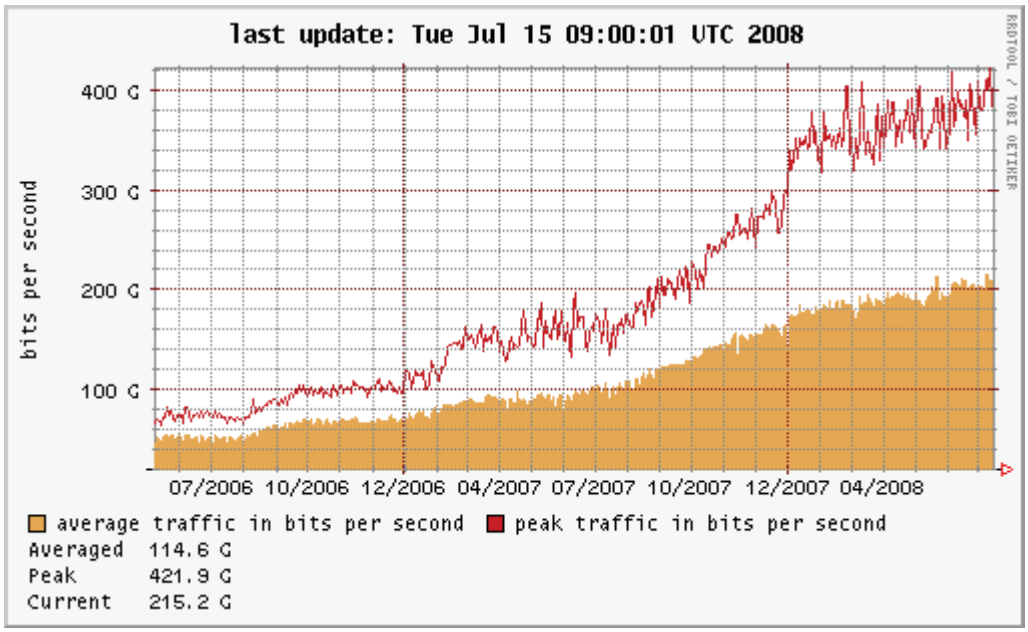


Abbildung 10 - Verkehrsaufkommen am größten europäischen Internetknotenpunkt DE-CIX in Frankfurt/Main (Quelle: [Ger08]).

Deshalb ist es notwendig, immer mehr Pakete in immer kürzerer Zeit zu klassifizieren. Auch die Größe von zu durchsuchenden Datenbanken steigt rasant an. Abbildung 11 illustriert die Entwicklung der Einträge in eine BGP-Forwardingtabelle eines Routers. Wie zu erkennen ist, hat die Anzahl der Einträge allein von Januar 2006 bis Januar 2008 von ca. 170.000 auf fast 250.000 Einträge zugenommen. Die Zunahme der Größe der Routingtabellen zwischen 1994 und 2000 wird in [Hus01] einer gründlichen Untersuchung unterzogen.

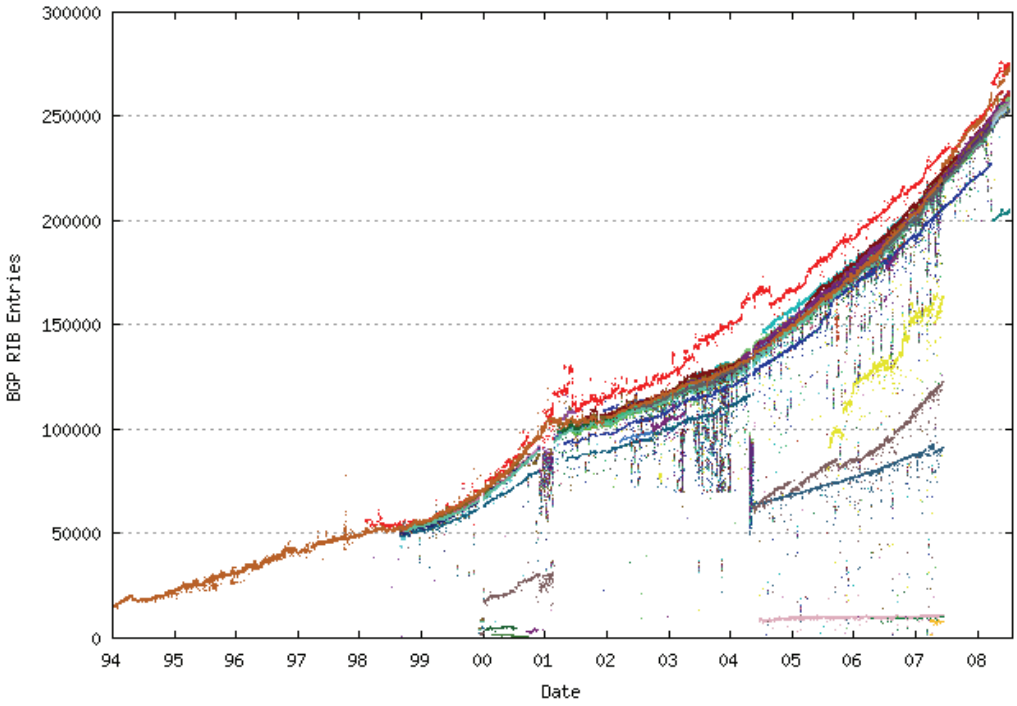


Abbildung 11 - Darstellung des Wachstums der Anzahl der Einträge in verschiedenen BGP-Routingtabellen (Quelle: [Hus08]).

Die Klassifizierung der Pakete kann als Suchproblem aufgefasst werden. Abbildung 12 stellt den Vorgang der Paketklassifizierung schematisch dar. Für jedes ankommende Paket ist ein Schlüssel zu extrahieren. Anhand dieses Schlüssels ist in einer Datenbank bzw. einem Speicher eine Regel oder zugeordnete Information zu finden. Diese definiert, wie mit dem Paket zu verfahren ist. Die Regel wird zusammen mit dem Paket einem Modul übergeben, das eine bestimmte Funktionalität realisiert. Anhand der Regel manipuliert, speichert, priorisiert oder versendet das Modul das Paket. Die Entscheidung, zu welcher Klasse (welchem Flow) ein Datenpaket gehört, wird im Allgemeinen anhand der Paketheader gefällt. Diese umfassen Informationen der Header der ISO-OSI Netzwerkschichten 2 bis 4, wozu beispielsweise die Source- und die Destination MAC-Adresse, die Source- und Destination IP-Adresse und das Source- und Destination TCP-Port gehören. In einigen Fällen ist auch der Inhalt der Nutzdaten zu inspizieren, um eine Klassifizierung des Pakets zu ermöglichen.

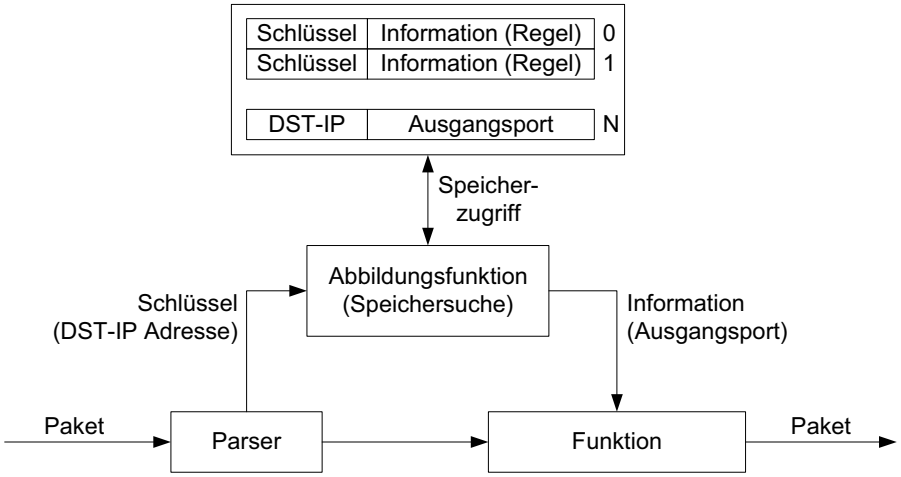


Abbildung 12 - Schematisch Darstellung der Paketklassifizierung.

Bei der Paketklassifizierung sind vier Trefferarten zu unterscheiden:

Exact Match: Ein gesuchter Schlüssel ist identisch in der Suchdatenbank abgelegt. Bei der Suche nach IP-Adressen tritt dies bei einer hundertprozentigen Übereinstimmung auf (Beispiel: $139.30.201.30 = 139.30.201.30$). Bei Switchingentscheidungen im LAN und der Teilnehmeridentifizierung im AN trifft das zu. Im LAN sind ausschließlich exakte MAC-Adressen auszuwerten [Var05]. Auch im AN, wo vorwiegend Nutzer und Nutzergruppen identifiziert werden müssen, sind Exact Matches zu ermitteln, um entsprechenden Paketverarbeitungsaufgaben gerecht zu werden.

Prefix Match: Der Anfang bzw. der vordere Teil eines Schlüssels wird als Präfix bezeichnet. Ein Prefix Match tritt auf, wenn der Präfix eines Schlüssels mit einem gespeicherten Eintrag übereinstimmt (Beispiel: $139.30.201.30/16 = 139.30.0.0/16$ (nur die ersten 16 Bit sind ergebnisrelevant)). Prefix Matches treten im Zusammenhang mit der Bestimmung des Zielnetzwerks mittels Subnetzmasken auf. IP-Routing erfolgt immer nach dem „Longest Matching Prefix“ (LMP)[1]. Prefix Matches kommen seit der Einführung des Classless Inter-domain Routings (CIDR) [Rek95]) zum Einsatz.

Range Match: Der Wert eines Teils des Schlüssels liegt in einem bestimmten Bereich des gespeicherten Eintrags. Beispiel: $139.30.201.30 = 139.30.x.30$. Range Matches kommen oft in Firewalls vor, die anhand von Portadressen der Transportschicht filtern. Dabei werden beispielsweise ganze Portbereiche gesperrt, also nicht zugelassen [SR06].

Regular Expression: Der Schlüssel stimmt mit einem regulären gespeicherten Ausdruck überein. Beispiel: Wenn r_1 und r_2 reguläre Ausdrücke sind, ist $r_1|r_2$ auch ein regulärer Ausdruck. Stimmt eine Zeichenkette mit r_1 oder r_2 überein, so stimmt sie auch mit $r_1|r_2$ überein. Eine gute

[1] Stimmen mehrere Präfixe unterschiedlicher Länge mit gespeicherten Einträgen überein, ist nur der längste Präfix entscheidend.

Einführung dazu ist z. B. in [SP01] zu finden. Network Intrusion Detection Systems (NIDS) nutzen unter anderem Regular Expressions.

Abhängig davon, ob zur Paketklassifizierung nur einer oder mehrere Parameter (z. B. Headerfelder) benötigt werden, unterscheidet man die einfache Paketklassifizierung und die Paketklassifizierung auf einem Tupel von Feldern.

2.1.3.2 Einfache Paketklassifizierung

Unter einfacher Paketklassifizierung ist die Klassifizierung auf einem einzelnen Feld eines Protokollheaders. Die Wegfindungsentscheidung in Routern ist ein Beispiel für die einfache Paketklassifizierung. Router entscheiden, welcher Ausgangsport einem eingehenden Paket zugewiesen wird. Bei klassischen Routingprotokollen, wie dem Border Gateway Protocol (BGP) [Rek+06], die momentan weltweit eingesetzt werden, trifft der verantwortliche Algorithmus diese Entscheidung ausschließlich anhand der Destination IP-Adresse eines jeden Pakets. Das Klassifizierungsproblem ist in diesem Fall auf das einfache Suchen nach einer Regel mittels eines singulären Schlüssels (DST-IP) reduziert. Eine Datenbank muss möglichst effizient durchsucht und eine Regel für den aktuellen Schlüssel gefunden werden.

2.1.3.3 Paketklassifizierung auf einem Tupel

Im TZN und anderen Netzwerksystemen, die Datenpakete verschiedenen Flows zuordnen, um beispielsweise QoS-Funktionalitäten zu realisieren, findet eine Klassifizierung von Paketen auf einem Tupel von Feldern statt. Die entwickelten Funktionalitäten zur MAC-Address-Translation und dem Traffic Management in [KWD+06] und dem Multi Protocoll Label Switching (MPLS) in [WKD+06] nutzen mehrere Headerfelder zur Flowzuordnung. Die Kombination der Funktionalitäten, vorgestellt in [WKT+06], ist konfigurierbar. Es können sieben verschiedene Headerfelder (SRC- und DST-MAC, SRC- und DST IP, zwei VLAN-Tags, sowie das DSCP[2]-Feld des IP-Headers) als Schlüssel verwendet werden. In Teilnehmerzugangs-systemen dient die Klassifizierung in den meisten Fällen der Identifikation von Nutzern. Deshalb ist hier kein „Prefix Match“ oder „Range Match“ notwendig. Zur Klassifizierung im TZN reicht „Perfect Match“ auf einem Tupel von Feldern aus.

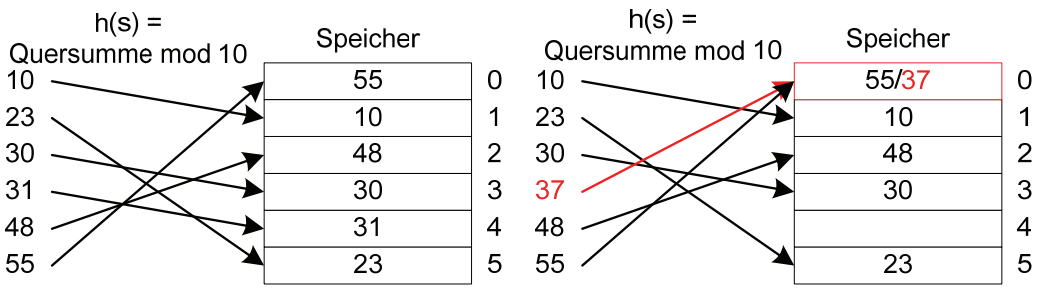
2.2 Grundlagen des Hashings

Das Hashing kann ein gutes Mittel zur effizienten Lösung des Paketklassifizierungsproblems sein, denn es ermöglicht die Suche nach einem Schlüssel mit einer potentiell konstanten Zeit- und linearen Speicherkomplexität. Daher sollen die Grundlagen in diesem Kapitel erläutert werden.

[2] Differentiated Service Code Point

2.2.1 Hashing

Der Begriff Hashing bedeutet, etwas zu zerhacken bzw. etwas durcheinander zu bringen. Die Nutzung von Hashing zur Suche von Daten basiert darauf, Teile eines Schlüssels zu zerhacken. Diese gehashten Teilinformationen sind dann zur Suche, z. B. als Index, nutzbar. Im besten Fall werden alle Elemente einer Menge von Schlüsseln so gehasht, dass jeder dieser Schlüssel einen anderen Hashwert aufweist. Der Hashwert kann dann zur Adressierung des Eintrags in einem Speicher verwendet werden. Man spricht dann von perfektem Hashing. Dies trifft jedoch in den seltensten Fällen zu. Ein Beispiel: Gegeben seien 6 Zahlen zwischen 1 und 100 (10, 23, 30, 31, 48, 55), welche in einen Speicher einsortiert werden sollen. Der Speicher enthält 6 Einträge (Abbildung 13). Wählt man als Hashfunktion $h(s)$ die Quersumme mod 10 der zu hashenden Zahlen, so ergibt sich für das gegebene Beispiel eine perfekte Abbildung. Um den Eintrag mit dem Wert 31 zu suchen, ist an der Speicherstelle seiner Quersumme (4) der Speicher auszulesen. Der Eintrag ist mit einem einzigen Speicherzugriff gefunden.



a) perfekte Abbildung b) Kollision

Abbildung 13 - Quersummenbildung als Hashfunktion.

Sobald jedoch die Verteilung der Zahlen eine andere ist, soll beispielsweise statt einer 31 eine 37 gespeichert werden, kommt es zu einer Kollision zwischen dem Eintrag der 37 und der 55. Beide haben den Hashwert 0. Eine Kollision tritt immer auf, wenn gilt:

$$h(x) = h(y), \qquad x \neq y \tag{1}$$

Da im Allgemeinen die Mächtigkeit der Menge möglicher Schlüssel S größer ist, als die Mächtigkeit der Menge möglicher Hashwerte H (|S|>|H|), sind Kollisionen unter Umständen nicht vermeidbar. Im Quersummenbeispiel ist |S|=100 und |H|=10.

Da Kollisionen auftreten können, müssen sie erkannt werden. Deshalb ist der Schlüssel s stets mit dem Datum an Adresse $h(s)$ zu vergleichen. Stimmen Eintrag und Schlüssel nicht überein, trat eine Kollision auf. Jede Kollision muss aufgelöst werden. Dafür gibt es verschiedene Strategien, welche im Folgenden näher erläutert werden sollen.

2.2.1.1 Kollisionsauflösung durch Verkettung

Eine offensichtliche Möglichkeit der Kollisionsauflösung ist die Verkettung von Einträgen. Hierbei steht zusammen mit einem Eintrag die Adresse des folgenden Eintrags mit demselben Hashwert an jeder Speicherstelle. Nach der Ermittlung des Hashwertes eines Schlüssels ist die jeweilige Liste dann solange linear zu durchsuchen, bis der korrekte Eintrag gefunden ist. Diese Methode ist für Softwareimplementierungen mit verketteten Listen zweckmäßig, wobei der Hashwert bestimmt, in welcher Liste nach einem Eintrag zu suchen ist. Wie aus Abbildung 14 hervorgeht, ist für den Fall einer Hardwareimplementierung in einer fest definierten Speicherstruktur für jeden verketteten Eintrag ein kompletter Speicher vorzusehen. Dies beschränkt die Flexibilität dieser Methode enorm. Gleichzeitig ist ein hoher Speicheraufwand notwendig, da die Anzahl der vorzusehenden Speicherblöcke der maximalen Anzahl auflösbarer Kollisionen entspricht. Überschreiten die auftretenden Kollisionen für einen Hashwert diese Zahl, ist das Problem der Kollisionsauflösung nach wie vor ungelöst. Dann müssen Einträge gelöscht, bzw. überschrieben werden. Ein solches Vorgehen findet bei verschiedenen hash-basierten Switches Verwendung [HAG06].

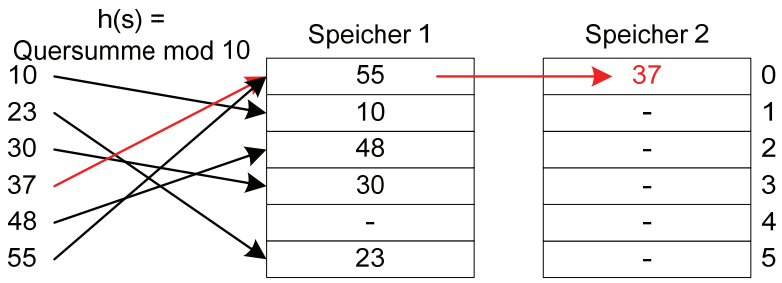


Abbildung 14 - Kollisionsauflösung durch Verkettung.

2.2.1.2 Kollisionsauflösung durch offene Adressierung

Eine weitere Möglichkeit der Kollisionsauflösung ist die Nutzung von ein und derselben Speicherstruktur zur Suche nach dem korrekten Eintrag für einen Schlüssel. Bei diesem Verfahren, der offenen Adressierung, wird ein einzelner Speicher solange durchsucht, bis der korrekte Eintrag oder eine freie Speicherstelle gefunden ist. Wird eine freie Speicherstelle gefunden, existiert kein Eintrag für den gesuchten Schlüssel. Zur Suche im Speicher stehen als Strategien das Rehashing und die lineare Kollisionsauflösung zur Verfügung.

Rehashing

Das Rehashing zur Kollisionsauflösung nutzt eine zweite Hashfunktion $h_2(s)$. Tritt eine Kollision auf, wird das Ergebnis der zweiten Hashfunktion zum Ergebnis der ersten addiert, um eine neue Speicheradresse zu finden. Das ist so lange zu wiederholen, bis der richtige Eintrag oder eine freie Speicherstelle gefunden ist.

Damit das möglich ist, muss das Ergebnis der Hashfunktion zwischen 0 und *M*-1 liegen, außerdem muss es relativ prim (teilerfremd) zu *M* sein. *M* bezeichnet hierbei den Adressraum des Speichers. Bei $M = 2^m$ ($m \in \mathbb{N}$) gilt das für alle ungeraden Zahlen. Die Funktionen $h()$ und $h_2()$ sollten nach Möglichkeit voneinander unabhängig sein, damit gute Ergebnisse erzielt werden. Dann gilt:

$$P[h(x) = h(y)] = \frac{1}{M}$$
$$P[h_2(x) = h_2(y)] = \frac{1}{M} \qquad (\forall (x,y)\ \ x \neq y) \qquad (2)$$
$$P[h(x) = h(y) \wedge h_2(x) = h_2(y)] = \frac{1}{M^2}$$

Lineare Kollisionsauflösung

Die lineare Kollisionsauflösung ist funktional noch einfacher als das Rehashing. Hierbei ist im Falle des Auftretens einer Kollision nur ein lineares Durchsuchen des Speichers durchzuführen. Dabei wird der Speicherindex *i* solange neu gebildet, bis der korrekte Eintrag oder eine freie Speicherstelle gefunden wurde. Ein neuer Speicherindex für Suche und Einfügen ergibt sich durch die Addition einer Konstanten *c* auf den aktuellen Index:

$$i_j = \begin{cases} h(s) & j = 0 \\ i_{j-1} + c & j \neq 0 \end{cases} \qquad (3)$$

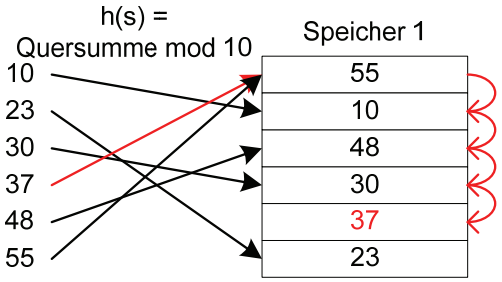


Abbildung 15 - Lineare Kollisionsauflösung durch Addition einer Konstanten (hier $c = 1$).

2.2.1.3 Belegungsfaktor

Um zu verhindern, dass es beim Hashing auch bei weniger guten Hashfunktionen zu übermäßig vielen Kollisionen kommt, kann der Belegungsfaktor α des Speichers variiert werden. Der Belegungsfaktor ist der Quotient aus der Anzahl der Einträge *N* in einen Speicher mit *M* Speicherplätzen. Je kleiner α ist, desto geringer ist die Anzahl der Kollisionen.

$$\alpha = \frac{N}{M} \tag{4}$$

2.2.2 Anwendungsbereiche von Hashfunktionen

Hashfunktionen werden in vielen Bereichen in der Kommunikationstechnik und Datenverarbeitung eingesetzt. Im Folgenden sollen kurz einige prominente Anwendungsbereiche beschrieben werden.

2.2.2.1 Hashing zur Signierung von Daten

Eine weite Verbreitung finden Hashfunktionen im Bereich der Kryptographie. Sie dienen dort in erster Linie der Signierung von Daten und Kommunikationsverbindungen. Dafür wird zum Beispiel der weit verbreitete MD5 Message-Digest-5 Algorithmus [Riv92] verwendet. Er dient vorrangig zur Signierung von Nachrichten und Daten. Der Algorithmus ist verhältnismäßig komplex, und seine Berechnung ist rechenintensiv. Mittels MD5 ist man in der Lage, einen 128-Bit Hashwert aus Nachrichten beliebiger Länge zu erzeugen. Andere Algorithmen sind z. B. der Secure Hash Algorithm (SHA)-1 [Nat02] oder Whirlpool [BR03].

2.2.2.2 Hashing zur Verifikation – CRC Generierung

Beim Hashing zur Verifikation ergibt sich der Hashwert oft einfach aus dem Rest bei einer Modulo-Division: $h(s) = s \bmod M$. In diesem Fall ist der Wert von M ganz entscheidend für die Qualität der Hashfunktion. Ist M beispielsweise gerade, ist $h(s)$ auch immer gerade, wenn s gerade ist. Ist M eine Zweierpotenz, so umfasst $h(s)$ die niederwertigsten Bits von s. In beiden Fällen ergibt sich eine qualitativ schlechte Hashfunktion.

Bei der Berechnung von Cyclic Redundance Check (CRC) – Codes, die hauptsächlich zur Fehlererkennung bei der Datenübertragung eingesetzt werden, handelt es sich ebenfalls um eine Modulo-Division. Hierbei wird der Datensatz durch ein Generatorpolynom geteilt. Der Rest der Division bildet dann den CRC-Wert. Im Ethernet wird ein CRC32 eingesetzt, dessen Polynom aus einer 32-Bit Zahl besteht. Das Polynom ist:

$$CRC32 = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \tag{5}$$

2.2.2.3 Hashing zur Suche

Wird das Hashing zur Beschleunigung eines Suchvorgangs verwendet, ist die Zielstellung, einen sehr großen Suchraum mittels einer Hashfunktion möglichst effizient zu durchsuchen. Um beim Beispiel des Routings als Ausprägung des Paketklassifizierungsproblems zu bleiben, sollen für gegebene IP-Adressen von 32 Bit Breite die jeweils zu nutzenden Ausgangsports gefunden werden. Unter der Annahme, dass ein durchschnittlicher Router $N = 128k = 2^{17}$ Regeln hat, ergibt sich eine Abbildung von: $2^{32} = 4.294.967.296 \rightarrow 131.072 = 2^{17}$. Es ist also

immer eine 32 Bit breite Zahl auf eine 17 Bit breite Zahl abzubilden. Das naive Durchsuchen einer Datenbank mit 128 k Einträgen erfolgt mit linearem Zeitaufwand, während bei einer sortierten Liste immer noch logarithmische Zeitforderungen bestehen. Die Suche mittels einer Hashfunktion hat jedoch eine ausgezeichnete Suchkomplexität. Sie ist im Allgemeinen konstant. Im Idealfall, also einer perfekten Hashfunktion, bei der keinerlei Kollisionen auftreten, ist nur genau ein Speicherzugriff notwendig, um eine Information zu finden [FKS84].

Tabelle **1** - **Komplexitäten verschiedener einfacher Suchverfahren**

Algorithmus	Zeitkomplexität			Speicher-komplexität
	Suche	Löschen	Einfügen	
Lineare Suche	$O(N)$	$O(N)$	$O(1)$	$O(N)$
Suche in sortierter Liste	$O[\log(N)]$	$O(N)$	$O(N)$	$O(N)$
Hashing	$O(1)$	$O(N)$	$O(1)$	$O(N)$

Die Konstruktion einer perfekten Hashfunktion ist jedoch sehr zeitaufwendig. Auch sind die sich ergebenden Hashfunktionen sehr komplex und rechenintensiv. Treten Kollisionen auf, sind diese mittels der oben genannten Verfahren aufzulösen. Jedoch bleibt die Komplexität unabhängig von der Anzahl der Einträge in der Datenbank. Die tatsächliche Leistungsfähigkeit des Hashings ist jedoch immer von der eingesetzten Funktion und den tatsächlich zu hashenden Elementen aus der Schlüsselmenge abhängig.

Einen entscheidenden Nachteil haben hash-basierte Suchverfahren jedoch. Ist die Hashfunktion schlecht, erzeugt sie sehr viele Kollisionen. Damit ist die Suchleistung des Verfahrens ebenfalls sehr schlecht. Die schlechteste vorstellbare Hashfunktion, um *N* Werte zu hashen, ist eine Funktion, die für jeden Wert den gleichen Hashwert erzeugt. Das bedeutet für die Suche, dass diese zu einer linearen Suche entartet. Die Qualität einer Hashfunktion für eine unbekannte oder veränderliche Schlüsselmenge ist a priori nicht bekannt. Deshalb liegt die theoretische obere Schranke der Suchkomplexität bei $O(N)$ und nicht bei $O(1)$. In [RFB97] wird Knuth aus [Knu05] zitiert: „Finally, we need a great deal of faith in probability theory when we use hashing methods, since they are efficient only on the average, while their worst case is terrible!... Therefore, hashing would be inappropriate for certain real-time applications such as air traffic control, where people's lives are at stake". Es bleibt allerdings die Tatsache bestehen, dass die Wahrscheinlichkeit eines solchen Eintretens verschwindend gering ist [Con81]. Für alle Anwendungen, bei denen keine Menschenleben auf dem Spiel stehen, sind Hashfunktionen zweifellos eine sehr gute Alternative.

Hashfunktionen zum Verschlüsseln und zur Verifikation sind prinzipiell natürlich auch für Suchalgorithmen geeignet. Die Hashwerte von CRC32, MD5 oder SHA1 bzw. Teile davon

können als Index für die Suche Verwendung finden. Die Eignung der hier vorgestellten Beispiele wird in späteren Abschnitten der Arbeit untersucht.

2.3 Grundlagen evolutionärer Algorithmen

Man kann evolutionäre Algorithmen als Suchalgorithmen betrachten. Sie dienen der Suche nach der Lösung eines Problems in einem großen Suchraum. Sie stellen eine Gruppe von speziellen Optimierungsverfahren dar. Evolutionäre Algorithmen werden besonders dort eingesetzt, wo die Komplexität eines Problems eine „herkömmliche“ Lösung nicht erlaubt. Das Finden einer perfekten oder zumindest „guten“ Hashfunktion stellt ein solches Problem dar. Auch das Beispiel in Abbildung 16, der Aufbau der Aufhängung einer Satellitenantenne, ist ein interessantes Problem. Die Aufhängung ist mittels eines evolutionären Algorithmus so optimiert, dass sie weniger anfällig gegen Vibrationen und somit stabiler ist als der Originalentwurf. Es ist leicht zu erkennen, dass die durch den evolutionären Algorithmus gefundene Lösung keine ist, die ein Ingenieur in einem normalen Entwicklungsprozess hervorbringen würde. Auch im Bereich des Internetroutings werden evolutionäre Algorithmen angewendet. [Ahn06] stellt einen solchen Algorithmus vor, um das Routingproblem zu lösen.

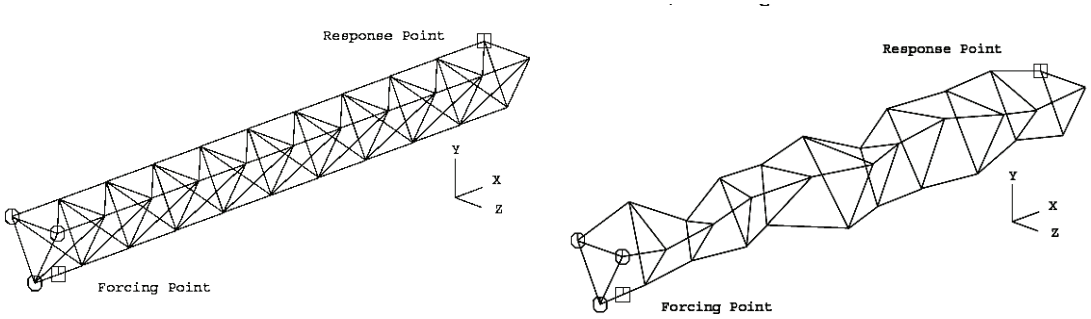


Abbildung 16 - Geometrie der Aufhängung einer Satellitenantenne. Links das originale reguläre Design. Auf der rechten Seite ist das finale Design abgebildet, welches mit einem genetischen Algorithmus ermittelt wurde [KB96].

Evolutionäre Algorithmen basieren auf zwei biologischen Grundlagen. Sie orientieren sich an den Theorien Charles Darwins zur Herkunft der Spezies [Dar59] und an der molekularen Genetik. Die Darwinschen Theorien werden auch als makroskopischer Blick auf die Evolution bezeichnet. In diesem spielt die natürliche Auslese eine zentrale Rolle: In einer Umgebung mit begrenzten Ressourcen überleben nur die Individuen einer Spezies, welche am besten an ihre Umgebung angepasst sind. Mit anderen Worten überleben nur die fittesten Individuen. Ein zweiter Punkt in Darwins Theorien ist, dass bestimmte Eigenschaften von Individuen deren Interaktion mit der Umwelt beeinflussen. Sind dieses positive Eigenschaften, so kann sich ein Individuum fortpflanzen. Sind sie es nicht, sterben sie mit dem Individuum aus.

Aus der molekularen Genetik erwächst die Erkenntnis, dass jedes Individuum zwei Seiten hat. Das sind zum einen der Phänotyp, die Summe aller äußerlich feststellbaren Merkmale, zum anderen der Genotyp, die genetische Ausstattung. Beide Begriffe prägte der dänische Botaniker Wilhelm Ludvig Johannsen [Joh09]. Der Genotyp, die Gesamtheit der Gene, beinhaltet die notwendige Information, um einen bestimmten Phänotyp auszubilden. Es ist zu beachten, dass ein bestimmter Genotyp genau eine phänotypische Ausprägung hervorruft. Individuen mit dem gleichen Phänotyp können aber unterschiedliche Genotypen aufweisen.

Man unterscheidet bei evolutionären Algorithmen vier unterschiedliche Ausrichtungen [ES03, Gol89, Hir96]. Diese unterscheiden sich im Wesentlichen in der Kodierung des Genotyps, der Ausprägung des Phänotyps und der Implementierung der genetischen Operatoren:

- Der **Genetische Algorithmus (GA)** [Whi94] manipuliert den Genotyp, der als Bitstring repräsentiert ist, um einen bestimmten Phänotyp zu realisieren. Als Operatoren finden Mutationen auf den Bits des Genotyps und Rekombinationen der einzelnen Bits Anwendung.
- **Evolutionäres Programmieren (EP)** [Fog94] nutzt keine Unterscheidung von Geno- und Phänotyp, wie das bei einem GA der Fall ist. Stattdessen findet EP auf Ebene der phänotypischen Repräsentation statt. Operationen sind auf die Mutation beschränkt.
- **Genetische Programmierung (GP)** [Koz94] nutzt relativ abstrakte hierarchische Repräsentationen, um Softwareprogramme weiter zu entwickeln. Rekombination wird of genutzt, um Codeteile des Programms zwischen verschiedenen Programmrepräsentierungen auszutauschen. Mutationen werden im Allgemeinen nicht genutzt.
- **Evolutionsstrategien (ES)** [Sch81] legen ihr Augenmerk auf die Manipulation von Phänotypen, um Optimierungsprobleme wie das „Travelling Salesman"-Problem[3] zu lösen. Im Allgemeinen wird die Mutation als Operator benutzt.

Auf Grund der rechentechnisch sehr gut verarbeitbaren Darstellung des Genoms als Bitstring, wurden im Rahmen dieser Arbeit genetische Algorithmen eingesetzt, weshalb diese im folgenden Abschnitt näher erläutert werden sollen.

2.3.1 Genetische Algorithmen

Genetische Algorithmen, wie andere evolutionäre Algorithmen auch, können als interaktive Funktionen betrachtet werden. Sie bestehen zu jeder Zeit aus einer konstanten Population P^t von μ Individuen, die die Lösung eines bestimmten Problems repräsentieren. Ein genetischer Algorithmus kann durch das folgende Tupel dargestellt werden:

[3] Travelling-Salesman Problem: Gegeben sind eine Anzahl von Städten und die Reisekosten zwischen jeder Stadt zu jeder anderen Stadt. Gesucht ist nun die kostengünstigste Route, um jede Stadt genau einmal zu besuchen und dann zur Ausgangsstadt zurückzukehren.

$$GA = (P^0, \mu, \lambda, l, s_e, r, m, s_u, f, t) \tag{6}$$

mit

$P^0 = (a_1^0, ..., a_\mu^0) \in I^\mu \quad I = \{0,1\}^l$	Initiale Population	(7)
$\mu \in \mathrm{N}$	Populationsgröße	
$\lambda \in \mathrm{N}$	Anzahl der Nachkommen	
$l \in \mathrm{N}$	Genomlänge	
$s_e : I^\mu \to I^\lambda$	Selektionsoperator (Elternselektion)	
$r : I^\mu \to I$	Rekombinationsoperator	
$m : I \to I$	Mutationsoperator	
$s_u : I^\lambda \to I^\mu$	Selektionsoperator (Umweltselektion)	
$f : I \to \mathrm{N}$	Fitnessfunktion	
$t : I^\lambda \to \{0,1\}$	Abbruchkriterium	

Ein GA besteht aus einer Population von Individuen, deren Genom durch eine Bitkette aus '0' und '1' repräsentiert wird. Diese werden als Eltern bezeichnet. Aus den Eltern werden Nachkommen erzeugt. Das geschieht durch die Anwendung verschiedener Operatoren auf die Eltern. Aus der Nachkommenschaft selektiert der Algorithmus wiederum diejenigen Individuen, die das avisierte Problem am besten lösen. Diese bilden dann eine neue Elterngeneration, welche den Algorithmus erneut durchläuft. Das geschieht solange, bis ein Abbruchkriterium erreicht wird.

2.3.1.1 Operatoren

Die in der Einführung des Kapitels vorgestellten natürlichen Vorgänge spiegeln sich in den einzelnen Operatoren genetischer Algorithmen wieder. Die am weitesten verbreiteten Operatoren bei Genetischen Algorithmen (GAs) sind Elternselektion (Parent Selection), Rekombination (Crossover), Mutation und Umweltselektion (Survivor Selection).

Elternselektion

Die Elternselektion bildet die geschlechtliche Fortpflanzung aus makroskopischer Sicht ab. Der Operator der Elternselektion dient der Auswahl von λ Elementen aus den μ Individuen einer aktuellen Generation zur Bildung der Nachkommen. In [ES03] sind drei unterschiedliche Möglichkeiten der Elternselektion beschrieben.

Die *fitnessproportionale Elternselektion* ist eine probablistische Selektion. Bei dieser kann jedes Individuum mit einer bestimmten Wahrscheinlichkeit als Elter[4] ausgewählt werden. Die

[4] Ein Elternteil wird, für den deutschen Sprachgebrauch ungewöhnlich, als Elter bezeichnet.

Größe der Auswahlwahrscheinlichkeit p_{s_i} eines Individuums *i* hängt dabei von der absoluten Fitness aller Individuen ab:

$$p_{s_i} = \frac{f_i}{\sum_{j=1}^{\mu} f_j} \tag{8}$$

Diese Methode hat drei Nachteile. Zum einen ist nicht gewährleistet, dass das fitteste Individuum zur Fortpflanzung ausgewählt wird. Zum anderen können Individuen, deren Fitness weitaus besser ist als die der anderen, innerhalb weniger Generationen die gesamte Population übernehmen. Der Algorithmus konvergiert verfrüht. Zum dritten ist bei sehr geringen Unterschieden in den Fitnesswerten der Selektionsdruck (die Notwendigkeit, einen sehr guten Fitnesswert zu haben) so gering, dass die Selektionswahrscheinlichkeiten nahezu gleichverteilt sind. Es ist also nicht von nennenswertem Vorteil, fitter zu sein als andere Individuen einer Generation.

Bei der *Selektion nach dem Rang* ist nicht die absolute Fitness eines Individuums sondern der Rang desselben entscheidend. Die Auswahlwahrscheinlichkeit richtet sich nach der Position des Individuums bei einer nach ihren Fitnesswerten sortierten Population. Wie viel fitter als andere ein Individuum ist, ist nicht von Belang. Die Abbildung des Ranges auf die Auswahlwahrscheinlichkeit variiert. Sie kann z. B. linear oder exponentiell abnehmen. Bei der exponentiellen Abnahme werden fittere Individuen weitaus stärker gewichtet als bei einer linearen Abnahme.

Die *Wettkampfselektion* benötigt zur Auswahl von λ Eltern genau λ Wettkämpfe. Bei jedem Wettkampf werden aus der aktuellen Population zufällig *k* Individuen ausgewählt, welche den Wettkampf austragen. Dabei werden die *k* Fitnesswerte der Individuen verglichen. Der beste Wettkampfteilnehmer wird zur Fortpflanzung ausgewählt. Der Wettkampf hat gegenüber den oben genannten Selektionsschemata wichtige Vorteile: Es sind zum einen keine Kenntnisse über die gesamte Population notwendig. Zum anderen ist die Implementierung auch und vor allem in Hardware sehr einfach. Außerdem kann der Selektionsdruck auf die Population sehr einfach über den Parameter *k* gesteuert werden. Ist *k* groß, werden bevorzugt fittere Individuen ausgewählt. Ist *k* dagegen sehr klein, können auch weniger fitte Individuen den Selektionsprozess überstehen. Im Extremfall $k = 1$ ist die Auswahl aus der Population zufällig und die Fitness eines Individuums hat keinen Einfluss mehr.

Rekombination

Bei der Rekombination handelt es sich wie bei der Elternselektion um die Abbildung der geschlechtlichen Fortpflanzung. In diesem Fall allerdings aus mikroskopischer Sicht. Einzelne Abschnitte der genetischen Information meist zweier Individuen – der Eltern – werden ausgetauscht. Die so entstandenen Nachkommen beinhalten dann jeweils Gene beider Eltern. Den allgemeinen Fall stellt das sogenannte n-point Crossover (Abbildung 17) dar.

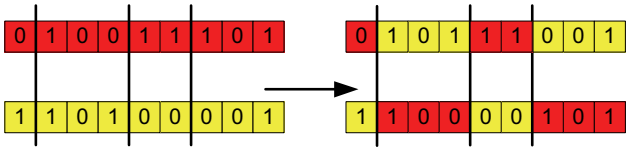


Abbildung 17 - n-point Crossover.

Hierbei werden die Genome der Eltern an n zufälligen Stellen unterteilt und zwischen den Eltern ausgetauscht. Den Spezialfall stellt das 1-point Crossover (Abbildung 18) [Hol75] mit der Unterteilung des Genoms an nur einer Stelle dar.

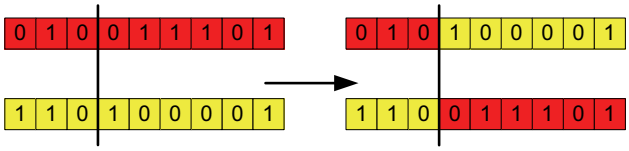


Abbildung 18 - 1-point Crossover.

Die Wahrscheinlichkeit p_r, dass eine Rekombination stattfindet, ist relativ hoch. Sie liegt üblicherweise im Intervall (0,5;1,0]. Meist liegt die Rekombinationswahrscheinlichkeit bei 0,95. Ist eine gezogene Zufallszahl kleiner als p_r, findet eine Rekombination statt. Ist das nicht der Fall, werden beide Nachkommen asexuell erzeugt, wobei kein Austausch genetischer Informationen stattfindet.

Mutation

Durch die Rekombination von Eigenschaften der Eltern und die Übertragung auf die Nachkommen kann keine zusätzliche Information in das System einfließen. Information kann allenfalls verloren gehen. Als Folge konvergiert eine Population dahingehend, dass sie nur aus einem Individuum besteht. Um diesen Mangel auszugleichen, findet der Mutationsoperator Anwendung. Bei der Mutation findet eine zufällige Veränderung des Genotyps eines Individuums statt. Bei einer binären Repräsentation ist die Mutation als Umkippen eines Bits von $0 \rightarrow 1$ bzw. von $1 \rightarrow 0$ zu verstehen (Abbildung 19).

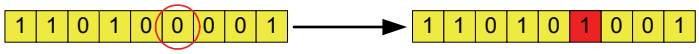


Abbildung 19 - Mutation eines Bits des Genoms.

Die Wahrscheinlichkeit p_m, dass eine Mutation auf einem Bit stattfindet, ist sehr gering. Die mittlere Anzahl der Mutationen auf einem Genom der Länge *l* beträgt demzufolge $l \cdot p_m$. Durch die Mutation gelangt stetig neue Information in die Population. Die Wahl der Größe von p_m ist

von großer Bedeutung. Ist p_m zu klein gewählt, konvergiert der GA nur sehr langsam und ist möglicherweise nicht in der Lage, ein lokales Optimum wieder zu verlassen. Wählt man p_m zu groß, artet die Suche nach einem optimalen Individuum zu einer zufälligen Suche im gesamten Suchraum aus. Der Bereich sinnvoller Mutationsraten variiert je nach Problem. Er wird auch als Evolutionsfenster [Rec94] bezeichnet. Durch eine kleine Änderung am Genotyp kann sich eine große Veränderung am Phänotyp und damit auch der Güte eines Individuums ergeben. Die Mutation soll die Erreichbarkeit aller Punkte im Suchraum (jeden möglichen Genotyp) ermöglichen. Ist die Mutationsrate groß und ruft starke Veränderungen sowohl im Geno- als auch im Phänotyp hervor, spricht man von Makromutation [Wei02].

Umweltselektion

Die Umweltselektion bildet das makroskopische Konzept des „Überleben des Fittesten“ auf GAs ab. Funktional ähnelt die Umweltselektion der Elternselektion. Allerdings findet dieser Operator während einer Generation als letzter Anwendung. Bei der Umweltselektion müssen aus den μ Eltern und λ Nachkommen der *t*-ten Generation μ Eltern für die Gereration *t+1* ausgewählt werden. Die Selektion kann mittels verschiedener Strategien erfolgen. Die meist verwendeten sind die altersbasierte Ersetzung und die fitnessbasierte Ersetzung.

Die *altersbasierte Ersetzung* bezieht die Fitness der einzelnen Individuen nicht mit in die Auswahl der neuen Generation ein. Bei dieser Strategie werden einfach alle μ alten Eltern durch ihre λ Nachkommen ersetzt. Wobei meistens $\mu = \lambda$ gilt. Die altersbasierte Ersetzung lässt sich auch für Fälle von $\lambda < \mu$ anwenden. Hierbei ist in einer Generation nur ein Teil der Eltern durch die Nachkommen zu ersetzen.

Die *fitnessbasierte Ersetzung* nutzt alle $\mu + \lambda$ Individuen und ermittelt die μ Individuen mit der besten Fitness. Diese bilden dann die neue Generation. Die fitnessbasierten Ersetzung ist auch auf Fälle von $\lambda > \mu$ anwendbar.

2.3.1.2 Fitnessfunktion

Einen wichtigen und zumeist problematischen Teil genetischer Algorithmen stellt die Fitnessfunktion dar. Die Fitnessfunktion definiert, inwieweit ein bestimmtes Individuum aus der Population den Anforderungen, die an die Lösungskandidaten des Problems gestellt wurden, gerecht wird. Die Auswahl einer geeigneten Fitnessfunktion ist kein triviales Problem. Zum einen muss die Fitnessfunktion die Eignung eines Individuums abbilden, eine bestimmte Problemstellung zu lösen, damit Lösungskandidaten mittels der Fitnessfunktion vergleichbar sind. Sie ist demzufolge zum Teil sehr komplex, um für ein Problem angemessen zu sein. Zum anderen muss die Fitnessfunktion überhaupt bzw. mit angemessenem Aufwand berechenbar sein, denn die Berechnung der Fitnessfunktion ist in jeder Generation für jeden Lösungskandidaten notwendig. Deshalb stellt die Fitnessfunktion den bestimmenden Faktor für die

Laufzeit und die Komplexität eines genetischen Algorithmus dar. Um beispielsweise die Qualität der Aufhängung der Satellitenantenne zu bewerten, sind für jeden Lösungskandidaten Statikberechnungen notwendig, damit die Tragfähigkeit bewertet werden kann.

2.3.1.3 Schwächen

Die Nutzung von GAs als Optimierungsstrategie birgt auch Nachteile. Eine Schwäche von GAs, wie auch aller anderen evolutionären Algorithmen ist, dass die Optimalität einer gefundenen Lösung nicht garantiert werden kann. Es ist möglich, bei vielen komplexen Problemen sogar wahrscheinlich, dass ein eingesetzter GA zu einem lokalen Maximum im Lösungsraum des Problems konvergiert. Abbildung 20 verdeutlicht diesen Sachverhalt anhand eines eindimensionalen Lösungsraumes. Mit verschiedenen Maßnahmen wie dynamischen Schrittweiten [Och00], einem angepassten Selektionsdruck [Rec94] oder einer variablen Populationsgröße [CC05] kann dieser Schwäche bis zu einem gewissen Grad begegnet werden. Diese Maßnahmen ermöglichen dem GA eine größere Diversifizierung der genetischen Informationen innerhalb der Population.

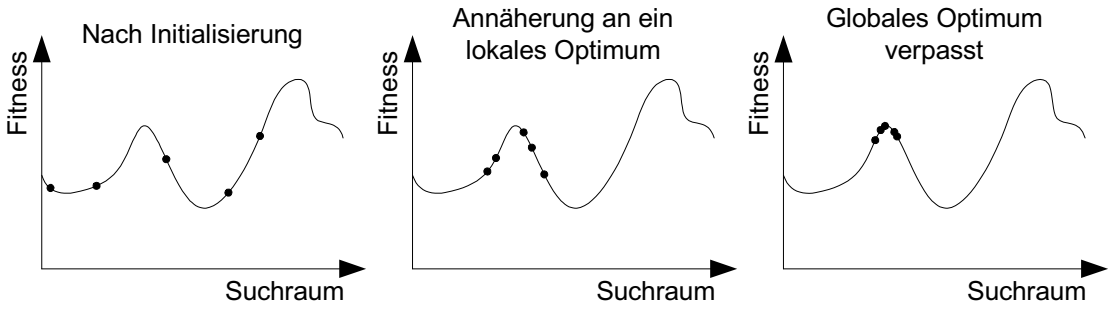


Abbildung 20 - Verlauf eines GA. Das globale Optimum wird nicht erreicht. Der Algorithmus konvergiert zu einem lokalen Optimum [Toc07].

Ein weiteres Problem von GAs ist die unbestimmte und eventuell sehr lange Laufzeit, bis eine befriedigende Lösung gefunden wird. Dies liegt in den nicht deterministischen Operationen von Rekombination, Mutation und zum Teil auch Selektion begründet. Außerdem beeinflussen die Beschaffenheit der Fitnessfunktion *f* und die Wahl des Abbruchkriteriums *t* die Laufzeit maßgeblich, wenn t eine Funktion von *f* ist.

Aufgrund des Zeitverhaltens von GAs, wie es in Abbildung 21 dargestellt ist, kann die Wahl einer zu hohen Abbruchschwelle für *t* sogar dazu führen, dass der GA das Abbruchkriterium niemals erfüllt. Nach einer Phase starker Qualitätsverbesserungen zu Beginn der Laufzeit eines genetischen Algorithmus nimmt diese mit längerer Laufzeit immer weiter ab, bis der Algorithmus in einem Optimum (lokal oder global) konvergiert.

Schließlich ist die große Zahl variierbarer Parameter $(\mu, \lambda, p_r, p_m, s, f, t)$ problematisch, um für ein spezielles Problem den optimalen genetischen Algorithmus zu finden.

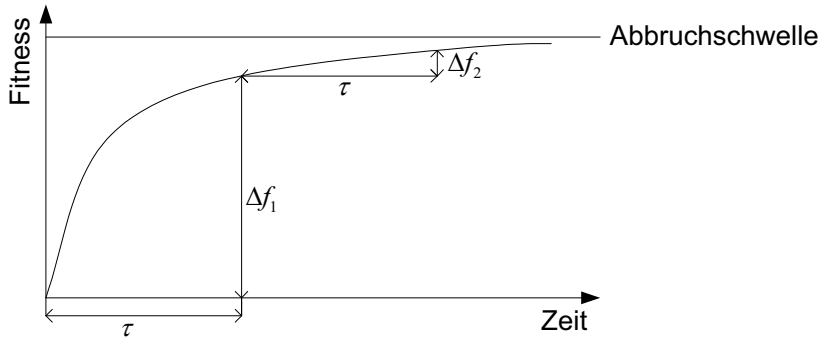


Abbildung 21 - Typisches Laufzeitverhalten von evolutionären Algorithmen [Toc07]. Dabei kann es von einer Generation zur nächsten, abhängig von der gewählten Umwelt-selektionsstrategie, auch zu graduellen Verschlechterungen der Fitness kommen.

2.3.1.4 Zusammenfassung

Im Allgemeinen wird der Selektionsdruck bei GAs allein durch die Elternselektion aufgebaut. Die entstehenden Nachkommen ersetzen die Elterngeneration komplett. Es gilt: $\mu = \lambda$, da eine einfache altersbasierte Ersetzung als Umweltselektion durchgeführt wird. Die Rekombination gilt für GAs als primärer Operator und wird mit einer hohen Wahrscheinlichkeit angewendet. Die Mutation wird als sekundär betrachtet und p_m ist sehr viel kleiner als p_r. Die Mutation ist jedoch von großer Bedeutung, um die Diversität innerhalb der Population zu erhalten. Die Mutation kann allerdings nicht verhindern, dass die Population konvergiert, das heißt, dass sie nur noch Individuen mit ähnlichen Genotypn enthält [Wei02].

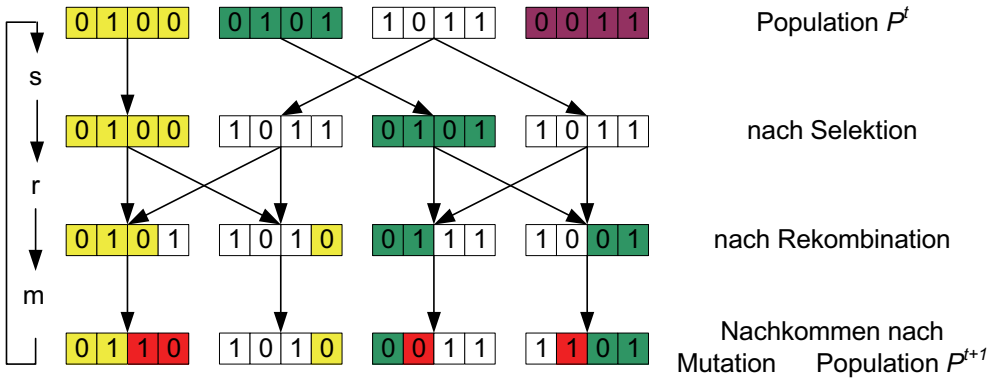


Abbildung 22 - Vereinfachter Ablauf eines genetischen Algorithmus mit den Operatoren Elternselektion (s), Rekombination (r) und Mutation (m).

2.3.2 Evolvable Hardware

Die logische Fortsetzung bei der Nutzung evolutionärer Algorithmen ist deren vollständige Umsetzung in eine Hardwareimplementierung − eine Evolvable Hardware (EHW). Als Evolvable Hardware bezeichnet man Hardware, die ihre eigene Architektur und Eigenschaften dynamisch und autonom durch Interaktion mit ihrer Umgebung ändern kann [GT07]. Man unterscheidet bei EHW zwischen der sogenannten extrinsischen und der intrinsischen EHW [YH99]. Extrinische EHW simuliert die Evolution in Software. Nur die beste Konfiguration jeder Generation wird auf die Hardware herunter geladen. Bei intrinsischer EHW wird jedes Genom einer Generation auf die Hardware geladen und die Fitnessevaluation findet auf der Hardware statt. In [TH99] wird zudem eine dritte Form der EHW vorgestellt – Complete Hardware Evolution (CHE). Im Gegensatz zur intrinsischen EHW implementiert CHE sowohl den evolutionären Algorithmus als auch die zu optimierende Schaltung auf der Zielplattform. Dies sind meist rekonfigurierbare Plattformen wie Field Programmable Gate Arrays (FPGAs).

2.3.2.1 Field Programmable Gate Array

FPGAs eignen sich auf Grund ihrer Rekonfigurierbarkeit sehr gut für die Implementierung einer Evolvable Hardware. Es handelt sich um integrierte Schaltkreise von sehr regelmäßiger Struktur (Abbildung 23). Sie bestehen im Kern aus konfigurierbaren Logikblöcken (engl. Configurable Logic Block – CLB), welche eine gewünschte logische Funktion abbilden können. Die CLBs sind über programmierbare Routingverbindungen miteinander vernetzbar. IO- (engl. Input-Output) Blöcke zur Kommunikation mit externen Systemen vervollständigen das Bild.

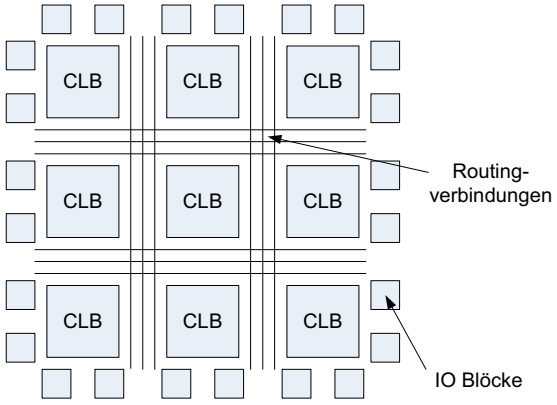


Abbildung 23 - Eine einfache FPGA-Architektur [GT07].

Jeder CLB enthält vier sogenannte Slices. Jeder Slice der Virtex-4 Reihe von Xilinx enthält unter anderem zwei Lookup Tables (LUTs) mit je vier Eingängen, außerdem Multiplexer und vier Flipflops zur Abbildung taktgesteuerter Logik [Xil04a]. Die LUTs sind in der Lage, jede logische Funktion mit vier Eingängen abzubilden. Neben den CLBs besitzen moderne FPGAs auch spezielle Hardwareblöcke, um IO-Funktionalität (Ethernet MACs), spezielle arithmetische Funktionen (Multiply and Accumulate) oder Speicher (Block RAM – BRAM) bereitzustellen.

Alle Elemente sind programmierbar. Die Programmierung erfolgt häufig mittels SRAMs (Static RAM). Die Konfiguration der SRAM-Zellen geschieht durch einen Bitstring. In diesem Fall kann also bei dem Konfigurationsstring vom Genotyp und bei der Funktion, die mit Hilfe des FPGAs abgebildet wird, vom Phänotyp der Hardwarefunktionalität gesprochen werden. Moderne FPGAs unterstützen partielle Rekonfigurierbarkeit zur Laufzeit. Dabei werden immer nur Teile des FPGAs rekonfiguriert. Der Rest bleibt in seiner Struktur unverändert. Damit können sich Teile einer Hardware selbst verändern. Während die Chips diese Funktionalität bereits unterstützen, sind die Softwarewerkzeuge noch nicht ausgereift genug, um partielle Rekonfigurierbarkeit zur Laufzeit im täglichen Einsatz zu ermöglichen. Es ist allerdings zu erwarten, dass sich dies in naher Zukunft ändert, da entsprechende Werkzeuge und Techniken Gegenstand der aktuellen Forschung sind [HLT+03, YAF+03, McD08, LPS08].

2.3.2.2 Alternative Plattformen für Evolvable Hardware

Als Hardwareplattformen für eine EHW kommen nicht nur FPGAs in Frage, sondern auch andere rekonfigurierbare analoge und digitale Schaltkreise.

Programmable Logic Device

Neben FPGAs sind auch Programmable Logic Devices (PLDs) aufgrund ihrer Konfigurierbarkeit geeignet, als Evolvable Hardware eingesetzt zu werden. PLDs und ihre Erweiterungen Complex PLDs (CPLDs) haben allerdings den Nachteil, dass sie nicht rekonfigurierbar sind. Einmal konfiguriert, steht die Funktionalität fest. Insofern stellen FPGAs die bessere Lösung im Bereich der digitalen Logik dar.

Analoge Schaltkreise

Es gibt einige Forschungsbemühungen, um auch im Analogbereich Evolvable Hardware einzuführen. Das ist im Vergleich zu digitaler Logik weitaus komplizierter. Die Eigenschaften eines Transistors hängen beispielsweise von vielen Fertigungsparametern ab. Eine digitale Funktion kann immer als Sum-of-Products ausgedrückt werden. Es gibt nichts Äquivalentes im analogen Bereich. In [GT07] werden drei Ebenen beschrieben, auf denen EHW angewendet werden kann. Zum einen kann ein einzelner Transistor in seinen Eigenschaften beeinflusst werden, indem Gate, Source und Drain mit steuerbaren Multiplexern verbunden werden, die wiederum verschiedene Signale an den Transistor durchschalten können [LBF+01]. Ein solcher Schaltkreis wird als Field Programmable Transistor Array (FPTA) bezeichnet. Eine Abstraktionsebene höher ist das Transistorarray von JPL[5] angesiedelt [SKA+04]. Hier können mehrere Transistoren mittels steuerbarer Schalter miteinander verbunden werden. Durch die verschiedenen Verschaltungen sind unterschiedliche Funktionalitäten realisierbar. Der ispPAC10 von Lattice Semiconducter [Lat00] geht einen Schritt weiter. Hier werden analoge

[5] JPL: Jet Propulsion Laboratory

Blöcke (Operationsverstärker) unterschiedlichen verdrahtet. Zusätzlich ist es möglich, passive Komponenten (in diesem Fall Widerstände) mit unterschiedlichen Werten zwischenzuschalten, um sich verändernde Funktionalität zu erzeugen. Analoge Schaltkreise auf höheren Ebenen werden als Field Programmable Analog Array (FPAA) bezeichnet.

EHW auf Basis analoger Schaltkreise wird oft auch in Verbindung mit FPGAs verwendet. So wurde in [SZK+06] beispielsweise ein rekonfigurierbares analoges Array entwickelt, welches sich mittels eines GA an unterschiedliche Temperaturumgebungen anpassen kann. Der GA ist hierbei in einem FPGA implementiert.

2.3.2.3 Umsetzung von Zufallsprozessen in FPGAs

Wie aus den Erläuterungen zu den Operatoren von GAs hervorgeht, sind diese nahezu alle auf die Erzeugung von Zufallszahlen angewiesen, damit der Algorithmus wie vorgesehen arbeiten kann. Die Erzeugung von Zufallszahlen in Hardware ist nicht unproblematisch. Zunächst ist festzustellen, dass sowohl in Hard- als auch Software Pseudozufallszahlen unterschiedlicher Qualität Verwendung finden. Für die ressourcensparende Erzeugung von pseudozufälligen Zahlen ist das sogenannte Linear Feedback Shift Register (LFSR) [KD73] gut geeignet. Das LFSR übernimmt bei einer steigenden Flanke einen Startwert. Dieser wird mit jedem Takt durch das Register geschoben. Dabei bewegt sich der Wert von Stelle *i* an Stelle *i*+1. Um eine Sequenz von Zahlen zu erzeugen, werden nun Registerwerte von unterschiedlichen Positionen auf die erste Stufe zurückgeführt. Im Feedbackpfad zur ersten Stufe werden alle zurückgeführten Werte miteinander addiert (Modulo-2-Addition). Effektiv müssen also XOR-Gatter eingesetzt werden. Die Eigenschaften der Zufallszahlen sind dadurch bestimmt, welche und wie viele Registerausgänge auf den Eingang zurückgeführt werden.

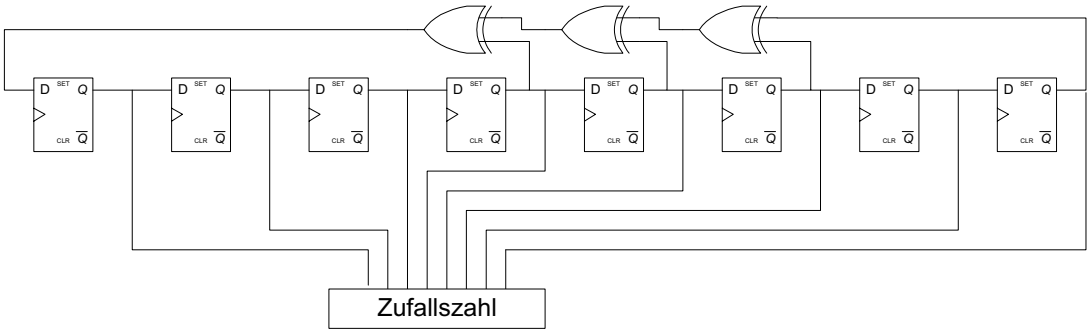


Abbildung 24 - Aufbau eines 8-Bit LFSR. Die Werte der Stufen vier, fünf, sechs und acht werden auf den Registereingang zurückgeführt.

Für den Einsatz in FPGAs bietet die Firma Xilinx IP-Cores an, die es ermöglichen, LFSRs mit einer Bitbreite zwischen 3 und 168 Bit einzusetzen [Xil96]. Diese enthalten maximal sechs Bitpositionen, von denen das Feedback ausgeht. Xilinx nutzt für das Feedback XNOR-Gatter. Im Rahmen der Entwicklungen dieser Arbeit wurden ausschließlich derartige LFSRs unterschiedlicher Bitbreite eingesetzt.

3 Architekturen für paketverarbeitende Systeme

Der folgende Abschnitt stellt den aktuellen Stand der Technik der beiden für diese Arbeit relevanten Gebiete dar. Das sind zum einen Architekturen paketverarbeitender Systeme. Zum anderen werden verschiedene Algorithmen und Architekturen zur Paketklassifizierung dargestellt. Bei der Paketklassifizierung werden besonders solche Architekturen vorgestellt, die auf Hashfunktionen basieren, denn Optimierungen dieser Hashfunktionen sind ein Ziel dieser Arbeit. Die verwendeten Hashfunktionsarchitekturen werden ebenfalls erläutert.

3.1 Paketverarbeitende Systeme

Paketverarbeitung kann entweder per Software auf Mikroprozessoren oder Netzwerkprozessoren oder als Hardwarelösung auf speziellen ASICs[6] bzw. FPGAs stattfinden. Außerdem existieren spezielle Hardwarearchitekturen, die die Verarbeitung von Code zur Paketverarbeitung beschleunigen. In Abbildung 25 ist qualitativ das Verhältnis der unterschiedlichen Varianten zueinander dargestellt. Je nach Anforderung an Geschwindigkeit und funktionale Variabilität sind Soft- oder Hardwarelösungen vorzuziehen. Während Mikroprozessoren wegen ihrer Programmierbarkeit die größte Flexibilität bieten, können sie leistungsmäßig nicht mit speziellen ASICs konkurrieren. Diese sind aber weit weniger flexibel. Einige Lösungen sind in diesem Abschnitt dargestellt.

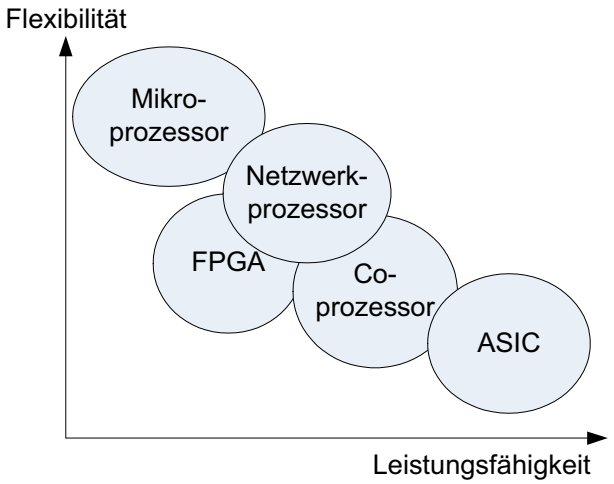


Abbildung 25 - Verhältnis von Flexibilität und Leistungsfähigkeit verschiedener paketverarbeitender Systeme nach [Sha01].

[6] ASIC: Application Specific Integrated Circuit

3.1.1 Prozessorarchitekturen für Softwarelösungen

Netzwerkprozessoren werden von vielen Herstellern angeboten und sind Gegenstand intensiver Forschungsbemühungen. Sie basieren zumeist auf RISC[7]-Prozessoren, welche jedoch um spezielle Instruktionen zur Paketverarbeitung erweitert wurden.

3.1.1.1 Intel IXP45X Netzwerkprozessor

Der IXP von Intel ist ein Netzwerkprozessor auf Basis eines XScale Kerns [Int06b]. Um Paketverarbeitung in wirespeed zu ermöglichen, besitzt der Prozessor zusätzliche sogenannte Network Processing Engines (NPEs). Davon besitzt der IXP455 drei. Die NPEs sind Prozessoreinheiten, die mittels Coprozessoren MAC, CRC-Funktionalität, Verschlüsselung (AES, DES) und Hashing (SHA, MD5) durch Hardwarekomponenten unterstützen. Eine NPE hat die Aufgabe, rechenintensive Funktionen, die nur schwer mit einem normalen RISC-Prozessor umzusetzen sind, schnell und effizient zu bearbeiten. Der NP arbeitet mit einer Taktfrequenz von 133 MHz und ist als Havard-Architektur implementiert. Der IXP unterstützt Hardwaremultithreading mit mehreren Kontexten, zwischen denen innerhalb eines Taktes gewechselt werden kann. Damit sind Wartezyklen bestimmter Threads[8] sinnvoll durch andere Threads nutzbar und werden so maskiert.

3.1.1.2 IBM PowerNP

Der PowerNP [ABB+03] integriert MACs, Prozessoren, Funktionalitäten zum Traffic Management und einen eingebetteten PowerPC. Die wichtigsten Komponenten sind der eingebettete Prozessorkomplex (EPC), die Datenflusseinheit (DF), ein Scheduler, die MACs und Coprozessoren. Die Coprozessoren bieten Hardwareunterstützung für verschiedene wichtige Funktionalitäten an, wie z. B. die Suche in Tabellen oder die Berechnung eines CRC-Wertes. Die eigentliche Berechnungsleistung wird im EPC erbracht. Der EPC kann Daten sowohl von der Eingangs- als auch der Ausgangs-DF verarbeiten. Im EPC befinden sich insgesamt 8 zweiteilige Prozessoreinheiten (Dyadic Protocol Processor Unit – DPPU). Diese bestehen aus zwei Prozessoren, Coprozessoren und Hardwarebeschleunigern. Die Prozessoren sind sogenannte 32 Bit Picoprozessoren. Das sind abgespeckte RISC-Prozessoren, die mit 133 MHz betrieben werden. Sie bestehen aus 16 32 Bit oder 32 16 Bit Registern für jeden Thread und einer ALU, die innerhalb eines Taktes verschiedene spezielle Befehle, wie z.B. „Count Leading Zeros“ ausführen kann. Insgesamt kann der PowerNP 32 Thread bearbeiten, davon 16 gleichzeitig. Zusammenfassend ist zu sagen, dass die Leistung des NP durch Parallelität von vielen spezielle Befehle beherrschenden Prozessoren und speziellen Hardwarebeschleunigern erzielt wird. Damit ist der PowerNP in der Lage, einen Ethernetdatenstrom von 3GBit/s und minimalen Paketgrößen zu verarbeiten.

[7] RISC: Reduced Instruction Set Computing

[8] Thread: Ausführungsstrang bei der Abarbeitung von Softwareprogrammen

3.1.1.3 N-Core

Der N-Core Prozessor ist ein Softcore, vergleichbar mit dem Nios von Altera [Alt04] oder dem Microblaze von Xilinx [Xil06]. Er liegt als VHDL-Beschreibung vor und kann sowohl auf einem FPGA als auch auf einem ASIC implementiert werden. Es handelt sich um einen RISC-Prozessor mit 16 32 Bit Registern und Instruktionen von 16 Bit Breite. 11% der Instruktionen sind ungenutzt, was eine Erweiterung des Befehlssatzes für spezielle Applikationen ermöglicht. Es sind bereits spezielle Instruktionen zur Beschleunigung von Headermodifikationen von Kommunikationsprotokollen und für das IPsec-Protokoll implementiert [KLS+04, GKL+04]. Außerdem bietet der N-Core eine Coprozessorschnittstelle zur Anbindung von speziellen IP-Cores. Der N-Core ähnelt in seinem Aufbau der RISC-Architektur des PowerNP. Er ist bitkompatibel zum M-Core von Motorola [Mot98]. Die FPGA-Implementierung einer Minimalversion benötigt 3727 Slices auf einem Virtex-1000 FPGA und erreicht 12 MHz.

3.1.1.4 NetVM

NetVM [DBB+05] ist ein virtueller Netzwerkprozessor. Er ist am ehesten mit einer Java Virtual Machine vergleichbar. Diese virtualisiert eine CPU, während NetVM einen NP virtualisiert. Dabei kann NetVM auf jeder Plattform eingesetzt werden, sei es ein Mikro- oder ein spezieller Netzwerkprozessor wie der Intel IXP oder der N-Core. NetVM ist eine virtuelle Maschine eines RISC-basierten Prozessorkerns mit speziell auf die Paketverarbeitung ausgerichteten Instruktionen. Das heißt, Befehle für die Fließkommaverarbeitung fehlen, während zusätzliche Befehle für Bitmanipulationen den Befehlssatz ergänzen (`set.bit`, `clear.bit`, `flip.bit`, `test.bit`).

Da Paketverarbeitung mit einer Pipeline bzw. auf einem Array von Paketprozessoren auszuführen ist (einzelne zu verarbeitende Pakete sind weitgehend unabhängig voneinander), setzt auch der NetVM auf eine modulare Architektur, die auf dem Konzept des Processing Elements (NetPE) basiert. Diesen Ansatz kann man mit dem Einsatz mehrerer N-Cores oder den parallelen Verarbeitungseinheiten des Intel IXP vergleichen. Jedes NetPE ist eine virtuelle CPU mit eigenem lokalen Speicher und Befehlssatz. Es führt seine individuellen Funktionen aus und behält dabei seinen eigenen privaten Zustand, unabhängig von anderen NetPEs. Die gesamte NetVM Anwendung setzt sich dann aus den einzelnen Tasks auf den NetPEs zusammen. Dieser Grad an Modularität resultiert aus der Beobachtung, dass nicht nur die Pakete selbst voneinander unabhängig sind, sondern auch die Funktionen, die auf dem einzelnen Paket auszuführen sind. NetVM kann deshalb die einfachen NetPEs zu komplexeren Strukturen verbinden, die sowohl sequentielle als auch parallele Vorgänge abzubilden vermögen.

Zusätzlich zu den NetPEs werden Coprozessoren eingefügt. Zum normalen Befehlssatz kommen spezielle Befehle, die für die Paketverarbeitung sinnvoll sind. Als Beispiel wird die CRC32-Berechnung angeführt. An dieser Stelle ist aber auch ein Befehl zum Hashen denkbar. Die speziellen Befehle stehen allen NetPEs zur Verfügung. Je nachdem, welche Möglichkeiten die Zielhardwareplattform der NetVM anbietet, sind die Coprozessoren als Hardwareblock

nutzbar. Ein Hashbefehl könnte dann z.B. durch die Hardwarehashengine eines Intel IXP ausgeführt werden.

Die Leistungsfähigkeit des NetVM ist allerdings begrenzt. Ein IPv4 Filter benötigte mit NetVM 2236 Taktzyklen, wobei keine Angaben über die Zielplattform und deren Frequenz gemacht wurden, auf der die virtuelle Maschine ausgeführt wurde. Um jedoch einen GBit Ethernetdatenstrom auf einem Prozessor schritthaltend zu verarbeiten, wären Frequenzen von mehr als 3 GHz notwendig.

3.1.2 Existierende Speziallösungen für ASICs

3.1.2.1 Programmierbare Zustandsmaschine

Die Autoren der Programmable State Machine (PSM) [LL03] bieten eine Lösung an, um das Problem der Invariabilität von paketverarbeitenden Systemen auf ASICs zu lösen. Viele Problemstellungen in der Paketverarbeitung befassen sich mit Protokollverarbeitung, die mit einer definierten Architektur, z.B. einer Finite State Machine (FSM), effizient lösbar sind. Flexible NPs sind hier nicht notwendig. Dennoch wird eine gewisse Programmierbarkeit benötigt, um sich beispielsweise Protokolländerungen anzupassen. PSMs bieten dies. Bei einer PSM handelt es sich im Prinzip um einen stark vereinfachten RISC-Prozessor. Zur Unterstützung der Paketverarbeitung stehen insgesamt 18 Instruktionen einer festen Länge von 29 Bit zur Verfügung. Diese werden in Register-, Immediate- und Sprungbefehle unterteilt. PSMs sollen lediglich unflexible FSMs ersetzen. Probleme mit der Reihenfolge von Programmen oder der Datenintegrität (Daten- oder Kontrollhazards), die bei normalen Prozessorarchitekturen beachtet werden müssen, treten hier nicht auf. Auch ist keine Interuptverarbeitung notwendig, was die Architektur der PSM vereinfacht. Hardware-architekturen, die zur Paketverarbeitung vorgesehen sind, enthalten viele Zustandsmaschinen („..tens of FSMs…“ [LL03]). Durch PSMs sollen all diejenigen ersetzt werden, die von einer gewissen Programmierbarkeit profitieren. Andere, als Beispiel werden DMA-Controller genannt, sind nicht zu ersetzen, da diese von einer Programmierbarkeit nicht profitieren würden. PSMs können direkt mit anderen FSMs oder PSMs verbunden werden. Sie kommunizieren über Register, wie das bei normalen FSMs auch der Fall ist. Sollen nun im Feld Veränderungen vorgenommen werden, müssen nur die entsprechenden Zeilen des Instruktionscodes geändert werden. In [LL03] zeigen die Autoren dies an einem Beispiel.

Die Implementierung auf einem ASIC mit einem 0.13µm-Prozess erreicht 250 MHz. Diese Leistung reicht aus, um Datenraten von 2.5 GBit/s zu verarbeiten.

3.1.2.2 GigaNetIC

Der GigaNetIC (Abbildung 26) [NPR05, NPP+07] basiert auf einem Network-on-Chip (NoC), genannt GigaNoC als Kommunikationsinfrastruktur zwischen Funktionsblöcken. Dieses

GigaNoC ist ein hierarchisches hybrides NoC. Es sind drei Hierarchiestufen zu unterscheiden: PE (Processing Element) Ebene, Cluster-Ebene und System-on-Chip (SoC) Ebene.

Auf PE-Ebene ist der oben beschriebene RISC-basierte N-Core [LNP+02] eingesetzt. Er hat die Aufgabe, alle relevanten Berechnungen auszuführen. Auf Clusterebene ist an jeden Router des NoCs ein Cluster von funktionalen Elementen angeschlossen. Diese stellen ein kleines SoC dar, das aus Prozessoren (N-Cores), Speicher, Peripherie oder speziellen IP-Blöcken bestehen kann. Die funktionalen Elemente kommunizieren auf Clusterebene über den Whishbone Bus [Sil02] miteinander. Auf PE-Ebene können funktionale Erweiterungen nur von dem N-Core genutzt werden, an den sie gekoppelt sind. Im Gegensatz dazu können auf Clusterebene alle an den Whishbone angeschlossenen N-Cores bzw. funktionalen Elemente Zugriff auf speziell implementierte IP-Cores erlangen.

Auf oberster Ebene tauschen Router (Switch Boxes) Daten untereinander aus. Hier ist das GigaNoC als Kommunikationsinfrastruktur implementiert. Das GigaNetIC ist sehr groß und komplex. Es eignet sich deshalb nicht für eine effiziente Implementierung in einem FPGA. Die Autoren des GigaNetIC sehen es für eine ASIC-Implementierung vor. In [NPS+05] schätzen die Autoren die Leistungsfähigkeit des Systems für eine Implementierung als DSLAM ab. Zusammenfassend ist daraus abzuleiten, dass das System in der Lage ist, Berechnungen wie IP-Headerchecks oder CRC-Berechnungen durchzuführen. Es sind jedoch viele N-Cores notwendig. Für einen DSLAM mit 64 line cards und jeweils 96 DSL-Ports wären allein 13 N-Cores nur mit der Aufgabe beschäftigt, den IP-Header zu überprüfen. Für eine volle Funktionalität und hohe Bandbreiten sind also sehr viele N-Cores notwendig.

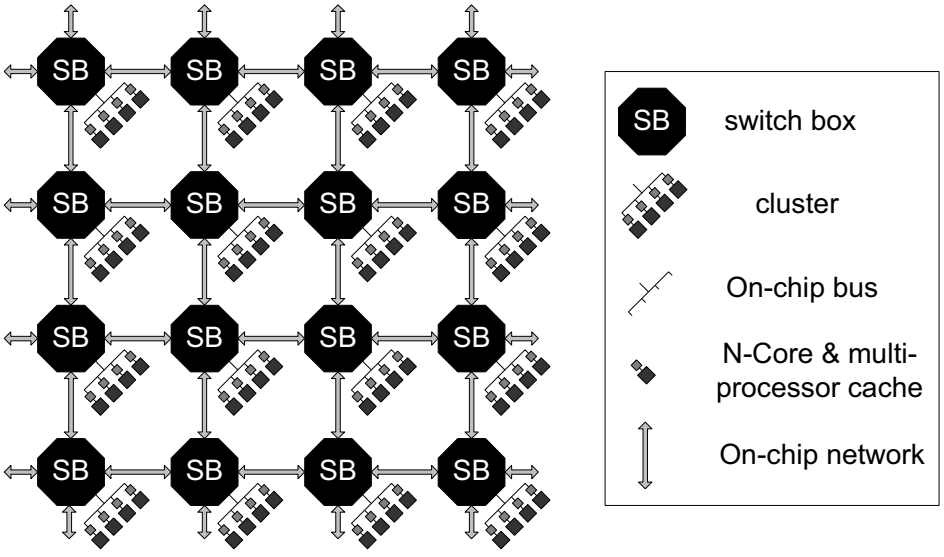


Abbildung 26 - Architektur des GigaNetIC nach [NPS+05].

3.1.2.3 Programmierbare Processing Engine

In [PVN+04] stellen Papaefstathiou et.al. eine Hardwarearchitektur vor, die in der Lage ist, Paketverarbeitung effizienter auszuführen als gängige NPs (Intel IXP, PowerNP). Es handelt sich hierbei um eine 3-stufige Pipeline, die Programmable Processing Engine (PPE). Die PPE besteht aus einem Field Extractor (FEX), einem RISC-Kern und einer Field Modification Unit (FMO). Diese werden von einem Datenpaket nacheinander durchlaufen und haben die folgenden Aufgaben:

- FEX: Der FEX ist eine programmierbare Hardware, die in der Lage ist, unterschiedliche Felder aus einem Datenpaket zu extrahieren. Die Operationen sind durch einen Microcode gesteuert, der in einem SRAM gespeichert ist. Der Microcode unterstützt folgende Operationen:
  - Feldextraktion mit variabler Länge (1 – 32 Bit)
  - Bewegung vorwärts und rückwärts auf dem Paket
  - Bedingte Sprünge
  - Additionen

  Damit ist es möglich, Datenpakete sehr variabel zu parsen. Die extrahierten Daten werden dann dem RISC-Kern übergeben.
- Der RISC-Kern ermittelt, welche Modifikationen an einem Datenpaket durchgeführt werden sollen. Er kann beispielsweise eine Paketklassifizierung anhand der Daten des FEX durchführen.
- Die FMO hat die Aufgabe, die Datenpakete zu manipulieren. Sie ist auch in der Lage, Pakete zu erzeugen. Das geschieht mit Hilfe einer hart codierten Firmware.

Das System hat eine Größe, die einem Gatecount von etwa 50.000 entspricht. Das ist in etwa mit 10.000-25.000 Slices auf einem FPGA vergleichbar[9]. Was die Leistungsfähigkeit betrifft, verweisen die Autoren auf eine 2,5-fache Leistung im Vergleich zu einem Intel IXP 120. Daten, den Paketdurchsatz betreffend, wurden nicht angegeben.

3.1.3 Lösungen auf FPGA-Basis

Von Architekturen auf FPGA-Basis können aus den bereits hinreichend diskutierten Gründen sehr gute Kompromisse aus Leistungsfähigkeit und Flexibilität erwartet werden.

3.1.3.1 Programmierbare Verarbeitungseinheit

Die Autoren von [MOW+07] stellen eine interessante und flexible Architektur vor, die es ermöglicht, Paketmanipulationen in Netzwerkprozessoren effizient durchzuführen. Es wird ein Post-Prozessor vorgestellt, der sich in das sogenannte FlexPath Konzept [OHW05] einfügt.

[9] Die Implementierung des in [SCK+06] vorgestellten NoC sowohl für einen ASIC als auch auf einem FPGA ergab ein Verhältnis zwischen 1:2,5 und 1:5,5 von Slices zu Gattern, abhängig von Syntheseparametern wie der Größe der eingesetzten FIFOs (siehe Anhang AAnhang A).

Dabei handelt es sich um ein Konzept, das Standardpaketverarbeitung am Prozessor vorbei in speziellen Hardwarestufen autonom ausführt, damit diese in wirespeed und hohen Durchsätzen ausgeführt werden kann. Es gibt drei Stufen: den Pre-Prozessor, den Path-Dispatcher und den Post-Prozessor. Das Konzept lässt sich in die Standardverarbeitung von Fast- und Slow-Path einorden. Der Pre-Prozessor ist für das Extrahieren der Header zuständig. Woraufhin der Path-Dispatcher entscheidet, ob spezielle Behandlungen (Slow-Path) notwendig sind oder nicht (Fast-Path). Der Post-Prozessor kann dann alle möglichen Aufgaben übernehmen, die mit der Paketverarbeitung zusammenhängen. Als Aufgaben für einen Post-Prozessor wurden die Manipulation von Headerfeldern, das Entfernen oder Hinzufügen von Daten in Pakete und die Berechnung von neuen Prüfsummen identifiziert. Um diese Aufgaben durchzuführen, besteht der Post-Prozessor aus einer Pipeline von relativ feingranularen Einheiten zur Datenmanipulation. Das können z.B. sein:

- Delete-Unit
- Insert-Unit
- Replace-Unit
- Increment- / Decrement-Unit
- IPv4 Checksum-Unit

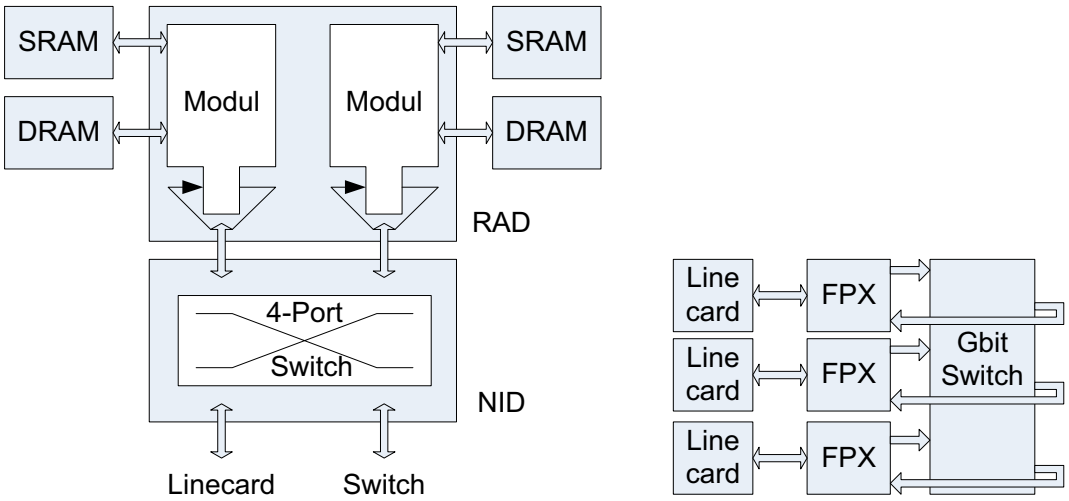
Die Einheiten werden von Paketen als nacheinander durchlaufen. Dabei manipulieren sie die Pakete. Jede Einheit benötigt immer die gleiche Zeit zur Verarbeitung, wodurch ein konstanter Durchsatz gewährleistet ist. Jedes Modul kann in Grenzen programmiert werden. Jedes Datenpaket durchläuft die Einheiten. Zeitgleich liegt ein Befehlssatz an den Einheiten an, der Context Information Output. Es handelt sich dabei um einen Befehlssatz, der die Funktion der Einheiten steuert bzw. die notwendigen Daten bereitstellt, um adäquate Manipulationen vorzunehmen. Beispielsweise stellt der Context Information Output eine DST-MAC bereit, wenn die DST-MAC des Paketes ersetzt werden soll. Der Post-Prozessor verbraucht insgesamt 2050 Slices auf einem Virtex-II-Pro und ist in der Lage, jedes Paket mit einer Verzögerung von nur 12 Takten zu verarbeiten. Ein Demonstrator ist in der Lage, 25.32 kpps[10] von Frames minimaler Größe zu verarbeiten.

3.1.3.2 Paketverarbeitung mit modularen Komponenten einer rekonfigurierbaren Hardware

Der in [LTT00] und [LNT+01] von Lockwood et. al. vorgestellte *Field Programmable Port Extender (FPX)* ermöglicht schnelle Paketverarbeitung als Erweiterung eines Netzwerk-Switches. Der FPX besteht im Kern aus zwei Teilen: einer Komponente die als Netzwerkschnittstelle dient (Network Interface Device – NID) und einem rekonfigurierbaren System, in das mehrere unterschiedliche Funktionalitäten integriert werden können (Reprogrammable Application Device – RAD). Der NID kommuniziert mit dem RAD mittels

[10] kpps : kilo packets per second

einer Schnittstelle, die Utopia ähnelt und nutzt dabei zwei seiner vier Ports. Die beiden anderen Ports dienen der Verbindung des FPX mit einer Linecard und einem Switch. An diesen Switch können so mehrere FPX angebunden sein (Abbildung 27b).



a) Aufbau des FPX Moduls.

b) Zusammenschaltung der FPX Module in einem größeren System.

Abbildung 27 - Aufbau des FPX-Systems nach [LNT+01].

In einer einfachen Konfiguration können zwei Funktionsmodule in den RAD integriert werden (Abbildung 27). Jedes Modul des RAD ist mit einem SDRAM und einem SRAM verbunden. Insgesamt steuern die Module vier unterschiedliche Speicherbanken. Sollen mehr Module zum Einsatz kommen, müssen diese mittels einer separaten Chip-internen Kommunikationsschnittstelle miteinander verbunden werden und sich den Zugriff auf die Speicherbänke teilen, was der Performance abträglich ist.

Durch die Möglichkeit beliebige Funktionen in den Modulen des RAD zu implementieren, kann der FPX flexibel für unterschiedliche Aufgaben im Bereich der Paketverarbeitung eingesetzt werden. Nachteilig ist, dass bestimmte Funktionen, falls notwendig, auf jedem FPX implementiert werden müssen.

3.1.3.3 Rekonfigurierbarer Coprozessor für Netzwerkprozessoren

In [AFK+06] stellen Albrecht et. al. eine FPGA-basierte Coprozessorarchitektur vor, die es ermöglicht, Funktionalitäten in Hardware auszuführen, die auf einem herkömmlichen Netzwerkprozessor nicht mit der notwendigen Geschwindigkeit ausgeführt werden können. In Verbindung mit einem Netzwerkprozessor kann *DynaCORE* beliebige Funktionserweiterungen anbieten, wodurch sich Flexibilität und Leistungsfähigkeit des NPs erhöhen. DynaCORE ist dynamisch rekonfigurierbar, wodurch eine große Anzahl von Hardwarefunktionalitäten flexibel unterstützt werden kann. Die Verbindung von NP und Coprozessor ermöglicht sowohl eine schnelle Protokollverarbeitung (durch den NP) als auch eine schnelle Verarbeitung von

speziellen Funktionen der Paketverarbeitung (durch den Coprozessor). Das können z. B. Aufgaben im Bereich der Verschlüsselung oder der Virensuche sein.

Der Coprozessor besteht aus einem Dispatcher, einem Konfigurationsmanager sowie verschiedensten sogenannten Hardware-Assists (HA), welche spezielle Funktionen in Hardware ausführen können (Abbildung 28).

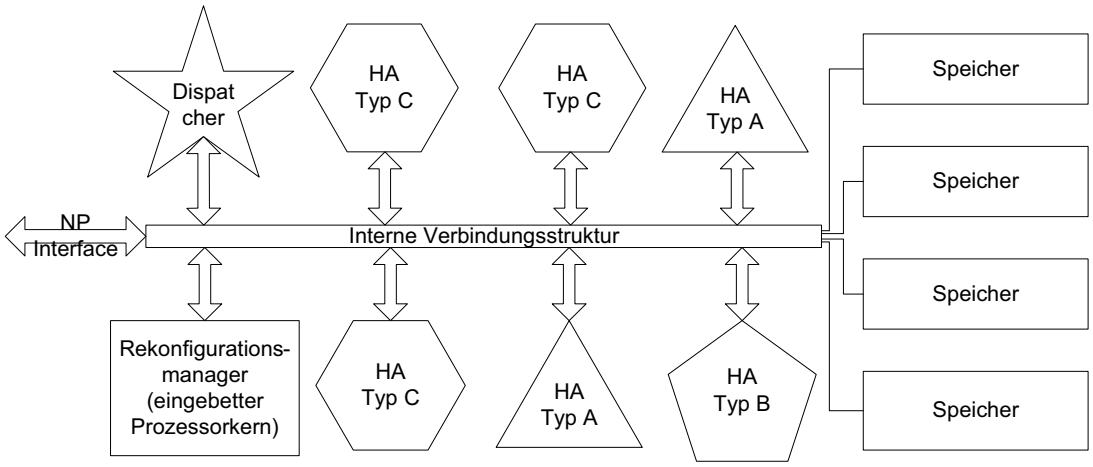


Abbildung 28 - DynaCORE Architektur (nach [AFK+06]).

Der Dispatcher hat dabei die Aufgabe, jedes ankommende Datenpaket an den entsprechenden HA weiterzugeben, wo das Paket dann verarbeitet wird. Ein HA ist ein Container für eine spezielle Verarbeitungslogik. Er kann ein Prozessor-basiertes System oder einen speziellen Hardwareblock enthalten. Der Rekonfigurationsmanager ist eine komplexe Software, die auf einem eingebetteten Prozessor läuft. Es entscheidet, welche Funktionalität bestimmten HAs zugewiesen wird, um der Verarbeitungsaufgabe am besten gerecht zu werden. Als interne Verbindungsstruktur wurden von den Autoren zwei Alternativen untersucht. Eine der Architekturen ist Pipeline-basiert. Sie ist sehr ressourcensparend. Sie hat jedoch den großen Nachteil, dass es nicht möglich ist, ein Paket nacheinander von unterschiedlichen oder auch mehrfach von ein und demselben HA verarbeiten zu lassen. Eine alternative Architektur nutzt ein Network-on-Chip zur Verbindung der Komponenten. Vorteilhaft hier ist die mögliche Flexibilisierung des Datenflusses. Datenpakete können beliebig zwischen den HAs übertragen und verarbeitet werden. Von Nachteil ist jedoch der große Implementierungsaufwand für ein NoC und dessen Hardwarekosten.

3.1.4 Zusammenfassung

Besonders im Bereich der Netzwerkprozessoren existieren viele Architekturen, auch von kommerziellen Herstellern. Die Prozessoren von Intel und Motorola werden aktuell von der Industrie eingesetzt. Alle basieren auf RISC-Kernen, die um spezielle Instruktionen erweitert wurden. Zusätzlich beinhalten die Architekturen Coprozessoren oder Hardwaremodule zur

Beschleunigung spezieller Aufgaben. Sie alle sind aber nur begrenzt performant und nicht in der Lage, Datenraten von 4 Gbit/s zu verarbeiten. Im Bereich der ASICs wird versucht, Flexibilität durch den Einsatz von programmierbaren Elementen und Leistung durch eine starke Parallelisierung dieser zu erzielen. Es können hohe Leistungen erzielt werden, jedoch sind enorme Hardwarekosten die Folge. Der GigaNetIC verwendet z. B. viele N-Cores in einem NoC. Eine interessante, weil flexible Lösung ist die PSM, die FSMs, wo benötigt, durch ihr programmierbares Pendant ersetzen kann. Die Programmierbarkeit ist allerdings begrenzt und kann unter Umständen nicht jede Funktionalität abbilden. Gleiches gilt für die vorgestellte programmierbare Processing Engine. Die FPGA-basierten Lösungen sind auf Grund ihrer Flexibilität sehr vielversprechend, ermöglichen sie doch die einfache Entwicklung und Integration spezielle Hardwarefunktionen. Der rekonfigurierbare Coprozessor ist sogar in der Lage, sich dynamisch umzukonfigurieren.

3.2 Paketklassifizierung

Als integraler Bestandteil von paketverarbeitenden Systemen ist die Paketklassifizierung ein Problem, mit dem sich viele Forscher intensiv beschäftigen. Ein naiver Ansatz einer Paketklassifizierung wäre das Speichern der Regel für jeden möglichen Schlüssel. Der Schlüssel diente dann direkt als Speicheradresse. Das würde in genau einem Speicherzugriff zur Suche und einer Speicherkomplexität von $O(2^w)$ (w ist die Bitbreite des gesuchten Schlüssels) resultieren. Für das Longest Matching Prefix-Problem bedeutet das bei IPv4 2^{32} Einträge zu 4 Byte im Speicher. Dafür sind mindestens 16 GByte Speicher notwendig. Dieser Speicheraufwand ist derzeit für schnelle SRAMs inakzeptabel. Für das Suchen nach MAC-Adressen oder bei IPv6 ist eine Speicherkomplexität von $O(2^w)$ bei 2^{48} bzw. 2^{128} Einträgen ausgeschlossen. Deshalb sind Algorithmen und Architekturen notwendig, die den Speicherbedarf für eine Klassifizierung begrenzen und dennoch schnell arbeiten. Es gibt eine große Zahl von Algorithmen und Architekturen, die sich mit der effizienten Klassifizierung von Paketen auseinandersetzen. Einige wichtige, die sich mit der Suche nach dem LMP in IP-Routern befassen, sollen als Beispiele für die einfache Suche auf einem einzelnen Feld dienen. Vor allem Hardwarearchitekturen und auf Hashfunktionen basierende Architekturen werden im folgenden Abschnitt präsentiert.

3.2.1 Softwarelösungen

Klassische Algorithmen zur Paketklassifizierung sind baumbasierte Softwarealgorithmen. Neben der klassischen linearen oder binären Suche, die lineare bzw. logarithmische Suchkomplexitäten aufweisen, wurden vor allem in der Vergangenheit komprimierte Bäume zur Speicherung der Suchdatenbanken eingesetzt. Diese haben einen unterschiedlichen Aufbau und unterschiedliche Eigenschaften.

3.2.1.1 Radix Trie

Ein Radix Trie (kurz: Trie[11]) ist ein binärer Baum, der für eine bestimmte Bitkette durchlaufen wird (Abbildung 29). Dabei wird auf der *i*-ten Ebene des Baumes anhand des *i*-ten Bits der gesuchten Bitkette entschieden, ob der restliche Baum links „0“ oder rechts „1“ durchsucht werden soll. Ein Knoten auf der *i*-ten Ebene repräsentiert jeweils die von Bit 0 bis i durchlaufe Bitkette. Auf der Suche nach einem LMP wird nun solange in den Trie hinabgestiegen, bis ein Blatt gefunden wird. Die Bitkette auf der richtigen Seite in dem Blattknoten stellt dann den LMP für die gesuchte Bitkette (die IP-Adresse) dar. Ein Trie hat eine Tiefe von maximal *w*. Damit ergibt sich für eine Suche eine Komplexität von $O(w)$. Im Falle von IPv4 ist $w = 32$. Die Suche nach einem Präfix kann also bis zu 32 Taktzyklen beanspruchen. Um *n* Präfixe der Breite w zu speichern, beträgt die Speicherkomplexität $O(nw)$.

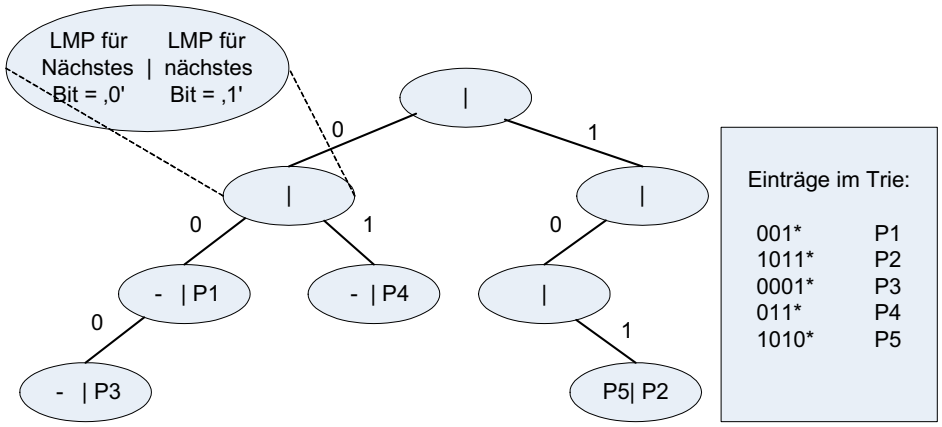


Abbildung 29 - Darstellung eines einfachen Tries und der Position von fünf Präfixen (P1 – P5).

3.2.1.2 PATRICIA Tree

Der PATRICIA[12] Tree [Mor68] ist eine vom Trie abgeleitete Baumstruktur (Abbildung 30). Der Unterschied ist, dass ein Patricia Tree keine Knoten ersten Grades hat. Entweder hat ein Knoten zwei Nachfolger oder er ist ein Blatt. Eine Verkettung wird zu einem Knoten komprimiert. Das hat zur Folge, dass die Bits, die in der Kette komprimiert sind, nicht bei der Suche im Baum zu betrachten sind. Deshalb muss jeder Knoten die Information beinhalten, welches Bit als nächstes zu inspizieren ist, um die Entscheidung für den weiteren Suchverlauf zu treffen. Da ein Patricia Tree ein vollständiger Binärbaum ist, hat er genau n Blätter und n-1 innere Knoten. Damit ist die Speicherkomplexität $O(n)$. Präfixe für eine Suche sind in den Blättern gespeichert, die eine lineare Liste von passenden Präfixen enthalten kann. Bei einer Suche ist im Baum abzusteigen, bis ein Blatt gefunden wurde. Stimmen die dort gefundenen

[11] Trie ist abgeleitet von re**trie**val wird aber ausgesprochen wie das englische Wort „try“.

[12] PATRICIA – **P**ractical **A**lgorithm **T**o **R**etrieve **I**nformation **C**oded **I**n **A**lphanumeric

Präfixe mit dem gesuchten überein, ist die Suche beendet. Falls nicht, muss rekursiv zum Elternknoten zurückgekehrt werden, um noch den zweiten Nachfolger zu überprüfen. Das ist notwendig, da während des Abstiegs nicht alle Bits des gesuchten Präfix überprüft wurden. Durch die Rekursion ergibt sich im schlechtesten Fall eine Suchkomplexität von $O(w^2)$. Der Speicheraufwand ist geringer, der Suchaufwand kann jedoch größer sein als beim Radix Trie. Abbildung 30 stellt den Patricia Tree des bereits für den Trie genutzten Beispiels dar.

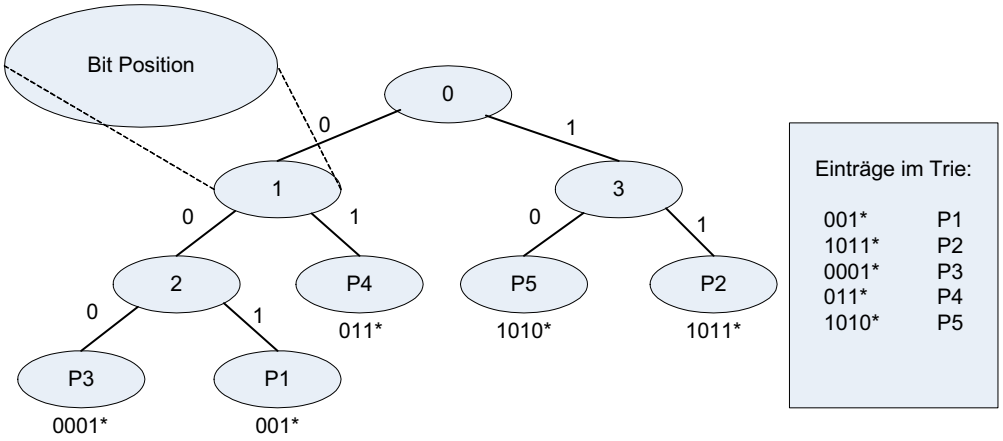


Abbildung 30 - Darstellung eines PATRICIA Trees und der Position von fünf Präfixen (P1 – P5).

Andere Algorithmen, die ähnlich dem PATRICIA Tree auf einer Komprimierung der Pfade basieren, stellen z. B. Gwehenberger in [Gwe68] und Sklower in [Skl93] vor. Auch gibt es Entwicklungen, die einen n-fachen statt einen Binärbaums nutzen [Sri99].

3.2.2 Existierende Hardwarelösungen

Da Softwarelösungen nicht mehr in der Lage sind, aktuelle Leistungsanforderungen befriedigend zu erfüllen, sind vor allem Hardwarelösungen das Mittel der Wahl, um eine leistungsstarke Paketklassifizierung zu ermöglichen.

3.2.2.1 Inhaltsadressierbare Speicher

Ein inhaltsadressierbarer Speicher (Content Addressable Memory – CAM) oder Assoziativspeicher ist die Hardwareimplementierung einer vollständig parallelen Suche. CAMs werden nicht durch das Anlegen einer Adresse angesteuert sondern durch das Anlegen des Suchschlüssels. Der CAM liefert den zu diesem Schlüssel abgelegten Wert zurück. Dieser Wert kann entweder bereits die gesuchte Klassifizierungsregel oder deren Adresse in einem herkömmlichen Speicher. CAMs habe jedoch entscheidende Nachteile. Zusammen mit den eigentlichen Speicherelementen ist die komplette Suchlogik mit zu implementieren. Diese benötigt sehr viele Hardwareressourcen. Die Speicherdichte eine CAMs ist deshalb sehr gering. Das hat zur Folge, dass unter anderem der Energieverbrauch von CAMs sehr hoch ist. Des Weiteren sind CAMs pro gespeichertem Bit wesentlich teurer als DRAMs, aber auch als

SRAMs. Es gibt allerdings viele Bestrebungen, die Energieeffizienz von CAMs zu verbessern [ZNB03], [ZHL04].

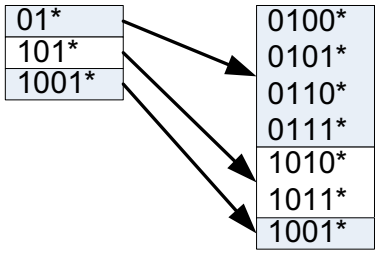
Man unterscheidet zwei verschiede Arten von CAMs, binäre und ternäre CAMs. Binäre CAMs speichern jeden Schlüssel direkt ab und vergleichen die gespeicherten Muster mit einem angelegten Wort auf Gleichheit. Sie können also zur Implementierung eines „Exact Match" genutzt werden. Ternäre CAMs (TCAMs) hingegen bieten für jedes zu speichernde Bit eines Schlüssels zwei Bit Speicherplatz an. Dadurch kann zusammen mit dem Schlüssel für jedes Bit ein „don't care" Bit abgespeichert werden. Das ermöglicht die Implementierung von LMP-Lösungen auf der Basis von TCAMs. TCAMs sind wie alle CAMs im Vergleich zu herkömmlichen SRAM-Chips sehr groß, langsam und weisen eine sehr hohe Leistungsaufnahme auf. Die Verhältnisse sind in Tabelle 2 dargestellt.

Tabelle 2- Vergleich der TCAM und SRAM-Technik [JP08].

Speicherchip	TCAM (18 Mbit)	SRAM (18 Mbit)
Maximale Taktrate (MHz)	266	400
Zellgröße (Anzahl der Transistoren pro Bit)	16	6
Leistungsaufnahme (W)	12-15	≈ 0,1

3.2.2.2 Suche mit mehreren Speicherzugriffen

Srinvasan und Vargehese stellen in [SV98] die *Controlled Prefix Expansion* vor. Um den Präfix einer 32 Bit breiten IP-Adresse zu finden, sind im Normalfall bis zu 32 Suchen im Speicher nach existenten Präfixen durchzuführen. Dabei wird zuerst in den 32 Bit breiten Präfixen gesucht. Bei Misserfolg muss absteigend in allen Teiltabellen mit Präfixen geringerer Länge nach einem passenden Eintrag gesucht werden. Um die Anzahl von Präfixen unterschiedlicher Länge in der Datenbasis zu verringern, wird die Controlled Prefix Expansion eingesetzt. Dabei werden Präfixe erweitert, so dass in einer Datenbank nur noch Präfixe weniger definierter Längen existieren (Abbildung 31a).



a) Präfixerweiterung auf ausschließlich 4 Bit breite Präfixe

Anzahl unterschiedlicher Präfixe	Anzahl der Tabellen-einträge
24	230766
13	285223
7	483877
4	2179717

b) Anzahl der Einträge einer realen Routingtabelle bei einer Präfixerweiterung auf unterschiedliche viele Präfixe

Abbildung 31 - Controlled Prefix Expansion.

Das hat zur Folge, dass nur noch wenige Präfixlängen nach einem Exact Match durchsucht werden müssen, erst die langen, danach die kurzen Präfixe. So kann die Suche nach dem LMP auch durch mehrere Exact Match-Suchen durchgeführt werden. Ein Nachteil dieser Methode ist jedoch das Anwachsen der Datenbank, denn soll beispielsweise eine Expansion eines 19 Bit Präfixes auf 24 Bit erfolgen, sind 32 24 Bit Präfixe notwendig, um den 19 Bit Präfix abzubilden. Je geringer die Anzahl der unterschiedlichen erlaubten Präfixlängen ist, desto mehr Speicher ist notwendig (Abbildung 31b). Die Suchkomplexität dieses Ansatzes hängt von der Bitbreite des zu suchenden Schlüssels *w* und der Anzahl Bits *a* von einer Bitstufe zur nächsten Bitstufe ab $(O(w/a))$, während der Speicheraufwand quadratisch mit *a* skaliert.

3.2.2.3 2-stufige Speichersuche

Gupta et.al. stellen in [GLM98] mit der *DIR-24-8 Architektur* eine Hardwarearchitektur vor, die mit maximal zwei Suchzugriffen auskommt, um eine Suche nach dem Präfix einer IP-Adresse durchzuführen. Ausgehend von der Annahme, dass DRAM in großen Mengen zur Verfügung steht, da er relativ preiswert ist, ist der Algorithmus so konzipiert, diesen extensiv zu nutzen. Der DIR-24-8 zugrundeliegende Algorithmus arbeitet in zwei Stufen (Abbildung 32).

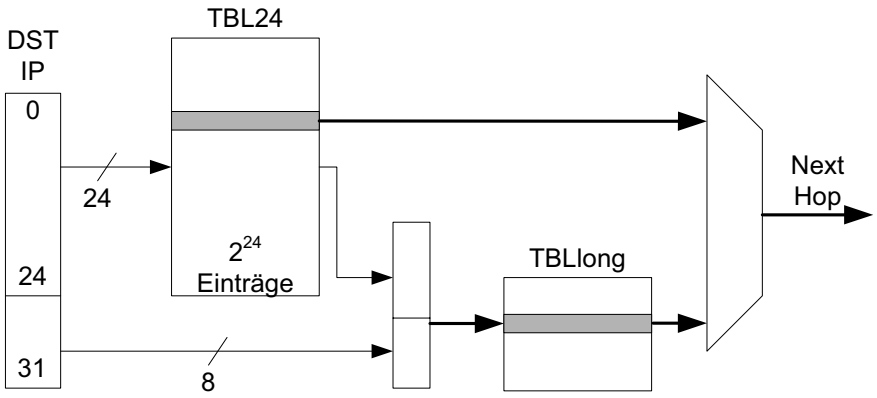


Abbildung 32 - DIR-24-8 Architektur [GLM98]. Das Klassifizierungsergebnis ergibt sich entweder direkt aus TBL24 oder aus TBLlong.

Im ersten Schritt nutzt er die ersten 24 Bit der IP-Adresse als direkten Index in einen 2^{24} Einträge fassenden Speicher (TBL24). Alle Regeln für Präfixe ≤ 24 Bit sind darin direkt abgelegt. Existieren Präfixe > 24 Bit, enthält die TBL24 einen Index für eine zweite Tabelle (TBLlong). In dieser zweiten Tabelle sind an den Adressen $Index \cdot 256$ bis $Index \cdot 256 + 255$ für alle möglichen 32 Bit Präfixe die korrespondierenden Einträge abgelegt. Es ist damit möglich, alle Präfixe mit bis zu 24 Bit durch einen Lesevorgang zu ermitteln. Die anderen Präfixe benötigen 2 Lesevorgänge, noch dazu in unterschiedlichen Speichern, was ein Pipelining ermöglicht. Damit stellt der Algorithmus eine Anwendung der Controlled Prefix Expansion auf nur noch zwei Präfixlängen (24 und 32 Bit) dar. Der Algorithmus hat allerdings

den Nachteil, dass sehr viel Speicherplatz notwendig ist. TLB24 allein benötigt 32 MByte Speicher. Die Größe von TBLlong hängt von der Anzahl der Präfixe > 24 Bit mit unterschiedlichen ersten 24 Bit ab. Für alle Präfixe, die sich in den ersten 24 Bit unterscheiden und größer als 24 Bit sind, sind in TBLlong 256 Einträge vorzusehen. Gupta geht von insgesamt 33 MByte aus, weil der Anteil von Präfixen > 24 Bit sehr klein sei. In [Hus01] wird allerdings gezeigt, dass der relative Anteil langer Präfixe im Verhältnis zu allen Präfixen ansteigt. Damit steigt die Größe von TBLlong stärker an. Sehr große Speicher können nur mit kostengünstigen, aber langsamen DRAMs implementiert werden, was die Lookupperformance beeinträchtigt. Veränderungen der Routingtabelle sind sehr kostenintensiv, da unter Umständen viele Einträge verändert werden müssen. Außerdem ist der Algorithmus nur sehr schwer auf das zukünftige IPv6 mit seinen 128 Bit langen Adressen anwendbar.

Der von Huang und Zhao in [HZ99] vorgestellte Algorithmus ähnelt dem von Gupta. Huang und Zhao teilen IP-Adressen in zwei Teile. Sie nutzen ein 16 Bit breites Segment und einen 16 Bit breiten Offset. Die ersten 16 Bit einer IP adressieren einen Speicher mit 64k Einträgen in dem entweder bereits Portinformationen (Next Hop) stehen, wenn der gesuchte Präfix kleiner oder gleich 16 Bit ist oder die Adresse eines zugehörigen *next hop arrays – NHA*. Im NHA, der 16k Einträge enthält, dienen dann die zweiten 16 Bit der Adresse als Index, um die Portinformation zu finden. Abbildung 33 stellt die Architektur schematisch dar.

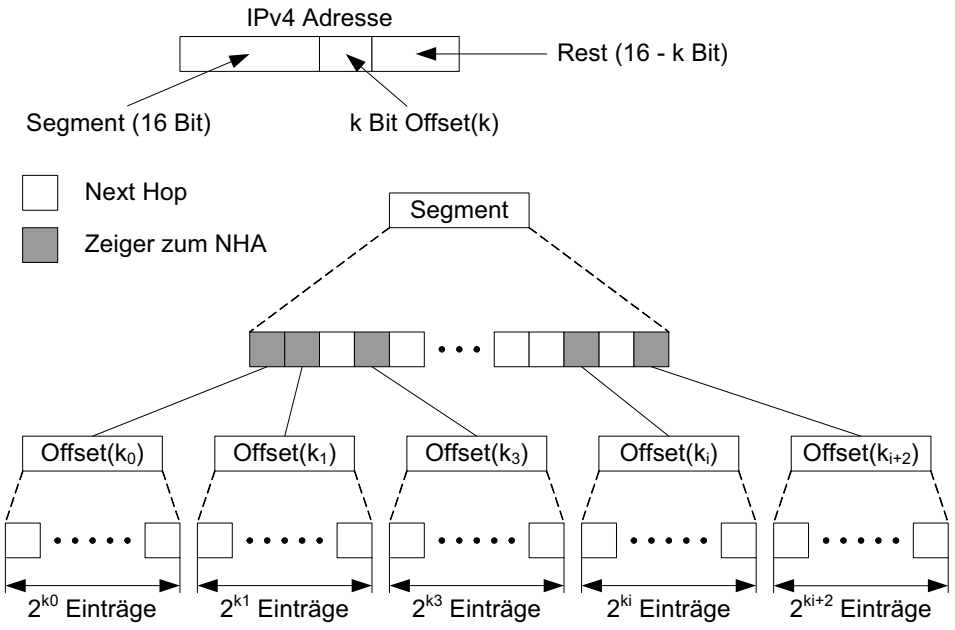


Abbildung 33 - Indirect Lookup-Mechanismus mit variabler Offsetlänge (nach [HZ99]).

Mit dieser Architektur sind Suchanfragen mit maximal zwei Speicherzugriffen möglich. Um Speicher einzusparen, werden nur dann NHAs angelegt, wenn für ein bestimmtes Segment auch Präfixe > 16 Bit existieren. Des Weiteren enthält eine NHA nur so viele Einträge wie notwendig

sind, um die entsprechenden Präfixe abzuspeichern. Hat der größte LMP innerhalb einer NHA beispielsweise 19 Bit, müssen insgesamt nur 8 Einträge in der NHA vorhanden sein. Es sind nur $2^k = 2^{19-16} = 8$ Einträge notwendig, womit sich der Offset auf $16 - k$ verkleinert. Zusätzlich nutzen die Autoren eine Datenkompression, um NHAs mit mehr als 8 Einträgen zu verkleinern. Als Ergebnis steht ein System, das zwischen 1 und 2 Speicherzugriffen benötigt, aber verhältnismäßig viel Speicher verbraucht. Der Speicherbedarf hängt von der Anzahl und Verteilung von Präfixen > 16 Bit ab. Die Autoren zeigen ein Beispiel, dass bei Unterstützung von 4000 Segmenten mit langen Präfixen $(24 < Länge \leq 32)$ zwischen 1,5 und 2 MB Speicher notwendig sind.

3.2.2.4 Baumbasierte Hardwarelösungen

Die von Pao et. al. in [PLW+03] vorgestellte Architektur basiert auf der Suche in einem binären Baum, dessen Teile parallel durchsucht werden können. Die Parallelisierung wird durch die Partitionierung des Binärbaumes ermöglicht. Dabei wird der vollständige Binärbaum, der für IPv4 eine Tiefe von 32 hat, in sich nicht überlappende Unterbäume aufgeteilt, die jeweils 255 Knoten haben (Abbildung 34). Die Baumstruktur ist in vier Ebenen unterteilt. Ebene 0 umfasst die Bits 1-8 und besitzt zwei Unterbäume mit je 255 Einträgen. Hier sind Präfixe bis 8 Bit Länge gespeichert. Ebene 1 von Bit 9-16 hat 512 Unterbäume. Hier sind Präfixe bis 16 Bit Länge gespeichert. Ebene 2 hat 128k (Präfixe bis 24 Bit) und Ebene 3 32M Unterbäume (Präfixe mit bis zu 32 Bit Länge).

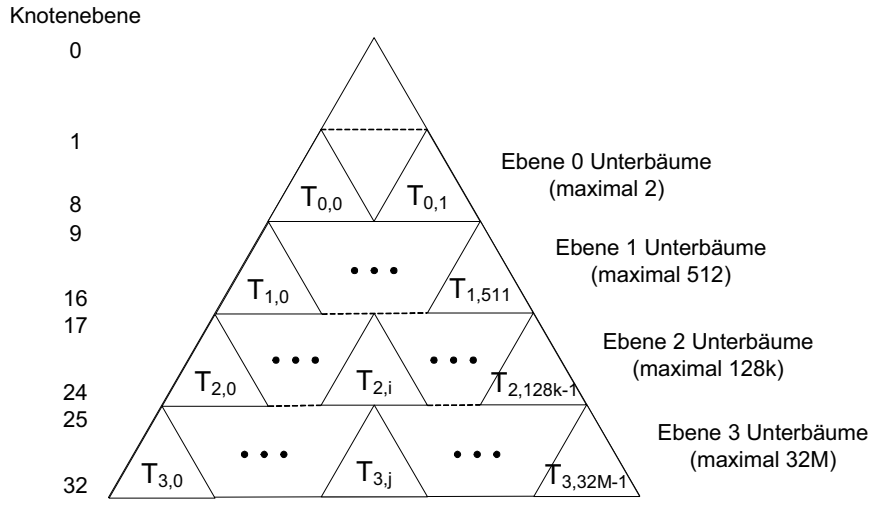


Abbildung 34 - Partitionierung des Binärbaums in Unterbäume mit 255 Knoten [PLW+03].

Nur wenn es in einem Unterbaum mindestens ein Präfix gibt, wird der Baum auch abgespeichert. Besonders auf Ebene 3, auf der Präfixe zwischen 24 und 32 Bit Länge gespeichert sind, ist diese Zahl relativ gering. Für die Ebenen 1-3 wird ein Indexblock (IB1-

IB3) erzeugt, der die Adressen der verschiedenen Bäume der Ebene enthält. Zur Suche werden die Ebenen 0 bis 2 parallel durchsucht:

- Ebene 0: Das erste Bit des Präfixes definiert den Unterbaum, der durchsucht werden muss. Dieser ist durch einen 128 Bit Vektor (*tree-vector*) repräsentiert, in dem existierende Präfixe mit einer 1 gekennzeichnet sind. Die Bits 2-8 des Präfixes können auch als 128 Bit Vektor gekennzeichnet werden (*mask-vector*). Die am weitesten rechts liegende '1' in einem Ergebnisvektor, der durch eine bitweise AND-Verknüpfung der beiden Vektoren entsteht, gibt den LMP an. Der eigentliche Hop (die Routinginformation) ist in einem *routing-vector* gespeichert.
- Ebene 1: Die Bits 1-8 definieren einen von 512 Unterbäumen, dessen physische Speicheradresse aus IB1 ermittelt wird. Der *tree-vector* des richtigen Baums wird dann mit dem *mask-vector* aus den Bits 9-16 verknüpft.
- Ebene 2: Das Vorgehen ist analog zum Vorgehen auf Ebene 1.
- Ebene 3: Diese Ebene ist nur schwach besetzt. Deshalb ist IB3 in 128k je 256 Einträge fassende Segmente unterteilt, deren Adressen in IB2 gespeichert werden. Bei der Suche auf Ebene 2 kann somit festgestellt werden, ob noch Einträge der Ebene 3 vorhanden sind. Wenn der Eintrag in IB2 auf Basis der Adressen 1-17 nicht Null ist, muss der korrespondierende Ebene 3 Unterbaum durchsucht werden.

Durch Pipelining und Parallelisierung kann erreicht werden, dass in jedem Takt ein Präfix gefunden werden kann. Damit hängt die Performance vom eingesetzten Speicher ab. Der Speicheraufwand hängt von Anzahl und Verteilung der Präfixe ab. Für jedes Paar von *tree-vector* und *routing-vector* benötigt die Architektur 255x9 Bit. Beispieltabellen benötigten Speichermengen von 1,63 bis 2,5 MB bei Tabellen zwischen 16445 und 41811 Einträgen. Es ist also zu erwarten, dass sehr große Tabellen mit mehr als 128k Einträgen relativ viel Speicher benötigen. Dieser muss als schneller SRAM ausgelegt sein, damit das System zufriedenstellend arbeitet. Eine Umsetzung auf IPv6 erscheint schwierig. Dort wären 14 Ebenen von Unterbäumen nötig. Die Umsetzung ist Forschungsgegenstand der Autoren.

Methrotha und Franzon verwenden in ihrer Hardwarearchitektur ebenfalls einen Trie zur Suche [MF02]. Im Gegensatz zu anderen Ansätzen versuchen sie allerdings die Hardwarekosten zu senken, indem sie nicht wie üblich in den Blättern des Tries die Routinginformation oder eine Adresse für die Routinginformation speichern. Beim Durchlaufen des Tries berechnet der Algorithmus die Adresse, an der die Routinginformation in einem externen Speicher steht. Dazu wird beim Aufbau des Tries die Routingtabelle in aufsteigender Ordnung sortiert und zwar derart, dass Präfixe kürzerer Länge als kleiner betrachtet werden als längere Präfixe. Dann kann der Trie aufgebaut werden. Zuerst wird der Wurzelknoten erzeugt und seine Nachfolger werden mit Null initialisiert. Dann wird jeder Eintrag aus der sortierten Liste mit den Routinginformationen nacheinander gelesen und gegebenenfalls erweitert, um den Trie aufzubauen (Abbildung 35).

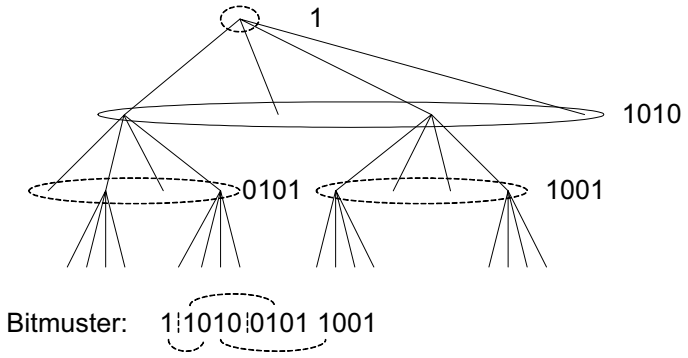


Abbildung 35 - Beispielhafter Trie vierten Grades mit dem korrespondierenden Bitmuster [MF02].

Der Trie wird dann im SRAM abgelegt, indem wie bei einer Breitensuche[13] (Breadth-First-Search – BSF) vorgegangen wird. Wird nun ein Präfix gesucht, ist der folgende Algorithmus zu durchlaufen:

1. Der Startzeiger wird mit den ersten *g* Bit aus dem SRAM initialisiert.
2. Die ersten $\log_2(g)$ Bits des Präfixes werden gelesen. Diese dienen als Offset für den Startzeiger.
3. Wenn das Bit, auf das der Offset zeigt, eine '1' ist, wird der Startzeiger zur nächsten Ebene bewegt. Diese Position wird durch die Summe der Einsen in der aktuellen Ebene multipliziert mit *g* gebildet. Die Schritte 1-3 werden wiederholt.
4. Wenn das Bit, auf das der Offset zeigt, eine '0' ist, ist die Suche beendet. Durch das zählen der Einsen auf der aktuellen Ebene kann die Adresse im externen Speicher, an der die Routinginformation steht, berechnet werden.

Zur Suche benötigt der Algorithmus immer $\log_g(w)+1$ Speicherzugriffe. Der Speicherbedarf im SRAM ist sehr gering, da hier keine Routinginformationen oder Adressen gespeichert werden müssen. Eine Tabelle mit 30.000 Einträgen benötigt z.B. nur 35 KB. Dazu kommt aber noch der externe Speicher, in dem die Routinginformationen gespeichert sind. Hier muss für jedes Präfix aber nur eine Speicherposition zur Verfügung stehen.

3.2.2.5 Zusammenfassung

Die hier vorgestellten Architekturen ermöglichen alle sehr gute Ergebnisse bei der Suche nach Präfixen. Mit 1-3 Speicherzugriffen, die z. T. gepipelined werden können, sind kaum Verbesserungen möglich. Methrota bildet hier die Ausnahme. Bei diesem hängt die Suchleistung von der Tiefe des Tries ab. Dafür kann diese Architektur durch geringe Speicher-

[13] Bei der Breitensuche werden die höheren Ebenen gegenüber tieferen bevorzugt. Erst wird der Knoten der 1. Ebene durchsucht, dann alle Knoten der 2. Ebene von links nach rechts, dann alle Knoten der dritten Ebene von links nach rechts usw.

anforderungen überzeugen. Die anderen Architekturen haben den Nachteil, verhältnismäßig viel Speicher zu benötigen. Das ist im Bereich des LMP-Problems für IPv4 noch nicht schwerwiegend. Jedoch ist die Anwendbarkeit der Algorithmen für Schlüssel zweifelhaft, die breiter als 32 Bit sind. Die Speicherkosten und die Suchzeit wachsen an oder die Algorithmen sind für derartige Probleme nicht anwendbar.

Tabelle 3 - Vergleich unterschiedlicher Hardwarelösungen.

Architektur	Speicherzugriffe	Speicheraufwand
CAM	1	-
Srinvasan und Vargehese (Controlled Prefix Expansion)	$(O(W/A))$	--
Gupta et.al. (DIR-24-8 Architektur)	1-2 (1 mit Pipelining)	--
Huang und Zhao	1-2	o
Pao et. al.	1 mit Pipelining	o
Methrotha und Franzon	$\log_g(w)+1$	+

3.2.3 Hashing als Mittel der Paketklassifizierung

Wird Hashing als Mittel der Paketklassifizierung eingesetzt, ist der Speicherbedarf weniger von der Schlüsselbreite abhängig. Beim Hashing können *n* 32 Bit breite Schlüssel an die gleichen Speicherstellen geschrieben werden wie *n* 128 Bit Schlüssel, da die Hashfunktion unabhängig von der Breite des zu hashenden Wertes potentiell in der Lage ist, einen Suchindex beliebiger Breite zu erzeugen. Deshalb sind hashbasierte Architekturen zur Paketklassifizierung interessante Alternativen, die das Potential haben, auch für schwierigere Klassifizierungsaufgaben wie LMP für IPv6 eingesetzt zu werden.

Hashing, aber auch CAMs und die binäre Suche sind aber grundsätzlich für den Einsatz in der Paketklassifizierung nur bedingt geeignet. Sie haben einen Nachteil, der nicht verschwiegen werden darf: Mit diesen Techniken kann nur nach einer exakten Übereinstimmung gesucht werden. Sie sind ohne Anpassungen nur in der Lage, Exact Matches zu finden (siehe Abschnitt 2.2). Range Matches oder Präfix Matches können nicht gefunden werden. Im Bereich der Paketverarbeitung im LAN (MAC-Adressen) und AN (Nutzeridentifikation) sind oftmals nur Exact Matches notwendig und hashbasierte Algorithmen ohne Einschränkung anwendbar. Es existieren aber auch Techniken, um das Hashing für LMP einzusetzen. Die vorgestellte Prefix Expansion Technik stellt zum Beispiel eine Möglichkeit dar, aus einem Prefix Match mehrere Exact Matches zu erzeugen.

3.2.3.1 Hash-Algorithmen

Hashing kann mit den verschiedensten Hashfunktionen durchgeführt werden. Es sind sehr viele Hashfunktionen für verschiedene Anwendungsgebiete bekannt. Zur Paketklassifizierung,

bei der hohe Datenraten und Geschwindigkeiten gefordert sind, eignen sich allerdings nur solche, die gute Hashergebnisse erzielen, die einfach in Hardware zu implementieren sind und bei denen der Hashalgorithmus schnell arbeitet (möglichst innerhalb eines Taktes). Nachfolgend sind einige Hashfunktionen mit derartigen Eigenschaften kurz vorgestellt.

Hashing mit Teilen der Adresse

Eine sehr naheliegende Möglichkeit, einen *m* Bit Hashwert aus einem *n* Bit breiten Schlüssel zu bilden, ist die einfache Bitselektion. Dabei werden *m* Bit des Schlüssels gewählt und bilden den Hashwert. Inwieweit die Auswahl eine gute oder schlechte Hashfunktion hervorbringt, hängt dabei von den Bits des Schlüssels ab und deren Informationsgehalt. Bei MAC-Adressen ist es nicht sinnvoll, die ersten 24 Bit als Schlüssel zu wählen, da diese den Hersteller einer Netzwerkkarte definieren und bei einem Hersteller immer gleich sind. Bei IP-Adressen wiederum enthalten die letzten ein oder zwei Byte tendenziell weniger Information, weil sie Subnetze definieren, die bei Routingentscheidungen nicht relevant sind. Der Greedy Algorithmus in [ZNB03] nutzt Adressbits zur Selektion einer Hashfunktion, die dann in einem Paketklassifizierungsalgorithmus Verwendung findet. Es werden dabei die am besten geeigneten Bits des Schlüssels gesucht. Um *m Bit*s zu finden, die den Hashwert bilden sollen, sind *m* Iterationen notwendig. Bei jeder Iteration wählt der Algorithmus dasjenige Bit aus dem Schlüssel aus, das die Schlüsselmenge aus 2^{m-1} Teilmengen am gleichmäßigsten auf die 2^m entstehenden Teilmengen aufteilt. Dadurch werden die Bits der Adresse für die Hashfunktion verwendet, die eine optimale Hashfunktion mit möglichst wenigen Kollisionen erzeugt. Der Algorithmus findet unter anderem beim *Content Addressable Random Acccess Memory* ([CMX+07]) Verwendung. Diese Architektur ist in Kapitel 3.2.3.2 näher beschrieben.

Cyclic Redundancy Check

Eine einfache Methode zu hashen ist die Berechnung des Cyclic Redundancy Check (CRC)-Wertes eines Schlüssels. Dabei wird eine Polynomdivision durchgeführt. Der Rest der Division des Schlüssels durch ein Generatorpolynom bildet dann den Hashwert. Bei der Verwendung des bekannten CRC32 [DIX82] hat die Binärdarstellung des Generatorpolynoms den Wert:

$$CRC32 = 10000010011000001000111010110111 \quad (9)$$

Der CRC ist sehr gut geeignet, viele verschiedene Fehlerarten zu erkennen und zeigt laut [Jai92] auch sehr gute Eigenschaften als Hashfunktion, da der Informationsgehalt pro Bit des CRCs bei den gemachten Untersuchungen sehr nahe an den Maximalwert 1 heranreicht.

Fletcher Checksumme

Eine weitere Methode des simplen Hashings ist die Fletcher Checksumme [Fle82]. Es handelt sich dabei um eine 16 Bit breite Checksumme, die aus einer *n* Byte langen Nachricht *b* erzeugt

wird. Sie ist sehr leicht zu berechnen. Die zwei Byte (*C(0)* und *C(1)*) der Checksumme berechnen sich folgendermaßen:

```
C[0]=0; C[1]=0;
for (i=0; i<n; i++)
{
      C[0] = C[0] + b[i];
      C[1] = C[1] + C[0];
}
```

Laut [Jai92] ist auch in diesem Fall der Informationsgehalt jedes Bits nahe 1, was auf gute Eigenschaften als Hashfunktion hinweist, da die Informationsredundanz der Hashwerte gering ist. Da die Fletcher Checksumme jedoch nur 16 Bit breit ist, sind mit ihr maximal Schlüsselmengen von 32k Einträgen zu verwenden (ausgehend von einem Belegungsfaktor von $\alpha = 0{,}5$).

XOR Methode

Bei der XOR Methode wird ein *n* Bit Schlüssel in *k* Segmente unterteilt. Die Bits in jedem der Segmente werden XOR-verknüpft, um eine Hashadresse zu erzeugen. Soll beispielsweise ein 10 Bit breiter Schlüssel in einen 3 Bit Hashwert transformiert werden, könnte eine mögliche Hashfunktion folgendermaßen aussehen:

$$h(x) = (x_0 \oplus x_4)(x_1 \oplus x_8)(x_2 \oplus x_6) \tag{10}$$

Es müssen nicht immer alle Bits des Schlüssels für die Hashfunktion verwendet werden. Problematisch ist die Auswahl der optimalen Bits. Unterschiedliche Bits können gute oder weniger gute Hashfunktionen erzeugen.

Hashfunktionen der Klasse H_3

Hashfunktionen der sogenannten Klasse H_3 wurden bereits 1981 von Carter und Wegman definiert [CW81]. Sie zeichnen sich durch gute Eigenschaften und leichte Implementierbarkeit in Hardware aus. Sie basieren auf der Menge Q aller möglichen booleschen Matrizen des Formats (*i,j*). Der Inhalt der Matrix definiert eine bestimmte Hashfunktion h_q, die einen i Bit breiten Schlüssel auf einen j Bit breiten Hashwert abbildet. Für eine bestimmte Matrix $q \in Q$ ist $q(k)$ der *k*-te Bitstring der Zeile k der Matrix. Er hat die Länge j. Das *k*-te Bit des Schlüssels *x ist* $x(k)$. Dann ist die Hashfunktion $h_q(x)$ wie folgt definiert:

$$h_q(x) = [x(1) \cdot q(1)] \oplus [x(2) \cdot q(2)] \oplus \ldots \oplus [x(i) \cdot q(i)] \tag{11}$$

Durch Nutzung einer anderen Matrix *q* ergeben sich andere Hashfunktionen. Die in [SDT+05] vorgestellte Architektur für die Paketklassifizierung nutzt Hashfunktionen der Klasse H_3. Abbildung 36 veranschaulicht die Erzeugung des Haschwertes.

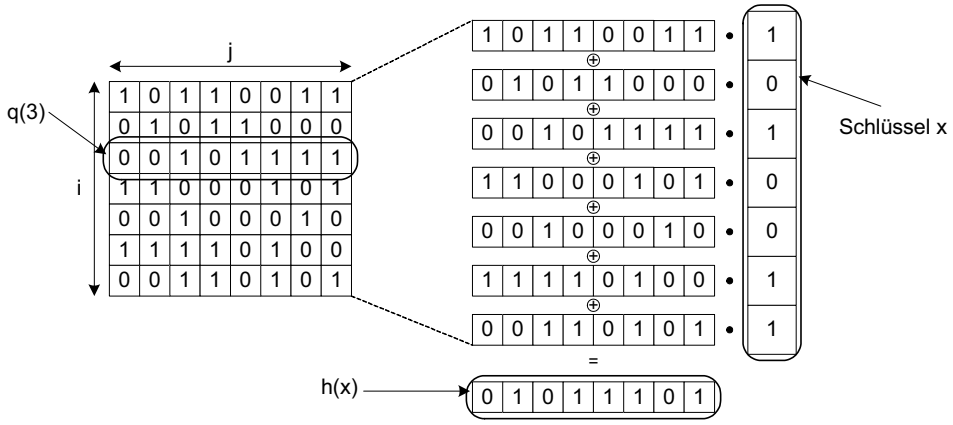


Abbildung 36 - Hashfunktionen der Klasse H_3.

Intel IXP Netzwerkprozessor Hash Architektur

Der IXP-1200 Netzwerkprozessor enthält zur Unterstützung von Berechnungen einen dedizierten Hardwareblock, der in der Lage ist, 48 und 64 Bit breite Werte zu hashen. Wie aus [Int01a] und [Int01b] hervorgeht, handelt es sich um eine auf Multiplikation basierende Hashfunktion. Der Multiplikator ist programmierbar. Das Ergebnis der Multiplikation wird durch ein nicht reduzierbares festes Generatorpolynom dividiert. Der sich ergebende Rest stellt den Hashwert dar. Dieser hat die gleiche Bitbreite, wie der Schlüssel (48 bzw. 64 Bit). Zum Hashen eines Wertes sind sechs (48 Bit Schlüssel) oder acht (64 Bit Schlüssel) Takte notwendig. Der IXP-2400 wird in [SR06] zur Paketklassifizierung in einem Network Intrusion Detection System (NIDS) eingesetzt. Dabei wird für die Klassifizierung des SRC- und DST-IP Paares der integrierte Hashblock verwendet.

Multiplikation ohne Carry

Broder und Karlin schlagen in [BK90] Hashfunktionen für eine Tabellengröße von 2^j der Form

$$\left[\left(A \cdot k + B\right) \bmod P_1\right] \bmod P_2 \tag{12}$$

vor, wobei *k* der Schlüssel der Länge $d+1$ Bit ist. A und B sind zufällige Polynome der Länge *d* Bit. P_1 ist ein konstantes, nicht reduzierbares Polynom des Grades $d+1$ und P_2 ein konstantes Polynom des Grades *j*. In der Praxis können laut den Autoren auch Funktionen der Form

$$(A \cdot k \bmod P_1) \bmod t^j \tag{13}$$

verwendet werden, ohne Leistung einzubüßen. Eine Hardwareimplementierung der Funktion ist unter Nutzung eines Multiplizierers für die Multiplikation und XOR-Gattern für die Addition möglich. P_1 kann als Shiftback-Register implementiert werden. t^j entspricht einer Maskierung.

Das Polynom *A* kann zum Beispiel durch einen Pseudozufallszahlengenerator erzeugt werden. Eine Hardwareimplementierung ist aufgrund der notwendigen Multiplikation aufwendig, aber vertretbar.

Message Digest 5 (MD5)

MD5 [Riv92] ist ein bekanntes Hashverfahren, das insbesondere im Bereich der Signierung von Nachrichten Verwendung findet. Dabei kann der Algorithmus aus einer Nachricht mit 512 Bit Länge einen 128 Bit breiten Hashwert erzeugen. Er läuft folgendermaßen ab:

1. Die Nachricht wird aufgefüllt, so dass sie kongruent zu 448 mod 512 ist. Es wird eine '1' und der Rest '0' angehängt. Mindestens ein Padding-Bit und maximal 512 Padding-Bits sind anzuhängen.
2. die 64 Bit breite Repräsentation der Länge der Nachricht wird an die Nachricht angehängt. Damit hat die Nachricht die Länge eines Vielfachen von 512 Bit.
3. Der Ausgabepuffer (vier 32 Bit Register) wird mit definierten Werten initialisiert.
4. Die Nachricht wird dann in vier Runden verschlüsselt. Jede Runde beinhaltet 15 Operationen, die auf einer nichtlinearen Funktion, modularer Addition und Linksrotation basieren.

Die Berechnung des MD5-Hashwertes ist aufwendig und erfordert in Software viele Taktzyklen. Hardwareimplementierungen sind entweder sehr hardwareintensiv oder benötigen ebenfalls viel Rechenzeit. Beide Architekturvarianten sind in [DHV01] dargestellt. Die gleiche Aussage kann zu SHA1 [Nat02] als Hashfunktion gemacht werden, der eine ähnliche Struktur wie MD5 hat. Die Hardwareimplementierung von SHA1 in [ZN03] benötigt beispielsweise 80 Taktzyklen zur Berechnung und mehr als 1600 Logikelemente auf einem Altera FPGA.

3.2.3.2 Architekturen zur Paketklassifizierung auf Basis von Hashfunktionen

Es existieren viele Architekturen, die auf Hashfunktionen basieren, um leistungsfähige Paketklassifizierungsalgorithmen zu implementieren. Im Folgenden sollen einige ausgewählte Beispiele vorgestellt werden.

Hashing von drei Präfixlängen

Xiaojun Nie et. al. stellen in [NWC+05] einen IP-Lookup Algorithmus vor, der auf einer dynamischen Hashfunktion basiert. Der Algorithmus arbeitet dreistufig. Die erste Stufe nutzt eine Indextabelle, die aus 256 Einträgen besteht. In dieser enthält jeder Eintrag Informationen, mit wie vielen Bits des Schlüssels gehasht werden soll. Die Auswahl des Eintrags erfolgt über die ersten 8 Bit des Schlüssels. Zur Entscheidungsfindung enthält der Eintrag ein Head-Feld und ein Subindexfeld. Der Head definiert, welche Bits des Schlüssels zum Hashen herangezogen werden sollen. Er kann die Werte 0 bis 3 annehmen:

- 0 ≙ benutze Default Hop (Es existiert kein Eintrag.)

- 1 ≙ 8 Bit
- 2 ≙ 16 Bit
- 3 ≙ 16 oder 24 Bit

Ist der Wert 3, muss das Subindexfeld geprüft werde. Dieses enthält 256 Bit. Der Index für dieses Feld ergibt sich aus dem zweiten Byte des Schlüssels. Existiert an entsprechender Stelle eine '1', sind 24 Bit des Schlüssels zu hashen, anderenfalls nur 16.

Im zweiten Schritt nutzt der Algorithmus den Hashwert des Schlüssels als Index einer Hashtabelle. Jeder Eintrag dieser Tabelle enthält vier Felder:

- Identifier: Unterscheidet die Schlüssel, die an dieselbe Adresse gehasht wurden. Er muss demzufolge 24 Bit breit sein.
- Small Search Group (SSG) Size: Größe der Gruppe, auf die ein Zeiger zeigt.
- SSG Pointer: Zeiger auf eine Gruppe, in der die Regeln zu einem Schlüssel abgespeichert sind.
- Collision Pointer: Zeiger auf eine andere Adresse der Hashtabelle und dient der Kollisionsauflösung.

Die Hashtabelle wird solange mittels Kollisionsauflösung durchsucht, bis der Eintrag für den aktuellen Schlüssel gefunden wurde.

Im dritten Schritt wird eine SSG linear durchsucht. Sie enthält maximal als 16 Einträge. In ihr sind passende Präfixe in absteigender Reihenfolge abgelegt. Jeder Eintrag enthält unter anderem den Suffix des Präfixes und die zugehörige Regel. Abbildung 37 stellt den Aufbau der drei Tabellen dar.

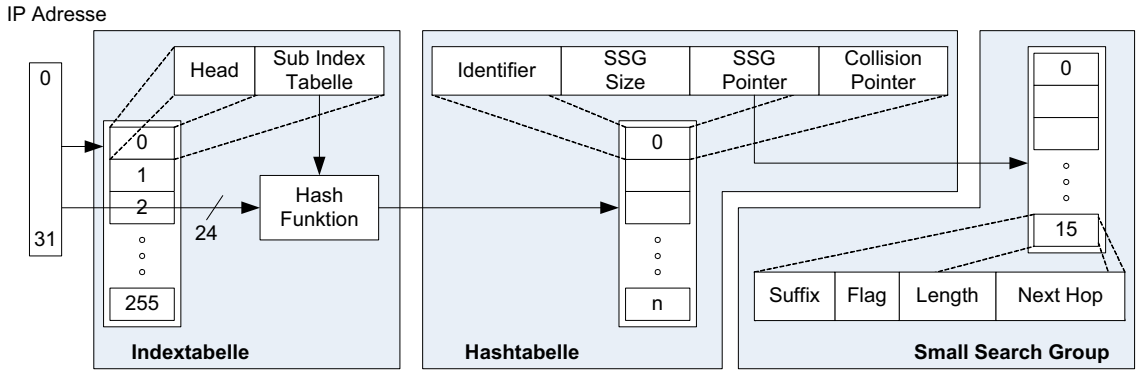


Abbildung 37 - Aufbau der Architektur von Nie.

Eigene Untersuchungen haben ergeben, dass eine Präfixerweiterung durchgeführt werden muss, um sicherzustellen, dass die SSG stets maximal 16 Einträge enthält. Bei 32.768 zufällig gewählten Präfixen ist eine Controlled Prefix Expansion notwendig, die die 32.768 Einträge auf bis zu 950.000 Einträge erweitert. Bei der Verwendung von realen Daten aus einer BGP-Routingtabelle [Rou07] ist für 32.768 Präfixe eine Erweiterung auf 55.329 Präfixe notwendig.

Über die verwendete Hashfunktion selbst und deren Architektur werden keine Aussagen getroffen. Es wird allerdings deutlich, dass eine gute Hashfunktion die Anzahl der Kollisionen in der Hashtabelle vermindern kann. Die meisten Speicherzugriffe erfordert jedoch das lineare Durchsuchen der SSGs.

Mehrfach paralleles Hashing

Waldvogel et. al. stellen in [WVT+97] einen Hash-basierten Algorithmus zur Lösung des LMP-Problems vor. Bei diesem Algorithmus wird Hashing verwendet, um das LMP einer IP-Adresse zu finden. Dazu werden für alle existierenden Präfixlängen eigene Hashtabellen erstellt. Für ein bestimmtes Präfix ist dann ein Hashwert zu bilden. An dieser Adresse beginnt die Suche in der entsprechenden Tabelle. Ist die Suche erfolglos, muss ein neuer Hashwert für das nächst kleinere Präfix gewählt werden. Damit ergibt sich eine lineare Suche in den Hashtabellen.

Waldvogel schlägt nun vor, diese Suche in den Tabellen binär durchzuführen. Dabei wird zuerst die Hashtabelle für die mittlere Präfixlänge durchsucht. Für den Fall, dass es in der unteren Hälfte des Suchbaums (mit den längeren Präfixen) Präfixe gibt, die ebenfalls zutreffen könnten, enthält der entsprechende Tabelleneintrag einen Marker, der darauf verweist. Es kann nun vorkommen, dass in den unteren Teilbaum abgestiegen wird, obwohl dort kein passendes Präfix gefunden wird. Dann ist es notwendig, wieder den oberen Teilbaum zu durchsuchen. Damit würde die binäre Suche im schlimmsten Fall wieder zu einer linearen Suche degenerieren. Um das zu verhindern, enthält jeder Marker, der in den unteren Teilbaum verweist, einen weiteren Eintrag, der das längste passende Präfix des oberen Teilbaums bereits enthält. Dann muss bei erfolgloser Suche nicht zurückgekehrt werden. Der LMP ist in diesem Falle der Markereintrag. Es ist somit möglich, die Suche auf eine logarithmische Komplexität zu beschränken. Weitere Optimierungen sind möglich, indem beispielsweise der Binärbaum ausbalanciert wird. Unterschiedliche Präfixlängen sind unterschiedlich häufig anzutreffen. Waldvogel schlägt deshalb vor, die Präfixlänge 19 als Wurzel zu verwenden. Details zu dieser und zu weiteren Optimierungen des Baumes sind [WVT+97] zu entnehmen.

Dennoch kommt bei jeder Suche innerhalb einer Hashtabelle die Qualität der Hashfunktion zum Tragen. Die Kollisionsauflösung innerhalb jeder Hashtabelle ist nach wie vor jedes Mal durchzuführen. Deren Kosten hängen von der Qualität der Hashfunktion ab. In [WVT+01] schlagen die Autoren unterschiedliche Hashfunktionen und Methoden zur Verringerung von Kollisionen vor. Als Hashfunktionen werden CRC32 und eine auf Multiplikationen basierende Funktion mit $h(key) = f(key \cdot scramble) \cdot bucket_count$ näher betrachtet. Diese beiden Hashfunktionen sind mit unterschiedlichen Tabellenfüllständen dargestellt. Des Weiteren wird die Präfixerweiterung nach Srinvasan und Vargehese ([SV98]) vorgeschlagen, um die Zahl der Kollisionen zu verringern. Dadurch wird es den Autoren zufolge möglich, eine befriedigende Hashperformance zu erreichen.

Mehrfaches Hashing mit Tabellenauswahl

Beim D-Left Algorithmus, vorgestellt von Broder und Mitzenmacher in [BM01], verwenden die Autoren mehrere Hashfunktionen, um einen IP-Lookup durchzuführen. Bei diesem Algorithmus wird ein Speicher von n Behältern[14] in d Gruppen, die jeweils n/d Behälter beinhalten, aufgeteilt. Die Gruppen sind von links nach rechts durchnummeriert. Ein Schlüssel wird nun mit d unterschiedlichen Hashfunktionen gehasht, deren Ergebnisse im Intervall *[1,n/d]* liegen. Dann wird der Schlüssel in den Behälter der Gruppe einsortiert, bei der der Behälter am wenigsten gefüllt ist. Bei Gleichheit der Füllstände wird stets der Behälter der Gruppe, die am weitesten links liegt, gewählt. Die asymmetrische Auswahl der Behälter ist günstiger bei der Vermeidung von Gleichheiten von Füllständen als eine gleichmäßige Verteilung. Insgesamt kann so eine bessere Balance erreicht werden. Um einen Eintrag zu suchen, sind *n/d* Speicherzugriffe notwendig, einer in jeder Gruppe. Da die Gruppen voneinander unabhängig sind, kann dieser Zugriff parallelisiert werden. Die einzelnen Behälter müssen dann linear durchsucht werden. Das kann beschleunigt werden, wenn beispielsweise jeder Behälter aus vier Einträgen besteht und diese alle gleichzeitig gelesen werden können (z.B. Eine Cacheline entspricht einem Behälter). Simulationen des Algorithmus ergeben sehr gute Ergebnisse. Bei Nutzung von drei Hashfunktionen und 30.000 Schlüsseln, die in 6.000 Behälter gehasht werden, ergibt sich eine durchschnittliche Füllung der Behälter von 5. Dennoch ist der maximale Füllstand nicht größer als 6. Damit zeigt sich, dass mit dem D-Left Algorithmus sehr speichereffizient gehasht werden kann. Als Hashfunktionen schlagen die Autoren einen Multiplizierer ohne Carry-Bits und einen einfachen CRC vor. Jede Hashfunktion kann dann auf einem anderen zufälligen Multiplizierer und einem andern nicht reduzierbaren Polynom basieren. Für Tests mit realen Daten wurde eine Routingtabelle mit 38.816 Einträgen verwendet. Die Präfixlängen wurden auf auf 16, 24 oder 32 Bit erweitert. Die Tabelle mit 24 Bit breiten Einträgen ist dann 198.734 Einträge groß. Die anderen beiden wurden ob ihrer geringen Größe ignoriert. Mit zwei Hashfunktionen und insgesamt 65.000 Behältern erreichten die Autoren eine maximale Füllung von 5 und bei 50.000 Behältern 6. Wenn Multiplizierer genutzt wurden, konnte bei 1000 Testläufen und 65536 Behältern 835 mal 5 und 165 mal 6 als Obergrenze ermittelt werden. Die CRC-Funktionen erwiesen sich als bessere Hashfunktionen.

Hashing zur Addressierung eines Content Addressable Random Access Memory

In [CMX+07] beschreiben die Autoren eine Hardwarelösung, die das Hashing mit einem S- oder DRAM kombiniert, um für Suchaufgaben, z.B. beim LMP-Problem, ein leistungsfähiges System bereitzustellen. Der Content Addressable Random Access Memory (CA-RAM) wird

[14] Behälter ist in diesem Zusammenhang eine Menge von Speicherstellen, die die gleiche Adresse aufweisen und beim Hashing die Kollisionsauflösung durch Verkettung abbilden.

dabei durch einen Hashwert adressiert, der aus Teilen der DST-IP eines Paketes besteht. Eine Kollisionsauflösung erfolgt durch die Speicherung aller Einträge mit dem gleichen Hashwert an der entsprechenden Adresse. Die gesamte Speicherzelle wird ausgelesen und die Schlüssel aller Einträge werden durch parallele Matchingprozessoren mit dem gesuchten Schlüssel verglichen. Übersteigt die Anzahl der auftretenden Kollisionen die Menge der speicherbaren Werte, werden die entsprechenden Einträge in spezielle Speicherzellen eingetragen. Diese müssen dann bei Bedarf linear durchsucht werden.

Die dabei eingesetzte Hashfunktion entspricht dem Greedy Algorithmus mit Teilen der Adresse aus [ZNB03].

Hashtabellensuche mit einem erweiterten Bloom Filter

Song et. al. beschreiben in [SDT+05] einen Algorithmus, der Bloom Filter [Blo70] verwendet, um Klassifizierungsaufgaben zu erfüllen. Ein Bloom Filter ist eine Hash-basierte Datenstruktur, um Daten effizient zu speichern. Dabei werden für einen Schlüssel *k* verschiedene Hashfunktionen berechnet und an den *k* Indizes in einer Bitmap die Felder markiert. Das ist für alle Schlüssel einer Menge zu tun. Bei einer Suche kann nun für einen Schlüssel überprüft werden, ob an den *k* Stellen der Bitmap Einsen stehen. Ist das der Fall, so ist der entsprechende Schlüssel sehr wahrscheinlich existent. Bloom Filter können aber zu falsch positiven Ergebnissen führen. Song verwendet einen erweiterten Bloomfilter – einen Counting Bloomfilter [FCA+00]. Statt einer Bitmap findet ein Array von Zählern Verwendung. Für jeden Schlüssel werden alle korrespondierenden *k* Zähler inkrementiert und das Datum an den *k* Adressen gespeichert. Kollisionsauflösungen erfolgen als verkettete Listen. Bei der Suche wird nur die Liste mit dem kleinsten Zähler durchsucht. Dadurch muss nur die kürzeste Liste durchsucht werden. Die anderen *k-1* Einträge in den anderen Listen können wieder gelöscht werden. Die Zählerstände bleiben jedoch unverändert. Die Suchperformance wurde analysiert und für 10.000 Schlüssel simuliert. Die Schlüssel wurden allerdings in einen sehr großen Speicher mit 128k Adressen gehasht. Als Ergebnis ergab sich, dass nur sehr wenige Einträge zwei oder mehr Kollisionen erzeugten. Das Ergebnis ist bei einem Belegungsfaktor der Hashtabelle α von weniger als 8% allerdings nicht überraschend.

Voll paralleles Hashing

Lim et. al. stellen in [LHJ03] eine Hardwarearchitektur zur Paketklassifizierung vor, bei der keine Präfixerweiterungen notwendig sind. Stattdessen wird eine voll parallele Suche auf allen Präfixen der Länge zwischen $l = 8$ und $l = 32$ Bit durchgeführt. Es existieren damit insgesamt 25 Sucheinheiten, für jede Präfixlänge eine, auf die parallel zugegriffen wird. Jede Einheit enthält zwei Tabellen. Die Haupttabelle enthält einen korrespondierenden Speichereintrag an der Adresse $h_l(x)$, wobei der Schlüssel *x* immer *l* Bit breit ist. Sie enthält nur dann einen Eintrag, wenn es keine Kollision gibt. Kommt es zu einer Kollision, verweist ein Zeiger in der

Haupttabelle auf eine Nebentabelle, in der mittels einer binären Suche die Hashkollisionen aufgelöst werden können. Auf diese Weise wird für jede Präfixlänge eine passende Speicheradresse gefunden, sofern sie existiert. Ein Prioritätsencoder leitet dann das Ergebnis des LMP weiter. Auf dessen Basis wird in einem Speicher der Next Hop ausgelesen. Als Hashfunktion verwenden die Autoren ein XOR-Hashing. Der Algorithmus liefert mit 1,93 Speicherzugriffen im Mittel und maximal 5 Speicherzugriffen gute Leistungswerte für einen Datensatz mit 37.000 Einträgen. Auch wird mit 189 KB nicht viel Speicher benötigt. Allerdings ist eine Umsetzung von 25 Speichern, auf die parallel zugegriffen wird, nicht unproblematisch.

Zusammenfassung

Alle vorgestellten Architekturen nutzen Hashfunktionen für die Paketklassifizierung und für jede einzelne Architektur gilt: Je besser die verwendete Hashfunktion ist, desto geringer ist der zeitliche Aufwand, um eine auftretende Kollision zu vermeiden. Wie Tabelle 4 zu entnehmen ist, profitieren ausnahmslos alle Architekturen von besserem Hashing, da sich dadurch die Anzahl der Speicherzugriffe für die Kollisionsauflösung verringert.

Tabelle 4 - Paketklassifizierungsalgorithmen und die verwendeten Hashfunktionen.

Architektur	Verwendete Hashfunktion	Speicheroperationen	Bewertung Suchperformance
Hashing von drei Präfixen	nicht angegeben	Lineare **Kollisions-auflösung** + lineare Suche in der SSM	-
Mehrfach paralleles Hashing	CRC / Multiplikation ohne Carry	Binäre Suche in Hashtabellen + **Kollisions-auflösung** in jeder Tabelle	o
Mehrfaches Hashing mit Tabellenauswahl	CRC / Multiplikation ohne Carry	Parallele Suche in Hash-tabellen + **Kollisionsauflösung** in jeder Tabelle	+
CA-RAM	Greedy	Einzelner Speicherzugriff + lineare **Kollisionsauflösung** für Sonderfälle	+
Hashing mit erweiterten Bloom Filtern	H_3	**Kollisionsauflösung** in einer einzelnen Hashtabelle	++
Voll paralleles Hashing	XOR-Methode	Binäre **Kollisionsauflösung**	++

Für die Architekturen auf Basis von mehreren Hashfunktionen kann man zusammenfassend sagen, je besser die Qualität des Hashings der einzelnen Funktion ist, desto weniger unterschiedliche Hashfunktionen sind notwendig, um die gewünschte Leistungsfähigkeit zu erreichen. Damit verringert sich auch die Komplexität einiger Architekturen.

3.2.4 Caching zur Paketklassifizierung

Zur Beschleunigung von Lesevorgängen auf Speichern, wie sie bei der Paketklassifizierung von zentraler Bedeutung sind, erscheinen Caches als Lösung des Paketklassifizierungsproblems geeignet. Caches sind schnelle Zwischenspeicher. In Rechnersystemen dienen sie beispielsweise als Pufferelement zwischen Hauptspeicher und CPU. Dort sind sie notwendig, da die Zugriffslatenz auf interne Register moderner CPUs intern um ein Vielfaches kleiner ist als die Zugriffslatenz auf externe Speicher. Das fortgesetzte Warten der CPU auf Daten aus dem Hauptspeicher würde die Rechenleistung der CPU stark einschränken. Liegen benötigte Daten im Cache, wird der zeitintensive Hauptspeicherzugriff vermieden. Moderne CPUs besitzen eine ganze Cachehierarchie mit bis zu drei Ebenen [Int06a]. Dabei gilt, je enger der Cache an die CPU angebunden ist, desto schneller, aber auch kleiner, ist er. Caches werden nicht wie herkömmlicher Speicher adressiert. Einträge werden über Teile der Adresse referenziert. Dieser Teil adressiert eine Cacheline, die aus mehreren Einträgen besteht. Die Abbildung von Teilen der Adresse auf die Speicheradresse im Cache könnte als einfache Hashfunktion betrachtet werden, denn es findet eine Abbildung des großen Hauptspeichers auf den kleineren Cache statt. Der Rest der Adresse ist der *Tag*. Es gibt verschiedene Möglichkeiten der Abbildung. Sie sind unterschiedlich gut geeignet, um auftretende Kollisionen zu vermeiden.

Direkte Abbildung: Eine Adresse verweist genau auf eine Cachezeile im Cache. Haben zwei Einträge die gleiche Adresse, muss eine Kollisionsauflösung erfolgen, d.h. nur ein Eintrag kann im Cache gespeichert werden.

Satzassoziativität: Ein Tag verweist auf *n* Cachezeilen mit Einträgen, die gleichzeitig überprüft werden können. Dementsprechend sind *n* Einträge mit dem gleichen Tag speicherbar. Eine Ersetzung hat erst zu erfolgen, wenn mehr als *n* Einträge um den Speicherplatz konkurrieren.

Vollassoziativität: Vollassoziative Caches haben nur Tags. Vollassoziative Caches sind mit CAMs vergleichbar, da alle Einträge gleichzeitig überprüft werden können. Kollisionen werden bis zur Kapazitätsgrenze des Caches vermieden.

Lokalitätsproblem

Ihre große Leistungsfähigkeit in CPUs beziehen Caches aus den Eigenschaften der ausgeführten Programme. Die verwendeten Daten und der ausgeführte Code weisen sehr oft eine räumliche bzw. temporäre Lokalität auf.

Räumliche Lokalität beschreibt den Umstand, dass, nachdem ein bestimmtes Datum verarbeitet wurde, das nächste zu verarbeitende Datum an der Folgeadresse im Hauptspeicher

zu finden ist. Ein Cache kann nun spekulativ die entsprechenden Daten laden und der CPU ohne Verzögerung zur Verfügung stellen. Das lässt sich am Beispiel der Verarbeitung von Videodaten verdeutlichen. Beim Abspielen eines Videos lässt sich sehr gut voraussehen, welche Daten als nächste benötigt werden. Die Annahme wird nur falsch sein, wenn eine unvorhergesehene Aktion, wie z.B. ein Rückspulvorgang ausgeführt wird.

Temporäre Lokalität liegt vor, wenn innerhalb eines bestimmten Zeitraums immer mit den gleichen Daten gearbeitet wird oder immer wieder dasselbe Codefragment ausgeführt wird. Dies ist z.B. bei wiederholten Schleifendurchläufen der Fall.

Bei der Paketklassifizierung in Kommunikationssystemen käme als vorliegendeLokalität eine temporäre Lokalität der Daten in Betracht. Eine Studie aus dem Jahr 1996 [Par96] erreicht mit einem Cache, der 6000 Einträge fasst, eine Trefferrate von 95%. Das heißt, das 19 von 20 Suchanfragen durch einen Cachezugriffe beantwortet werden. Diese Ergebnisse wurden allerdings auf Basis von Daten eines verhältnismäßig schmalbandigen 37 MBit/s Datenstroms erzielt, der nur wenige Teilnehmer auf sich vereinte. In [NML+97] wird angezweifelt, ob ein solche hoher Grad von Lokalität in hochbitratigen Datenströmen(1 GBit/s und mehr), die auch eine größere Zahl von Teilnehmern vereinen, heutzutage als realistisch zu betrachten ist. Vor allem in Kernnetzen mit Multi-GBit/s Datenströmen und tausenden oder hunderttausenden von Nutzern bzw. Flows ist zu bezweifeln, ob eine signifikante temporäre Lokalität vorliegen könnte. Je weiter die Paketklassifizierung aus dem Kern- in den Zugangsbereich verlagert wird, kann jedoch bei sinkenden Teilnehmerzahlen und Bandbreiten von zunehmender Lokalität ausgegangen werden. Die in [SMG03] und [RDS04] verwendeten Datensätze der Universität von Alberta und des National Laboratory for Applied Network Research weisen eine gewisse temporäre Lokalität auf. Auch die Daten aus [Fel88] basieren auf Daten eines Gateways, das nicht in einem Kernnetz liegt. Es werden Daten des MIT ARPANET Gateways verwendet, das den Übergang vom MIT-Campus in das Internet gewährleistet.

In der Literatur existieren verschiedene Ansätze zur Unterstützung von Paketklassifizierern durch Cachearchitekturen. [CP99] stellt einen Algorithmus zum Routing Table Lookup vor, der die Cachearchitektur des Alpha-Prozessors mit zwei Cachehierarchien effektiv ausnutzt. Die Autoren erweitern diesen Ansatz in [CP00] auf das Design eines Caches für Netzwerkprozessoren. Auch in [ONI+06] findet Caching Verwendung, um die Paketverarbeitung in Netzwerkprozessoren zu beschleunigen.

Zusammenfassend kann davon ausgegangen werden, dass im LAN unter Umständen eine gewisse temporäre Lokalität der Daten gegeben ist, da dort nur eine begrenzte Anzahl von Teilnehmern miteinander kommuniziert. Da im WAN-Bereich die Daten einer sehr großen Anzahl verschiedener Teilnehmer unabhängig voneinander parallel bearbeitet werden müssen, ist eine Datenlokalität nur sehr eingeschränkt bzw. gar nicht gegeben. Daraus folgt, dass herkömmliche Cachearchitekturen zur Paketklassifizierung allenfalls unterstützend einzusetzen sind. Eine rein cachebasierte Lösung erscheint nicht sinnvoll.

4 Paketverarbeitende Systeme im Teilnehmerzugangsnetzwerk

Im Rahmen der Forschungen für neue Funktionen im Bereich des Teilnehmerzugangsnetzwerks wurden vom Autor verschiedene Systeme entwickelt, welche im folgenden Kapitel im Einzelnen näher vorgestellt und erläutert werden sollen.

4.1 Funktionale und architektonische Anforderungen

In modernen TZNs steigen die Leistungsanforderungen an die zuständigen Systeme wie Digital Subscriber Line Access Multiplexer (DSLAMs), Gigabit-capable Passive Optical Networks (GPONs) oder Broadband Remote Access Server (BRAS) immer weiter an. Bezüglich der Bandbreite und des funktionalen Umfangs sind die Anforderungen von ISPs in den letzten Jahren immer größer geworden [Age05], [DSL03a]. Während früher nur ADSL über ATM unterstützt werden musste, sind im DSLAM-Bereich reines Ethernet und auch VDSL über Ethernet zu unterstützen [Par05]. ATM verschwindet mehr und mehr aus dem Kernnetzwerk und dem TZN [SGS05]. ISPs fordern, dass mehr Funktionalität bereits im Teilnehmerzugangsbereich angeboten wird. Das geschieht aus zwei Gründen. Zum einen ist es das Ziel der Provider, ihre Kernnetze vor zu großer Rechen- und Verkehrslast zu schützen. Zum anderen ist es notwendig, dass gewisse Dienste bereits im Teilnehmerzugangsbereich erbracht werden, denn diese bedürfen Informationen, die nur dort zur Verfügung stehen. Solche Informationen sind z.B. Portinformationen, die für QoS-Funktionen benötigt werden [DSL03b]. Die Information, an welchem Zugangsport ein Datenpaket in das Teilnehmerzugangssystem eintritt, ist nur in diesem System und nicht in späteren Netzwerkelementen verfügbar.

Viele der Funktionen im TZN, wie z.B. die Protokollerkennung, die Protokollklassifizierung, das Traffic Shaping, die Fehlererkennung und -korrektur und die Datenmanipulation sind paketorientierte Operationen. Sie können mit einem paketverarbeitenden System, einem Paketprozessor, ausgeführt werden. Die Aufgaben eines solchen sind die Manipulation und der Transport von Datenframes bzw. Datenpaketen. Wie bereits in Abschnitt 3.1 erläutert wurde, können solche Funktionalitäten mit einem Netzwerkprozessor implementiert werden. Alternativ ist eine Implementierung als ASIC, einer dedizierten Hardwarelösung möglich. Eine dritte Möglichkeit besteht in der Nutzung einer flexiblen Hardwareplattform, wie sie ein FPGA darstellt.

Ein Netzwerkprozessor kann zur Implementierung praktisch jeder Funktionalität im Bereich der Paketverarbeitung verwendet werden. Er hat den Vorteil größter Flexibilität, da er weitgehend frei, per Software, programmierbar ist. Dennoch disqualifizieren ihn seine größten Nachteile für den Einsatz in Teilnehmerzugangssystemen. Netzwerkprozessoren sind aufgrund geringer Stückzahlen verhältnismäßig teuer und weisen zu große Beschränkungen auf, was den

möglichen Datendurchsatz betrifft [Int03]. Auch wenn ein Netzwerkprozessor potent genug wäre, um in aktuellen Systemen Verwendung zu finden, ist es nicht möglich, seine Leistungsfähigkeit an steigende Anforderungen anzupassen. Wird mehr Bandbreite benötigt, sind die Kapazitäten eines Netzwerkprozessors erschöpft. Eine Anpassung ist nur durch den Einsatz eines leistungsfähigeren Prozessors möglich.

Die Alternativlösung auf Basis eines ASICs führt ihre Funktionen als reine Hardwarelösung aus. Dadurch hat ein ASIC hervorragende Leistungsdaten. Der Hauptnachteil einer solchen Lösung ist jedoch die mangelnde Flexibilität. Falls eine ASIC-Lösung eine spezielle Aufgabe nicht zufriedenstellend lösen kann, ist es unmöglich, einen Kompromiss zwischen Funktionalität und Leistung einzugehen. Die spezielle Funktionalität steht schlicht nicht zur Verfügung. Ein Coprozessor wäre notwendig, um die gewünschte Funktionalität zur Verfügung zu stellen.

Als Kompromiss zwischen Performance und Flexibilität stellen FPGAs eine gute Alternative dar. Die Autoren von [KR06] zeigen, dass die Performancelücke zwischen ASICs und FPGAs im Mittel kleiner ist als Faktor 5. Ein angepasstes FPGA-Design ist außerdem in der Lage, eine maßgeschneiderte Lösung für ein spezielles Problem zu bieten. Vielleicht noch wichtiger ist der Umstand, dass eine Anpassung an veränderte Anforderungen, sowohl Funktionalität als auch Leistungsfähigkeit betreffend, in verhältnismäßig kurzer Zeit möglich ist. Oft ist es sogar möglich, die FPGA-Plattform beizubehalten, um eine neue Version zu implementieren. Veränderungen können sogar „im Feld“, an bereits ausgelieferten und laufenden Systemen, vorgenommen werden.

Speziell für die Anforderungen der Paketverarbeitung im Teilnehmerzugangsbereich wurde eine Architektur eines funktionalen Moduls entwickelt, das große Datenmengen mit kurzen Latenzen und geringen Paketverlusten verarbeiten kann [WKT07].

4.2 Architektur funktionaler Module

Um die für die in aktuellen Teilnehmerzugangssystemen notwendige Datenrate von 4 Gbit/s zu ermöglichen, kommen Softwarelösungen auf Basis von Netzwerkprozessoren nicht in Frage. Einen guten Kompromiss zwischen flexibler Softwarelösung und leistungsstarker Hardwarelösung stellt eine FPGA-Lösung dar, weswegen diese als Basis für die hier vorgestellten funktionalen Module dient. Wie bereits in Kapitel 2.1.2 erläutert, weisen paketverarbeitende Systeme einen Kontrollpfad (Slow-Path) und einen Datenpfad (Fast-Path) auf. Das ist sinnvoll, um Operationen, die nur geringe Leistungsanforderungen haben, auf ressourcensparenden Softwaresystemen auszuführen. Für Operationen im Fast-Path sind Hardwarelösungen vorzusehen.

Um eine flexible und anpassungsfähige Lösung zu entwickeln, wird ein Ansatz vorgeschlagen, mit dem funktionale Module (FMs) erzeugt und zu komplexeren Systemen kombiniert werden können. In diesen FMs sind dann unterschiedliche Funktionalitäten für die

Paketverarbeitung implementiert. Ein funktionales Modul (FM) hat immer die gleiche, wohl definierte Schnittstelle zur Kommunikation mit anderen Komponenten. Unter anderem dadurch wird eine Kombination von FMs ermöglicht bzw. vereinfacht. Die Schnittstelle besteht aus einem Ein- und Ausgang für Daten des Fast-Path und einem Kontrollinterface. Das Kontrollinterface ermöglicht die Steuerung des FMs, das Sammeln von Statusinformationen und einen CPU-Zugriff, um die Operationen des Slow-Path auf einer externen CPU durchführen zu können. Durch die genau definierten Schnittstellen ist der interne Aufbau einzelner FMs für eine funktionsfähige Zusammenschaltung nicht relevant. Um jedoch bei einer seriellen Zusammenschaltung die Entstehung eines Flaschenhalses zu vermeiden, müssen alle zusammengeschalteten Module die gleiche Mindestdatenrate aufweisen, denn das schmalbandigste FM limitiert den Durchsatz des Gesamtsystems, das dann einer Pipeline gleicht.

Die entwickelte Architektur ermöglicht hohe Datendurchsätze, ohne große Latenzen in den Datenpfad einzufügen. Bei dieser Architektur (Abbildung 38) besteht ein FM im Kern aus drei Hauptelementen: Einem Framebuffer (FB) mit integriertem Datenparser, einem Controller und Arbiter (CA), der mit einem Speicher und einem Interface zum Slow-Path verbunden ist und einer Ausführungseinheit (AE). Die Ausführungseinheit beinhaltet die eigentliche Funktionalität, für die ein bestimmtes FM konzipiert ist. Das System arbeitet wie folgt:

- Der Framebuffer speichert ankommende Datenpakete. Der Parser extrahiert aus jedem Paket einen Bezeichner (Schlüssel) und übermittelt diesen an den CA.
- Der CA koordiniert die Speicherzugriffe, bei denen für jeden Schlüssel eine zugehörige Information gesucht wird.
- Mit der ermittelten Information, die der CA an die AE übergibt, und dem Datenpaket, das vom FB zur AE übertragen wird, kann die AE ihre vorgesehenen Funktionen ausführen und das Datenpaket manipulieren.
- Das manipulierte Datenpaket verlässt dann das FM.

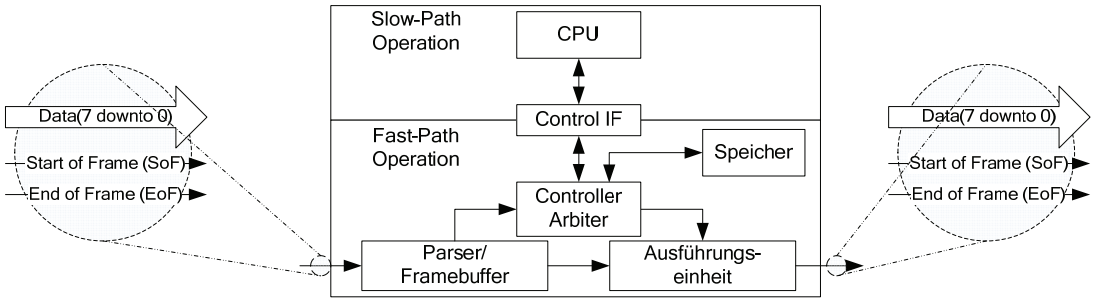


Abbildung 38 - Architektur eines Funktionalen Moduls bestehend aus FB, CA, Speicher und AE.

Ein Beispiel: In einem FM mit Routingfunktionalität könnte eine einzelne Forwarding-Regel wie folgt formuliert sein:

```
if (packet.dest_IP == 130.60.48.8) then route_to_port 5
```

Der Parser extrahiert die DST_IP des gespeicherten Pakets, übergibt sie dem CA, welcher die Suche nach der Regel steuert. Die Regel ist zusammen mit dem Schlüssel im Speicher abgelegt. Die gefundene Regel wird zusammen mit dem Paket an die AE übergeben, die das Datenpaket in eine Warteschlage einfügt, die Port 5 bedient. Im Folgenden sind der Aufbau und die Funktionsweise eines FMs genauer beschrieben.

4.2.1 Framebuffer mit Parser

Auf die durch das FM erzeugte Latenz im Datenstrom hat der Framebuffer den größten und entscheidenden Einfluss. Er hat zwei Aufgaben: Das Puffern der Datenpakete und das Extrahieren des Schlüssels aus jedem Paket. Alle ankommenden Pakete sind zu puffern, indem sie in einem First-In-First-Out Speicher (FIFO), bestehend aus internem Block-RAM (BRAM) des FPGAs, abgelegt werden. Die Größe des FIFOs ist variabel konzipiert und kann je nach Anforderung gewählt werden. Es ist nicht notwendig, ankommende Pakete in einem externen Speicher abzulegen, wie das bei herkömmlichen Netzwerkprozessoren der Fall ist [Int03]. Da die Speicherung der Pakete im Datenpfad und exklusiv für das FM geschieht, sind keine zeitraubenden und unnötigen zusätzlichen Speicheroperationen notwendig, wie es bei Lösungen mit externen Off-chip Speichern der Fall ist. Dadurch wird das wiederholte Lesen und Schreiben auf externe Speicher, was ein bekanntes Softwareproblem ist [HBB+05], vermieden und die Latenz im Datenpfad wird minimiert. Als moderner Vertreter seiner Art bietet der Xilinx XC4VFX20 FPGA insgesamt jedoch nur 1,2 MBit On-chip Speicher. Der einzige Weg, die interne Speicherkapazität zu vergrößern, wäre die Wahl eines größeren FPGAs. Die Grenze liegt derzeit bei etwa 16 MBit (Xilinx XC5VFX200T [Xil07b]). Die Größe der Puffer im FB ist also verhältnismäßig großen Einschränkungen unterworfen.

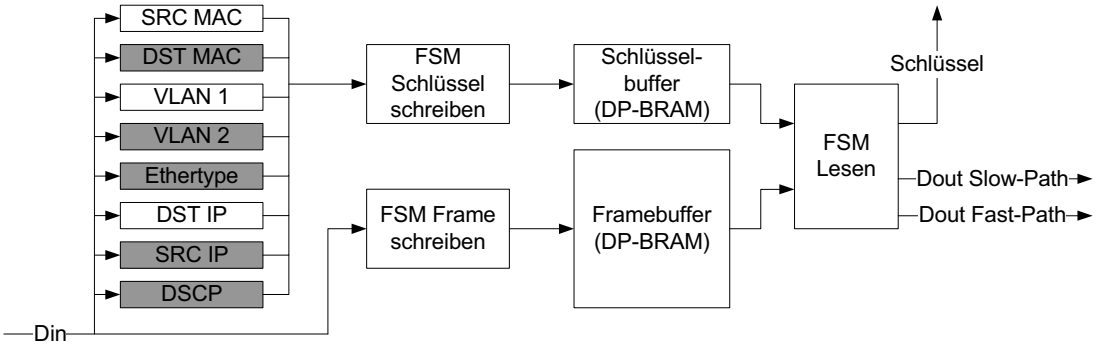


Abbildung 39 - Framebuffer mit Parser mit integrierten Teilmodulen zum Parsen von SRC-MAC, VLAN und DST-IP. Die grau schraffierten Elemente sind nicht für die Synthese konfiguriert.

Wie oben erwähnt wurde, hat der FB neben der Paketspeicherung eine zweite Aufgabe. Er durchsucht jeden Frame byteweise und extrahiert Teile des Frames als Schlüssel. Das sind Teile der Header aus den Schichten 2 und 3 des ISO-OSI Modells. Der Parser kann zur Synthesezeit

konfiguriert werden, um beliebige Kombinationen von Feldern der Protokollheader zu extrahieren. Dazu gehören: SRC-MAC, DST-MAC, VLAN, ein weiteres VLAN-Tag, falls diese gestapelt sind, SRC-IP, DST-IP, DSCP-Feld des IP-Headers. Für jedes Feld wird ein Hardwareblock instanziiert. Dadurch können Ressourcen eingespart werden. Denn es sind nur die Teile des Parsers implementiert, die notwendig sind. Außerdem ist es sehr einfach, Logikblöcke hinzuzufügen, um beispielsweise SRC-Port und DST-Port der Transportschicht zu nutzen. Damit kann die gleiche Parserarchitektur in unterschiedlichen FMs Verwendung finden. Aus allen genutzten Blöcken kombiniert der Parser den Schlüssel, der an den CA übertragen wird. Abhängig vom Ergebnis der Speicherabfrage transferiert der FB das Datenpaket entweder zur AE für Datenpfadoperationen (Fast-Path) oder zum Kontrollinterface, damit Kontrollpfad-operationen ausgeführt werden können (Slow-Path).

4.2.2 Controller und Arbiter

Das CA-Modul steuert die Kommunikation innerhalb eines FMs. Es arbeitet in der Hauptsache als Arbitrierungseinheit für Speicherzugriffe. Wie aus Abbildung 38 hervorgeht, scheint diese Aufgabe sehr einfach zu sein. Wenn gleichzeitige Zugriffe auftreten, muss nur entschieden werden, ob der Framebuffer mit seiner Suchanfrage oder das Kontrollinterface mit einer Anfrage von der CPU als erstes Zugriff auf den Speicher erhalten soll. Durch die Zugriffe von der CPU wird der Inhalt des Speichers konfiguriert und verändert. Diesen Zugriffen ist die höchste Priorität zuzuweisen, um Datenintegrität sicherzustellen.

Um die Bandbreite eines FMs an variierende Leistungsanforderungen anzupassen, die jenseits von 1 GBit/s liegen, ist es möglich, den Datenpfad (FB und AE) zu vervielfachen. Der Datendurchsatz eines FMs liegt, wenn es mit 125 MHz betrieben wird, bei 1 GBit/s. Aktuell liegen die Anforderungen an den Datendurchsatz in Teilnehmerzugangssystemen bei 4 GBit/s. Es sind also vier parallele Datenpfade notwendig. Damit ergibt sich eine Struktur, die der Darstellung in Abbildung 40 entspricht.

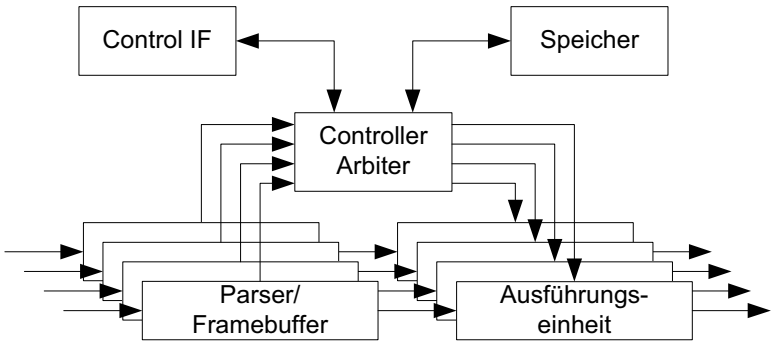


Abbildung 40 - Funktionales Modul mit vier parallelen Datenpfaden.

Die Datenpfade teilen sich jedoch ein Speichermodul. Auf Grund der geringen Verfügbarkeit von Block-RAM, ist es nicht möglich, zusammen mit den parallelisierten Datenpfaden auch den

Speicher zu vervielfachen. Demzufolge müssen sich alle Datenpfade einen angeschlossenen Speicher teilen, weshalb dessen Leistungsfähigkeit die Performance des Gesamtsystems bestimmt und begrenzt. Speicheranfragen aller Datenpfade konkurrieren nun um den Speicherzugriff. Das verkompliziert die Aufgabe des Arbiters. Es macht die Integration eines geeigneten Arbitrierungsalgorithmus notwendig.

Verschiedene Algorithmen sind denkbar. Eine einfache Lösung könnte ein Scheduling nach einem Round Robin Verfahren [Tan95] sein. Beim Round Robin bekommt nacheinander jeder Datenpfad vom Scheduler einen Zeitschlitz (ein Zeitquantum) zugeteilt, innerhalb dessen er Zugriff auf den Speicher erhält. Das Verfahren hat den Vorteil, leicht implementierbar und sehr fair zu sein. Ein entscheidender Nachteil ist jedoch, dass die Zuteilung des Speicherzugriffs unabhängig vom tatsächlichen Bedarf jedes einzelnen Datenpfads ist. Das kann bei unbalancierter Auslastung der Datenpfade zu unnötigen Paketverlusten führen. Eine bessere Möglichkeit stellt in diesem Fall das Earliest Deadline First (EDF) Scheduling dar [Der74, LL73]. Beim EDF bekommt derjenige Datenpfad den Speicherzugriff gewährt, dessen Framebuffer die geringste Restkapazität aufweist, also dessen FIFO am vollsten ist. Der vollste FIFO hat dann die nächstliegende Deadline und wird zuerst bedient. Datenpakete sind zwischen 46 und 1500 Byte lang. Es ist möglich, dass ein FIFO von 4 KByte bereits mit zwei Datenframes zu 75% gefüllt ist. Andererseits besteht die Möglichkeit, dass bei 75% Füllung fast 67 minimal große Pakete gespeichert sind. Der eine FIFO benötigt so 30mal mehr Speicherzugriffe zur Leerung als der andere. Nur den Füllstand des FIFOs für die Entscheidung zur Auswahl des Datenpfades für den Speicherzugriff zu verwenden, erscheint aus diesem Blickwinkel ebenfalls ungenügend. Aus diesem Grund wird für das Scheduling der Least Laxity First (LLF) Algorithmus verwendet [Loc85]. Wie anderen Algorithmen auch, wird LLF primär im Bereich des Prozessschedulings bei Taskwechseln in Betriebssystemen eingesetzt. Im Bereich des CA kann LLF verwendet werden, um dem Datenpfad mit dem kleinsten Schlupf den Zugriff auf den Speicher zu gewähren. Beim Prozessscheduling ist der Schlupf von zwei Parametern abhängig: der einzuhaltenden Deadline und der benötigten Rechenzeit. Der Schlupf berechnet sich aus der Differenz von Deadline und Rechenzeit. Im Falle von gleicher Rechenzeit bestimmt die Deadline, welcher Prozess Vorrang erhält. Im CA leitet sich die Deadline, wie bereits erläutert, aus der Größe des noch freien Teils des FIFOs ab, in dem die Datenpakete gespeichert sind. Die Rechenzeit ist durch die Anzahl der Datenpakete bestimmt, die bereits abgespeichert sind. Sowohl Deadline als auch Rechenzeit werden vom FB an den CA in Form von vier Bit breiten Werten zusammen mit dem aktuellen Schlüssel übergeben (Abbildung 41). Daraus erzeugen die Schlüsselmodule einen 10 Bit breiten Wert, dessen höherwertiger Teil den Schlupf enthält. Der niederwertige Teil des Datenwortes enthält die Deadline. Der Datenpfad mit dem kleinsten Schlupf erhält Zugriff auf den Speicher. Bei gleichem Schlupf entscheidet die kleinere Deadline über den Zugriff. Sind sowohl Schlupf als auch Deadline gleich, teilt der Scheduler den Speicherzugriff zufällig zu, um zu verhindern,

dass bei maximaler Systemauslastung einige Datenpfade anderen vorgezogen werden. Als Zufallszahlengenerator findet ein LFSR Verwendung.

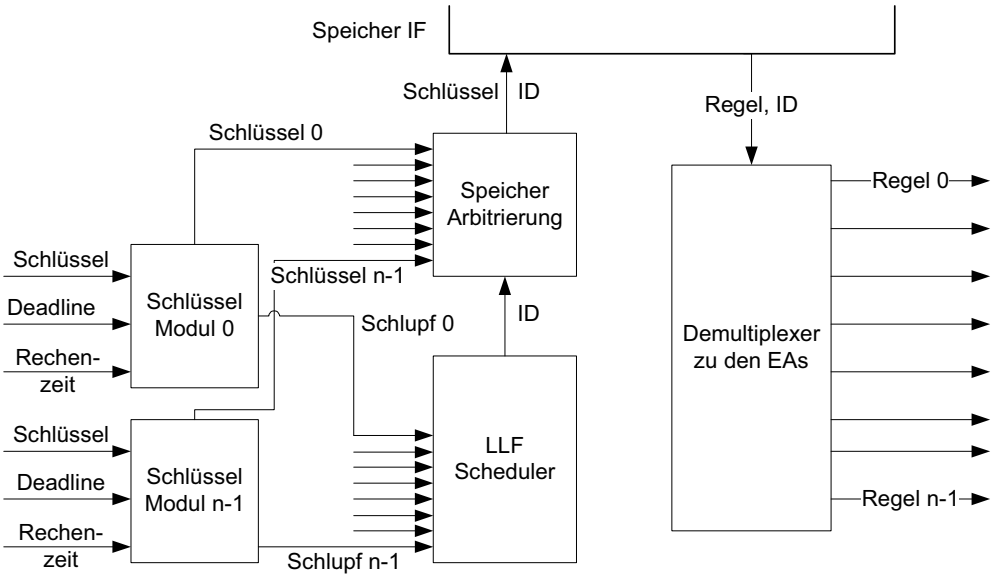


Abbildung 41 - CA Architektur mit LLF Scheduling Modul und Demultiplexer zur Übertragung von Regeln an die parallelen EAs.

Es wurde in Erwägung gezogen, den Enhanced LLF (ELLF) Algorithmus einzusetzen [HGT99, Hil04]. Dieser hat im Bereich des Prozessschedulings den Vorteil gegenüber LLF, dass das sogenannte Thrashing vermieden wird. Unter Thrashing versteht man nicht notwendige Taskwechsel. Diese können auftreten, wenn ein Prozess eine Zuteilung bekommt und sich demzufolge sein Schlupf wieder vergrößert. Ein weiterer Prozess erreicht dadurch im Verlauf der Abarbeitung des ersten einen geringeren Schlupf und bekommt die Prozessorzuteilung. So wechseln sich beide Prozesse ab, bis sie beendet werden. Weitaus effizienter ist eine Abarbeitung nacheinander, da im Betriebssystembereich Task- und Prozesswechsel mit hohen Kosten verbunden sind [Tan95]. Bei der Anwendung der Speicherzuteilung für einzelne Datenpfade bestehen diese Kosten nicht, da das Scheduling durch eine dedizierte Hardware ausgeführt wird. Zuordnungswechsel sind faktisch kostenfrei, weswegen eine Verhinderung des Thrashings keinerlei Vorteile mit sich bringt. Es ist folglich nicht implementiert.

4.2.3 Speichersystem

Bei der Implementierung des Speichersystems muss sorgfältig vorgegangen werden, da dessen Leistungsfähigkeit die Performance des Gesamtsystems limitiert. Der BRAM kann nicht zusammen mit den Datenpfaden parallelisiert werden, weil nicht genügend Ressourcen zur Verfügung stehen. Deshalb müssen sich alle Datenpfade den Speicher teilen. Da bei paketverarbeitenden Systemen für jedes Paket eine Bearbeitungsvorschrift gefunden werden muss, findet auch für jedes Paket eine Suche im Speicher statt. Diese muss deshalb mit

möglichst wenigen Speicherzugriffen auskommen. Bei Verwendung einer einfachen linearen Suche ist mit langen Wartezeiten und vielen Speicherzugriffen zu rechnen, da hierbei eine Suchkomplexität von $O(N)$ vorliegt. Um die Suchgeschwindigkeit zu optimieren, kann ein sortierter Speicher Verwendung finden. Dieser hat den Vorteil, dass ein binärer Suchalgorithmus verwendet werden kann, welcher eine Komplexität von $O(\log(N))$ hat. Bei aktuellen paketverarbeitenden Systemen im Teilnehmerzugangsbereich geht man von nicht mehr als 4k angeschlossenen Teilnehmern aus, deren Datenströme zu unterscheiden sind. Bei einer Verarbeitungsgeschwindigkeit von 125 MHz und einer Datenrate von 1 GBit/s ergibt sich für minimale Datenframes von 64 Byte Länge und einem IFG[15] von 12 Byte sowie der Präambel von 8 Byte eine Zeit von 84 Taktzyklen, die zur Suche nach einem Eintrag im Speicher zur Verfügung steht. Für vier parallele Datenpfade reduziert sich die Zeit auf 21 Takte. Bei einer binären Suche in 4k Einträgen sind maximal 12 Speicherzugriffe notwendig. Die reine Speicherbandbreite reicht für die Anwendung im Teilnehmerzugangsbereich also aus. Die Nutzung einer sortierten Liste hat aber den Nachteil, dass Änderungen (Hinzufügen oder Löschen von Einträgen) mit hohen Kosten verbunden sind, da eine neue Sortierung des Speichers notwendig wird. Die Komplexität von Einfüge- und Löschoperationen liegt bei $O(N)$. Im Mittel sind $\log_2(N) + N/2$ Speicherzugriffe notwendig. Da jedoch die Datenbasen in DSLAMs nicht oft verändert werden müssen, können diese zusätzlichen Kosten in Kauf genommen werden.

BRAMs haben eine Latenz von einem Takt. Ein einfacher Suchalgorithmus benötigt deshalb zwei Takte, um einen Eintrag zu überprüfen (Abbildung 42). Demzufolge sind die Kapazitäten für den oben skizzierten Extremfall nicht ausreichend. Aus diesem Grund wird eine Prädiktion eingeführt. Nachdem eine Adresse angelegt ist, wird im Folgetakt, ohne das Suchergebnis und damit die tatsächliche Folgeadresse zu kennen, die nächste wahrscheinliche Adresse angelegt. Dabei verwendet der Algorithmus die Mitte der oberen Hälfte des noch nicht durchsuchten Speichers als Adresse. Im Mittel ist diese Annahme in 50% der Fälle korrekt. Im anderen Fall wäre eine weitere Suche in der unteren Hälfte korrekt gewesen. Dadurch ergibt sich eine mittlere Latenz von 1,5 Takten pro Suchzugriff. Mit im Mittel 18 Takten ist dadurch die notwendige Leistungsfähigkeit gegeben, um Daten mit einer Datenrate von 4 GBit/s zu verarbeiten.

[15] IFG: Der Interframegap bezeichnet die Pause zwischen der Übertragung zweier Frames bei der Übertragung im Ethernet.

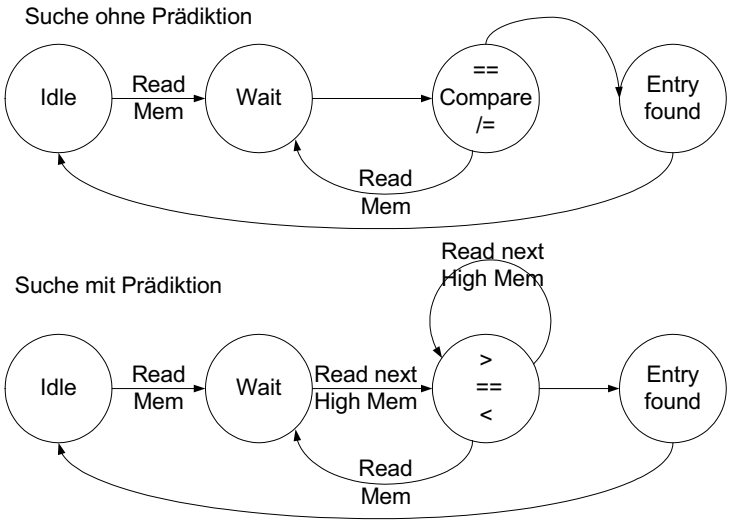


Abbildung 42 - Zustandsmaschine des Suchalgorithmus mit und ohne Prädiktion.

4.2.4 Ausführungseinheit

Die Ausführungseinheit bildet den funktionalen Kern eines FMs. Es können verschiedene Funktionalitäten implementiert werden. Als Teil der Untersuchungen wurden unterschiedliche AEs entwickelt, welche in Funktion und Leistungsfähigkeit in Kapitel 4.2.5 näher beschrieben werden. Die grundsätzlichen Architekturen aller Funktionalitäten gleichen sich. Jedes ankommende Datenpaket wird manipuliert. Das geschieht in Abhängigkeit von der Regel, die bei der Suche mittels des Schlüssels im Speicher gefunden wurde. Eine Ausführungseinheit beinhaltet eine oder mehrere Zustandsmaschinen unterschiedlicher Komplexität. Diese verändern, überprüfen, sortieren oder markieren Datenpakete. Danach verlassen die Pakete die AE, um weiter verarbeitet werden zu können. Die genaue Implementierung der AE hängt vom jeweiligen Anwendungsfall ab.

4.2.5 Zusammenfassung

Die entwickelte Architektur ähnelt in ihrem Aufbau den Architekturen zur Paketverarbeitung aus [PVN+04] und [MOW+07].

Die in [PVN+04] verwendete Programmable Processing Engine (PPE) aus Kapitel 3.1.2.3 besteht aus drei Teilen. Der FEX extrahiert Felder aus den Paketen. Dieses entspricht funktional den Aufgaben des Framebuffers mit Parser. Im Unterschied zum FEX ist der FB nicht im engeren Sinne programmierbar. Der FB wird zur Syntheseezeit konfiguriert, um die gewünschten Teile eines Headers zu extrahieren. In der Benutzung ist er deshalb etwas weniger flexibel als der FEX, jedoch ist die Implementierung dadurch leistungsstark und ressourcenschonend. Die Aufgaben des RISC-Kerns aus der PPE erfüllt beim FM die spezielle Logik in CA und Speichersystem. Die AE schließlich manipuliert die Pakete, gleich der FMO

der PPE. Der entscheidende Unterschied der Architekturen ist, dass beim FM in jeder Stufe speziell angepasste Hardwareblöcke (Zustandsmaschinen) zum Einsatz kommen, während bei der PPE überall programmierbare Module genutzt werden. Die FM-Architektur ist dadurch sehr leistungsstark und hardwareeffizient. Programmierbarkeit ist zur Synthesezeit gegeben. Hier können alle Komponenten an spezielle Aufgaben angepasst werden.

Die Programmable Stream Processing Engine [MOW+07] aus Kapitel 3.1.3.1 weist als Pendant zur AE eine Pipeline mit steuerbaren Funktionseinheiten auf, um verschiedene Funktionalitäten mit ein und derselben Hardware zu unterstützen. Nachteilig bei dieser Architektur ist, dass mit diesen Einheiten evtl. nicht jede mögliche bzw. nötige Funktion umsetzbar ist. Außerdem ist vorhandene Funktionalität für bestimmte Aufgaben nicht notwendig. Drittens ist die Größe der Funktionseinheit mit über 2000 Slices beachtlich. Spezielle AEs können sehr schlank implementiert werden. Das wird im folgenden Abschnitt verdeutlicht, in dem die vom Autor entwickelten Architekturen für den Teilnehmerzugangsbereich, alle auf Basis der FM-Architektur, vorgestellt werden. Die Leistungsfähigkeit der FM-Architektur wird ebenfalls anhand der Beispielimplementierungen verdeutlicht.

4.3 Beispiele paketverarbeitender Systeme im TZN

Die im Rahmen dieser Arbeit entwickelten funktionalen Module für den Teilnehmerzugangsbereich verbessern verschiedene Eigenschaften von Teilnehmerzugangssystemen. Dazu gehören Skalierbarkeit, Sicherheit, Zuverlässigkeit, Verfügbarkeit und Erweiterung der Funktionalität. Auf die Skalierbarkeit und Sicherheit von Teilnehmerzugangssystemen zielen die entwickelten Funktionalen Module zur MAC Address Translation (MAT) ab [KWD+06]. Mit dem Traffic Manager (TM) [KWD+06] soll die Verfügbarkeit von Diensten erhöht werden. Die Funktionserweiterung von Teilnehmerzugangsnetzen wird durch die Integration eines Multi Protocol Label Switching User Network Interface (MPLS-UNI) [WKD+06] ermöglicht. Die Verbindung der Funktionen zu MATMUNI[16], einer integrierten Lösung, wird ebenfalls erläutert [WKT+06]. Durch die Einführung der IP-Calling Line Identification Presentation (IPclip) [KWD+08a] können sogar neue Dienste angeboten werden.

4.3.1 MPLS User Network Interface

Die aktuelle Entwicklung von Teilnehmerzugangsnetzen deutet auf eine in Zukunft größere funktionale Diversifikation von Diensten hin. Im Moment favorisieren Endkunden Ethernet-basiertes DSL, während Geschäftskunden es vorziehen, ihre eigenen LANs mit dem Weitverkehrsnetz eines Netzbetreibers zu verbinden. Das heißt, in Zukunft sind nicht nur einzelne Kunden, sondern ganze Netze an die Linecards eines Teilnehmerzugangsknotens angeschlossen. Neben hohen Datenraten fordern die Kunden QoS, verschiedene Dienste und die

[16] MATMUNI ist ein Kunstwort aus der Verbindung von **MA**T, **TM** und MPLS-**UNI**.

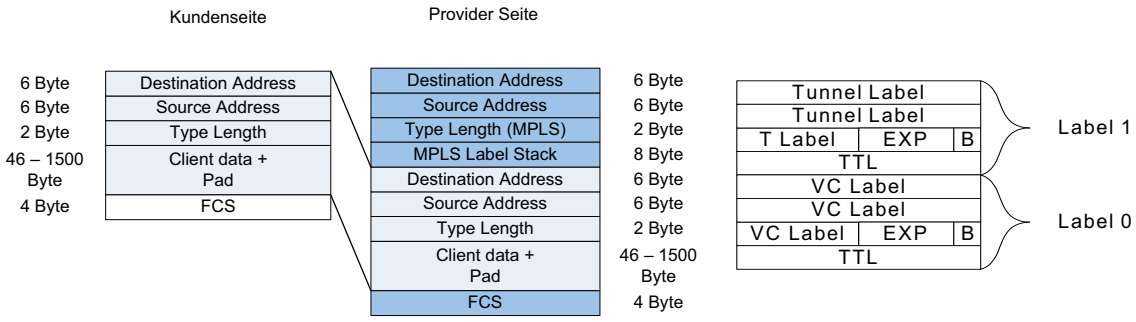
Unterscheidung von Verkehrsarten [San03]. Um diesen verschiedenen Anforderungen gerecht zu werden, müssen die Datenpakete mit zusätzlichen Informationen angereichert werden. Diese Informationen sind zum Teil nur im TZN verfügbar. Sie können aber an verschiedenen Stellen des Betreibernetzwerkes genutzt werden, um unterschiedliche Dienste anzubieten.

Eine mögliche Technik zum Einfügen von Informationen in einen Datenframe ist die Nutzung von VLAN (Virtual LAN) Tags [IEE06a]. Diese haben allerdings einen großen Nachteil. Die Größe der Nutzdaten in einem VLAN Tag ist auf 12 Bit begrenzt. Feingranular aufgelöste Teilnehmerinformationen sind mit nur 12 Bit nicht möglich. Stacked VLANs[17] (IEEE 802.1, Q-in-Q [CGE+04]) würden das Problem abmildern. Jedoch ist herkömmliches Netzwerkequipment nicht notwendigerweise in der Lage, mit Q-in-Q umzugehen.

Da sich Geräte für den Teilnehmerzugang wie DSLAMs an den Rändern von MPLS-fähigen Netzwerken befinden, liegt es nahe, Informationen mittels MPLS-Labels durch das Netzwerk zu transportieren. MPLS (Multi Protocol Label Switching) ist ein Verkapselungsverfahren [RVC01]. In MPLS-fähigen Netzwerken werden Datenpakete auf Basis von in jedem Paket befindlichen MPLS-Labels geroutet. An den Rändern eines MPLS Netzwerkes wird jedem von außen kommenden Paket ein MPLS-Label Stack zugewiesen [RTF+01]. Diese Aufgabe übernehmen sogenannte Label Edge Router (LER). Dann werden die Pakete entlang eines Label Switched Path (LSP) durch das Netzwerk transportiert. Jeder Label Switch Router (LSR) auf dem Weg trifft seine Routingentscheidungen nun nicht mehr nach einem klassischen Routingalgorithmus, sondern nur noch anhand des äußeren Labels im MPLS Label Stack. Damit jedem Paket der richtige Label Stack zugeordnet werden kann, ist ein Label Distribution Protocol (LDP) [And01] notwendig. Am Ausgang des MPLS Netzwerkes sorgt ein weiterer LER dafür, dass der Label Stack wieder entfernt und das Originalpaket wiederhergestellt wird.

Statt wie im Standard vorgesehen MPLS-Labels für das Routing und Switching zu verwenden, sollen beim MPLS-UNI die Labels lediglich dazu verwendet werden, beliebige Informationen zu transportieren. Demzufolge ist es nicht notwendig, ein LDP zu implementieren. Eine derartig reduzierte Funktionalität konnte als kostengünstige Hardwarelösung auf einem FPGA realisiert werden, dem MPLS User Network Interface (MPLS-UNI) [WKD+06, WKT+06]. Die entwickelte Hardware hat die Aufgabe, jedes ankommende Paket im Upstream (vom Kunden zum Netzwerk) mit einen MPLS Label Stack zu versehen und es dann an das Kernnetz des ISPs weiterzugeben. Im Downstream (vom Netzwerk zum Kunden) besteht die Aufgabe des MPLS-UNI darin, MPLS-Label Stacks aus allen Datenpaketen zu entfernen. Im Gegensatz zur klassischen Methode, bei der der MPLS Label Stack zwischen den Headern der Schichten 2 (Ethernet) und 3 (IP) eingefügt wird, findet das von Luca Martini in [Mar05] vorgeschlagene System zur Frameverkapselung Verwendung. Dieses hat den Vorteil, dass MPLS problemlos und unabhängig vom genutzten Schicht-3 Protokoll verwendet werden kann.

[17] Stacked VLANs kann man nutzen, um ein zweites VLAN Tag in einen Datenframe einzufügen.



a) Framestruktur eines Ethernetframe b) Aufbau des MPLS Label Stacks

Abbildung 43 - Struktur von Ethernet Frames ohne und mit MPLS Label Stack.

Die sich ergebende Framestruktur ist in Abbildung 43a dargestellt. Das MPLS-UNI verkapselt den kompletten Ethernetframe. Das FM platziert einen neuen Ethernet Header zusammen mit dem MPLS Label Stack vor dem ursprünglichen Frame. Die Frame-Check-Sequence (FCS) am Frameende ist über den gesamten neuen Frame zu berechnen und die originale FCS zu ersetzen. Der MPLS Label Stack ist in Abbildung 43b dargestellt. Er besteht aus zwei Labels. Normalerweise beschreibt das innere Label den virtuellen Kanal (Virtual Channel – VC), den das Datenpaket nehmen soll. Das äußere, das Tunnel Label, beschreibt den tatsächlichen Weg durch ein MPLS Netzwerk. Die eingetragenen Werte der Labels entsprechen im angewendeten Fall jedoch der zu übertragenden Nutzinformation. Pro Label stehen 20 Bit nutzbare Daten zur Verfügung. Damit können 40 Bit Daten mit dem Labelstack angefügt werden. Sollten 40 Bit nicht ausreichen, ist der Stack um weitere Label erweiterbar.

Implementierungsergebnisse

Das MPLS-UNI wurde mit einem Datenpfad als prototypisches System implementiert, dass auf einem Virtex-20 FPGA basiert. Die Hardwarekosten belaufen sich auf bis zu 4700 Slices. Das sind die Logikeinheiten auf einem FPGA der Firma Xilinx. Die genaue Aufteilung kann Tabelle 5 entnommen werden. Die Größe des MPLS-UNI variiert stark abhängig von der Größe des gewählten Schlüssels. Der maximale Schlüssel kann, wie in Abschnitt 4.2.1 beschrieben, aus bis zu 7 Headerfeldern bestehen. Diese weisen zusammen eine Größe von 216 Bit auf. Bei einer Minimalkonfiguration besteht der Schlüssel nur aus den 8 Bit des DSCP Feldes. In einer typischen Implementierung enthält der Schlüssel die SRC-MAC, DST-IP und ein VLAN Tag. Damit ergeben sich 96 Bit für den Schlüssel und 3400 Slices für das Gesamtsystem. In minimaler Konfiguration benötig das MPLS-UNI 2600 Slices. Es ist festzustellen, dass die eigentlichen EAs (MPLS Labeler und Delabeler) nur sehr wenige Hardwareressourcen benötigen.

Tabelle 5 - Hardwarebedarf des MPLS-UNI.

Modul	Slices minimal/typisch/maximal
MPLS Labeler	120
MPLS Delabeler	101
CA	152/203/343
CPU IF	640
FB	336/535/720
Speicher (intern für 1k Einträge)	633/1049/1594
Externe Beschaltung (Synchronisierungs-FIFOs, Ethernet MAC)	850
∑ Prototypsystem	2600/3400/4700

Das prototypisch implementierte System erreichte auch mit Maximalkonfiguration eine Geschwindigkeit von über 125 MHz, womit die Verarbeitung von GBit-Ethernet problemlos möglich ist.

4.3.2 Traffic Manager

Das sogenannte Overprovisioning ist bei Internet Service Providern weit verbreitet. Unter Overprovisioning versteht man das Anbieten (Verkaufen) von mehr Bandbreite als physikalisch zur Verfügung steht. Das wird unter der Annahme praktiziert, dass zu einem beliebigen Zeitpunkt niemals alle Kunden die ihnen zugewiesene Bandbreite nutzen. Das heißt, die Nutzer teilen sich ein gemeinsames Reservoir an Bandbreite, das kleiner ist als die Summe der zugesicherten Bandbreiten. Es ist jedoch nicht unwahrscheinlich, dass Kunden die ihnen zugesicherte Bandbreite fordern, denn „Internet users who pay a fixed fee have no incentive to limit their use of the network“ [MB00]. Es kann deshalb vorkommen, dass das TZN in Spitzenzeiten überlastet ist. Außerdem ist es sinnvoll, in Zeiten von geringerer Nachfrage wertvollen Kunden mehr Datenvolumen zuzugestehen, als die Menge, für die sie eigentlich bezahlt haben. Geschäftskunden beispielsweise sind für ISPs von größerer Bedeutung als Privatkunden und sollen dementsprechend zuvorkommend behandelt werden.

Der Traffic Manager (TM) mildert das Problem der Verwaltung von großer Last im Kernnetzwerk von ISPs ab [KWD+06, WKT+06]. Lastprobleme und Verstopfungen in Kernroutern werden standardmäßig durch das Verwerfen von Datenpaketen gelöst. Es ist sinnvoll, dieses Verwerfen von Daten nicht willkürlich durchzuführen. Eine bessere Lösung ist, das Verwerfen fair bzw. gesteuert zu gestalten. Damit kann sichergestellt werden, dass alle angeschlossenen Teilnehmer vom Verwerfen im gleichen Maß betroffen sind und die Auswirkungen auf den Einzelnen begrenzt bleiben. Alternativ können wertvolle Kunden davon

verschont bleiben. Im Idealfall ist die Quality-of-Experience (QoE) nur sehr gering oder überhaupt nicht eingeschränkt.

Um eine faire Entscheidung beim Verwerfen zu gewährleisten, sind die Pakete zu kennzeichnen, die verworfen werden sollten, verworfen werden könnten oder nicht verworfen werden sollten. Dazu markiert der TM alle Datenpakete im Upstream mit Farben. Die Pakete sind dann „grün", „gelb" oder „rot" markiert. Diese Funktionalität basiert auf den in [Hei99a] und [Hei99b] beschriebenen Algorithmen. Diese befassen sich mit dem „Single Rate Three Color Marking" ([Hei99a]) und dem „Two Rate Three Color Marking" ([Hei99b]). Um eine solche Farbmarkierung durchführen zu können, misst der TM den Datenverkehr jedes angeschlossenen Teilnehmers. Abhängig von der aktuellen Datenrate und zwei gespeicherten Werten markiert das Modul jeden Frame farblich. Diese Werte sind zum einen die sogenannte Committed Information Rate (CIR) und zum anderen die Burst Information Rate (BIR). Es besteht die Möglichkeit, bereits im TM „rot" markierte Pakete zu verwerfen, und somit eine vereinbarte maximale Datenrate durchzusetzen (Policing).

4.3.2.1 Algorithmus für das Trafficmanagement

Die CIR repräsentiert die Datenrate, mit der ein Teilnehmer auf jeden Fall senden darf. Die BIR gibt eine zusätzlich nutzbare Datenrate an, wenn das Gesamtverkehrsaufkommen dies zulässt. Der Algorithmus basiert auf CIR, BIR und zwei korrespondierenden Zählern CIR_C und BIR_C. Alle vier Werte sind individuell für jeden angeschlossenen Teilnehmer in einem Speicher abgelegt. Der Algorithmus arbeitet wie folgt:

- CIR_C und BIR_C werden periodisch auf die Werte von CIR bzw. BIR zurückgesetzt.
- Ein Messprozess ermittelt die Länge jedes ankommenden Datenframes.
- Die Framelänge wird entweder von CIR_C oder, wenn dieser $Zähler \leq 0$ ist, von BIR_C abgezogen.
- Ist nach der Subtraktion $CIR_C > 0$, wird das Paket grün markiert.
- Ist nach der Subtraktion $CIR_C \leq 0$ und $BIR_C > 0$, wird das Paket gelb markiert.
- Ist nach der Subtraktion $CIR_C \leq 0$ und $BIR_C \leq 0$, wird das Paket rot markiert (verworfen).

4.3.2.2 Farbmarkierung

Die Farbmarkierung der Datenframes kann je nach Konfiguration an unterschiedlichen Stellen des Datenframes erfolgen. Es ist möglich, die Farbmarkierung innerhalb eines MPLS Label Stacks, eines VLAN Tags [IEE06a] oder im DSCP-Feld des IP-Headers [NBB+98] vorzunehmen. Enthält ein Datenframe keine VLANs, MPLS Labels oder IP-Pakete, ist der Frame ohne Farbmarkierung weiterzuleiten.

Wie das TM Modul die Farbmarkierung vornimmt, ist durch einen online konfigurierbaren Modus festgelegt. Dieser ermöglicht die Durchführung der Farbmarkierung nach unterschiedlichen Methoden. Folgende Modi (1)-(4) können zur Laufzeit eingestellt werden:

Bypass:	kein Metering und Colormarking
MPLS:	Farbmarkierung im MPLS-Label (1)
VLAN:	Farbmarkierung im Outer VLAN-Tag eines Frames (2)
MPLS/VLAN:	Farbmarkierung im MPLS-Label oder im VLAN-Tag (1) oder (2)
DSCP:	Farbmarkierung im DSCP-Feld der IP-Pakete (3)
Provider Bridge:	Farbmarkierung im VLAN Tag nach IEEE 801.2ad (4) [IEE06b]

Eine nähere Erläuterung der durch den TM ermöglichten Farbmarkierungsvarianten kann Anhang B entnommen werden.

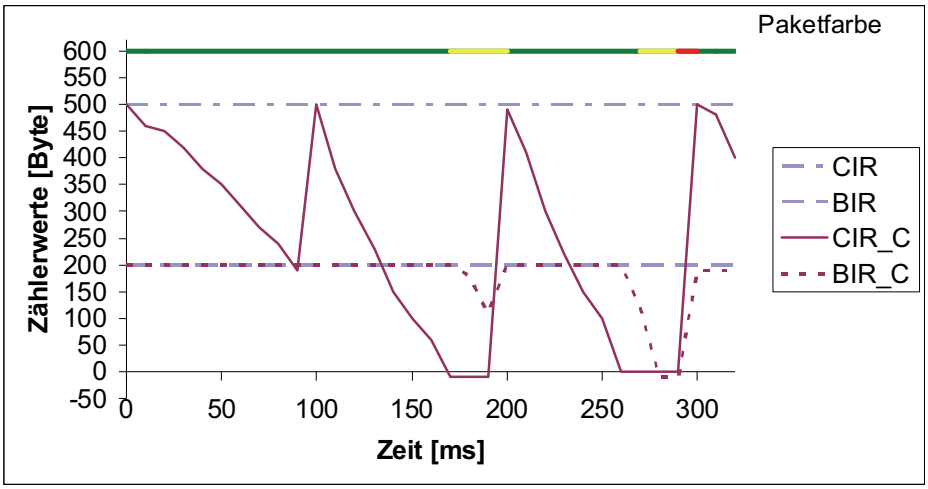


Abbildung 44 - Verlauf der Zählerwerte und Farbmarkierungen der Daten eines Nutzers.

4.3.2.3 Architektonische Besonderheiten

Zur Implementierung des TMs sind Anpassungen des Speichersystems der vorgestellten Architektur vorzunehmen. Wie oben erwähnt, ist es notwendig, die Zähler (CIR_C und BIR_C) regelmäßig zurückzusetzen. Das ist für alle eingetragenen Teilnehmer durchzuführen. Damit die notwendigen Speicherzugriffe nicht zu plötzlichen Spitzen beim Speicherzugriff führen, ist das Rücksetzen aller Teilnehmer in der Rücksetzperiode gleichmäßig verteilt. Ein Beispiel: Bei einer Periode von 100 ms und 1000 angeschlossenen Teilnehmern erfolgt alle 100 µs ein Zurücksetzen der Zähler für einen bestimmten Teilnehmer. Diese Funktionalität ist durch eine zusätzliche Zustandsmaschine in der Speicherlogik realisiert, welche einen etwaigen regulären Speicherzugriff unterbricht. Das ist notwendig, da die Updates der Zähler die höchste Priorität aufweisen.

Des Weiteren dekrementiert der TM, nachdem jeder Datenframe farblich markiert wurde, die korrespondierenden Zähler im Speicher um die Länge des Frames. Es ist also ein Datenkanal

von der AE zum CA vorgesehen, welcher diese Speicherzugriffe ebenfalls verwaltet. Insgesamt ist die Architektur des FMs für den TM (Abbildung 45) etwas komplexer als die vorgestellte allgemeine Architektur (siehe Abbildung 38).

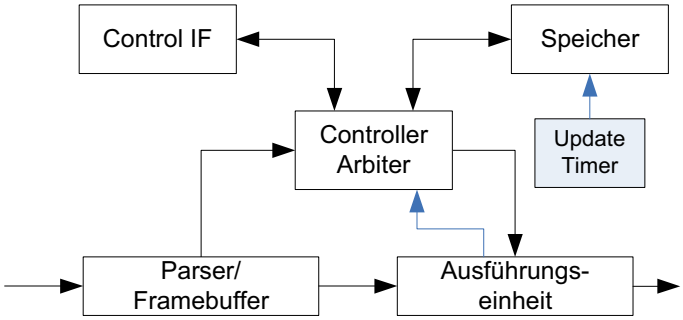


Abbildung 45 - Architektur des TMs (einzelner Datenpfad).

Mit einer Größe von 240 Slices ist die AE des Traffic Managers etwa doppelt so groß wie die des MPLS-UNIs. Die restlichen Elemente entsprechen in Funktionalität und Größe denen des MPLS-UNI. Zusätzlich wurde nur die Logik für den Update-Timer integriert, diese fällt aber nicht weiter ins Gewicht. Aufgrund der zusätzlich notwendigen Speicherzugriffe, um die Zähler zu dekrementieren bzw. zurückzusetzen, ist die Speicherlast beim TM größer als beim MPLS-UNI. Es ist allerdings zu beachten, dass für keine der zusätzlichen Speicheroperationen Suchen im Speicher durchzuführen sind. Das Zurücksetzen der einzelnen Speichereinträge wird linear vorgenommen. Das heißt, das zuständige Modul aktualisiert nacheinander alle Speichereinträge. Die Speichereinträge, die notwendig sind, um die Zähler eines Teilnehmers zu verändern, benötigen ebenfalls keine Suche. Es handelt sich um die Zähler, die für den aktuellen Datenframe bereits gesucht wurden. Damit ist deren Speicherstelle bekannt. Der CA übergibt der AE die korrekte Speicheradresse zusammen mit den Zählerständen.

4.3.3 Mac Address Translation (MAT)

Wegen der wachsenden Zahl angeschlossener Nutzer an Teilnehmerzugangssysteme verkompliziert sich das Management von Adresstabellen in den Netzwerkknoten und den zentralen Switches der Kernnetzwerke. Diese arbeiten auf ISO-OSI Schicht 2. Dort wird die sogenannte MAC Address Table Explosion[18] [CGE+04] zunehmend zu einem Problem. Außerdem ist zu beachten, dass Nutzer in der Lage sind, ihre eigene MAC-Adresse zu manipulieren. Jede MAC-Adresse ist als einzigartig konzipiert. Doppelungen dürfen nicht auftreten. Durch Manipulationen kann es jedoch zu genau solchen Doppelungen kommen. Dadurch werden auch andere ernste Probleme wie MAC Spoofing oder das Address Resolution Protocol (ARP) Spoofing [Bro04] ermöglicht.

[18] MAC Address Table Explosion: Das exponentielle Anwachsen der Größe von MAC-Tabellen.

Die MAT kann diese Problematik lösen. MAT ist eine Technik, die die MAC Adressen von Kunden (Customer-MAC – CMAC) austauscht und am Teilnehmerzugangssystem stattdessen vom ISP gestellte Provider-MACs (PMACs) in jeden Datenframe einsetzt. In umgekehrter Richtung wird die Ersetzung invers ausgeführt. Die Verwendung von PMACs in den Systemen des Providers hat den Vorteil, dass im Kernnetz nur MACs vorkommen, bei denen gesichert ist, dass sie nicht doppelt im System vergeben sein können, da die Wahl der PMACs im Einflussbereich des ISPs liegt.

Die grundlegende Funktionalität ist dabei recht einfach. Im Upstream filtert MAT die CMAC aus einem Datenframe. Diese dient als Schlüssel, um die einzusetzende PMAC zu bestimmen. Da sich das MAT Modul [KWD+06, WKT+06] an einem frühen Aggregationspunkt des Netzwerkes befindet, ist die Anzahl unterschiedlicher Teilnehmer und damit CMACs begrenzt und durch das System verwaltbar. Ist die PMAC im Speicher gefunden, ersetzt die AE die CMAC durch die PMAC und leitet den veränderten Frame weiter. Verschiedene Beziehungen zwischen CMACs und PMACs wie 1:1 oder n:1 sind möglich. Während 1:1 Beziehungen Verwendung finden, um die Netzsicherheit zu erhöhen (nur bekannte und vertrauenswürdige MAC-Adressen kommen im Kernnetz vor), kann eine n:1 Beziehung die Skalierbarkeit der Netze verbessern. Indem n CMACs auf eine PMAC abgebildet werden, verkleinert sich jede MAC-Adresstabelle im Kernnetzwerk um den Faktor n. Damit kann der MAC Address Table Explosion wirksam begegnet werden.

Bei der Verwendung einer n:1 Beziehung ist zu beachten, dass im Downstream keine eindeutige Zuordnung von PMAC zu CMAC möglich ist, um die Übersetzung rückgängig zu machen. Hier dient die Verbindung von PMAC und der DST-IP der Datenpakete zu den Teilnehmern als Schlüssel zur Ermittlung der CMAC. Bei gleicher PMAC unterscheiden sich die Teilnehmer immer noch in ihrer IP-Adresse. So ist es möglich, die korrekte CMAC wieder in den Datenframe einzusetzen. Bei der Bearbeitung von Frames sind verschiedene Sonderfälle berücksichtigt. Das MAT Modul unterstützt sowohl Black Lists (Frames abblocken) als auch White Lists (Frames, bei denen die CMAC nicht ersetzt werden soll). Auch einige spezielle Kommunikationsprotokolle benötigen eine gesonderte Behandlung. Dazu gehören das Address Resolution Protocol [Plu82], das Reverse Address Resolution Protocol ([FMM+84]) und das Dynamic Host Configuration Protocol (DHCP) [Dro97]. Die Behandlung im Einzelnen ist in [Kub08] und [WKD+06] umfassend dargestellt.

Das entwickelte Modul gleicht in seiner grundlegenden Architektur dem MPLS-UNI und dem TM. Im Gegensatz zu den genannten Beispielen sind für die MAT sowohl im Upstream (CMAC → PMAC) als auch im Downstream (PMAC → CMAC) AEs mit Speicherzugriff notwendig. Damit ergibt sich eine Architektur, wie sie in Abbildung 46 dargestellt ist. Das CA Modul für die MAT ist aufwendiger, weil nun jeweils vier parallele Datenpfade in Up- und Downstream, insgesamt acht, miteinander um den Speicherzugriff konkurrieren. Die Größe der AEs für Up- und Downstream beträgt je 180 Slices.

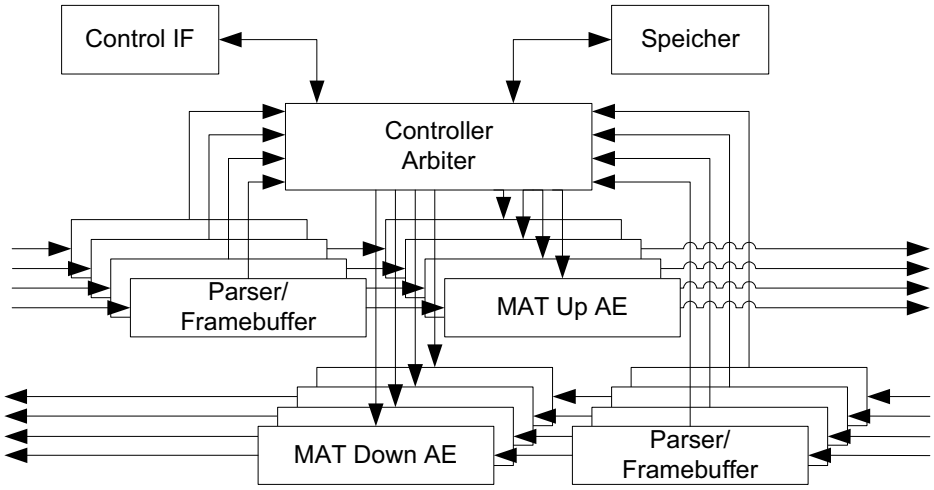


Abbildung 46 - Schematische Darstellung der Architektur des MAT Moduls (vgl. Abbildung 40).

In [KWT+07] ist eine vereinfachte Architektur, die auf MAT basiert, vorgestellt: Simplified MAT (sMAT). sMAT wird zur MAC Address Translation auf der Linecard eines DSLAMs eingesetzt, während MAT auf dem Zentralknoten des DSLAMs oder GPON-Systems zum Einsatz kommt.

4.3.4 Leistungsfähigkeit der funktionalen Module

Die Untersuchung der Leistungsfähigkeit der FMs erfolgte nach [Man00], wo Methoden und theoretische maximale Durchsatzraten definiert wurden. Um die FMs zu untersuchen, wurden die Module mit jeweils vier Datenpfaden implementiert. Damit haben sie eine Kapazität von 4 GBit/s. Zur Untersuchung des Durchsatzes wurden alle Datenpfade mit Ethernetframes in der maximalen Datenrate, die das GBit-Ethernet zulässt, belastet. Ethernetframes unterschiedlicher Größe von minimal (64 Byte) bis maximal (1518 Byte) fanden Verwendung. Neben dem Durchsatz ist auch die Verlustrate bei unterschiedlichen Framegrößen und Eingangsdatenraten untersucht worden. Zur Vermeidung von Sondereffekten, die zu Beginn einer Simulation auftreten können, wenn im System alle Puffer noch leer sind, wurden vor Beginn der Messung 250 Datenframes in das System injiziert. Damit wurde erreicht, dass sich die FMs zu Beginn der Messungen in einem realitätsnahen Zustand befinden. Alle Untersuchungen wurden anhand von ModelSim-Simulationen des entwickelten VHDL-Codes durchgeführt, da keine Hardwareplattform zur Verfügung stand, um vier bidirektionale Datenkanäle einer Kapazität von je 1 GBit/s zu implementieren.

4.3.4.1 MPLS-UNI

Das MPLS-UNI weist zwei Besonderheiten auf. Speichersuchen sind nur im Upstream notwendig, da im Downstream der MPLS-Delabeler Verwendung findet. Dieser muss nur den MPLS-Label Stack aus jedem Datenframe entfernen, wozu keine Suchen im Speicher

notwendig werden. Außerdem werden Datenframes im Upstream durch das Einfügen des MPLS-Label Stacks vergrößert. Deshalb wurden Untersuchungen durchgeführt, um die Performance mit und ohne das Einfügen von Stacks zu ermitteln. Die Graphen in Abbildung 47 zeigen die Leistungseigenschaften des MPLS-UNI. Wie der Abbildung zu entnehmen ist, kommt es nur bei der Verwendung von sehr kleinen Datenframes und Schlüsselmengen, die größer als 2048 sind, überhaupt zu Verlusten.

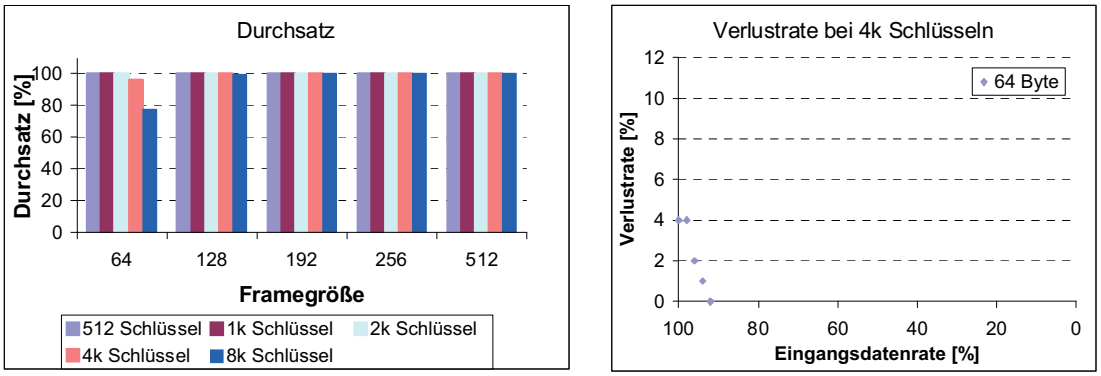


Abbildung 47 - Verhalten des MPLS-UNI mit 4 parallelen Datenpfaden und einer Gesamtdatenrate von 4 GBit/s.

Bei 4k Schlüsseln treten 4% und bei 8k Schlüsseln 23% Verluste auf. Bei einer Framegröße von 128 Byte gehen noch 1% der Frames verloren, wenn 8k Schlüssel verwendet werden. Sinkt die Dateneingangsrate unter 92%, kommt es bei 4k Schlüsseln zu keinerlei Verlusten mehr. Das MPLS-UNI ist damit in der Lage, Datenraten von 4 GBit/s problemlos zu verarbeiten.

Aufgrund des Umstandes, dass alle Datenframes durch das MPLS-UNI verlängert werden, ist es unmöglich, in der Realität bei 100% Eingangsdatenrate Frameverluste zu verhindern. Jeder Datenframe wird um 22 Byte verlängert, wenn ein MPLS-Label eingefügt wird. Minimale Frames wachsen dadurch von 64 auf 86 Byte an. Da der Inter-Frame Gap auch am Modulausgang eingehalten werden muss, kommt es zwangsläufig zu Verlusten.

4.3.4.2 Traffic Manager

Der TM ähnelt in seinem Aufbau dem MPLS-UNI. Der TM ist nur in Upstreamrichtung einzusetzen, weshalb auch nur im Upstream Speichersuchen notwendig sind. Damit entspricht der TM dem MPLS-UNI. Allerdings benötigt der TM zusätzliche Speicherzugriffe, um die Zähler (CIR_C und BIR_C) zu aktualisieren und timergesteuert zurückzusetzen. Deshalb ist zu erwarten, dass die Leistungsfähigkeit des TMs leicht unter der Leistungsfähigkeit des MPLS-UNI liegt. Die Simulationsergebnisse bestätigen diese Annahme. Abbildung 48 stellt die erreichte Performance für ein TM-Modul mit vier parallelen Datenpfaden dar. Es ist zu erkennen, dass größere Verluste als beim MPLS-UNI auftreten. Bei Schlüsselmengen zwischen 512 und 8.192 Schlüsseln und minimalen Frames erreicht der TM einen Durchsatz zwischen 85% und 75%. Bei 128 Byte großen Frames, kommt es noch zu leichten Verlusten, wenn die

Schlüsselmenge 4k (2%) oder 8k (6%) groß ist. Ein Absenken der Eingangsdatenrate auf 56% der maximal möglichen ist notwendig, damit es auch für minimale Frames bei einer Schlüsselmenge von 4k nicht mehr zu Verlusten kommen kann.

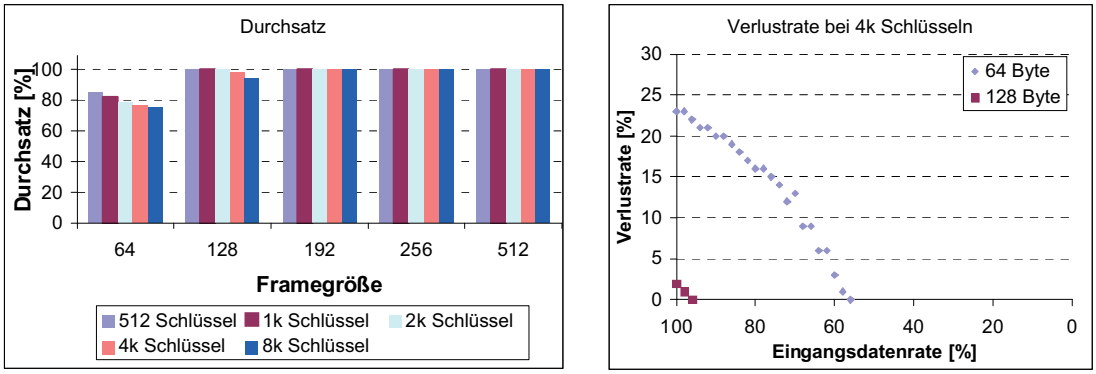


Abbildung 48 - Verhalten des TM mit 4 parallelen Datenpfaden und einer Gesamtdatenrate von 4 GBit/s.

4.3.4.3 MAT

Im Gegensatz zu TM und MPLS-UNI müssen für das MAT-Modul sowohl im Up- als auch im Downstream Speichersuchen durchgeführt werden. Das hat zur Folge, dass effektiv 8 Datenströme mit jeweils 1 GBit/s zu verarbeiten sind. Abbildung 49 verdeutlicht das resultierende Verhalten des MAT Moduls.

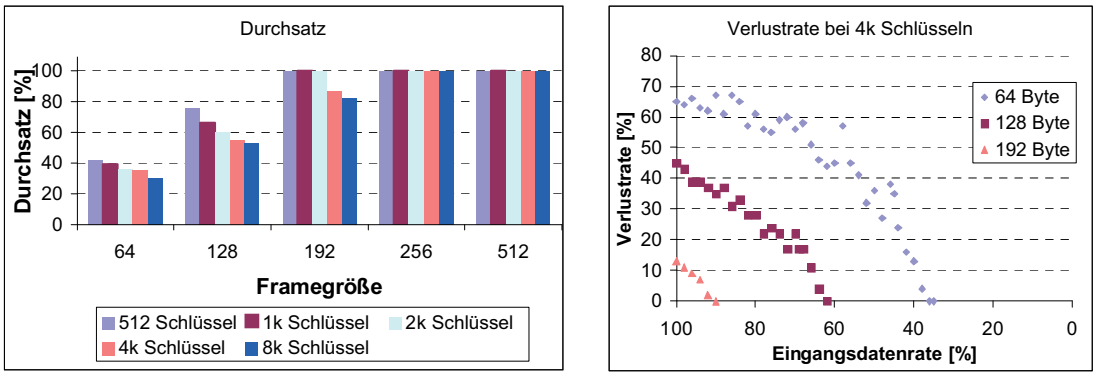


Abbildung 49 - Verhalten von MAT mit jeweils vier parallelen Datenpfaden im Up- und Downstream und einer Gesamtdatenrate von 8 GBit/s.

Wie zu erkennen ist, ist der Durchsatz für minimale Frames verhältnismäßig gering. Abhängig von der Größe der Schlüsselmenge werden nur zwischen 42% (512 Schlüssel) und 30% (8192 Schlüssel) der ankommenden Schlüssel verarbeitet. Die Verlustrate sinkt jedoch mit zunehmender Framegröße. Bei Framegrößen von 256 Byte kommt es in keinem Fall zum

Verwerfen von Daten. Betrachtet man nur eine einzelne Richtung, verhält sich MAT genau wie das MPLS-UNI, wenn keine Frameverlängerung vorgenommen wird (siehe Abbildung 47).

4.3.4.4 Bewertung

Wie schon zuvor dargelegt, ist bei der vorgestellten Architektur die Suche im Speicher der limitierende Faktor des Systems. Der Durchsatz nimmt ab, wenn die Anzahl der Einträge im Speicher zunimmt oder die Größe der gesendeten Datenframes abnimmt. Wenn die Eingangsframes größer als 256 Byte werden, kommt es bei keinem der vorgestellten FMs mehr zu Verlusten. Betrachtet man realistischen Datenverkehr, ist die Performance aller FMs absolut ausreichend, um sie in TZN mit ihren 4 GBit/s breiten Datenpfaden einzusetzen. Realistische Daten bestehen nicht nur aus minimalen Frames bzw. Frames mit konstanter Größe. In der Realität dominieren nach [Agi01] Framegrößen von 64 Byte (35%), 594 Byte (11%) und 1518 Byte (10%) den Verkehr. Die restlichen 44% der Frames sind in ihrer Größe in etwa gleichverteilt (Abbildung 50).

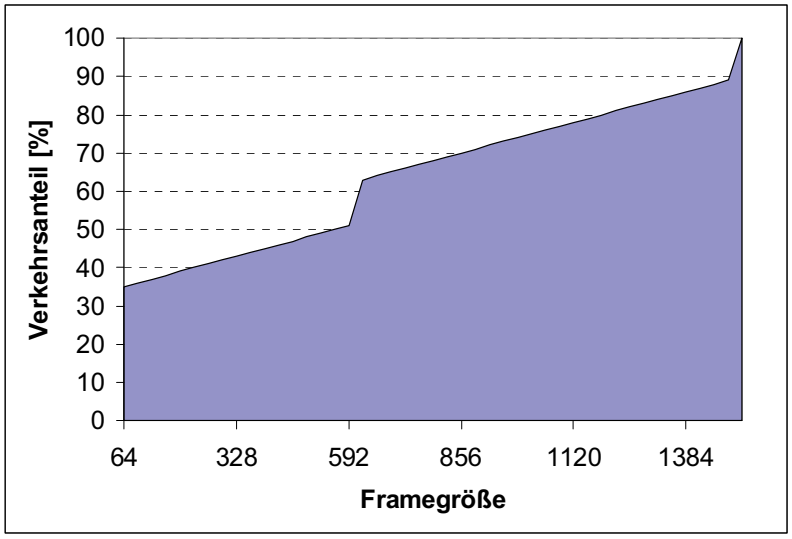


Abbildung 50 - Verteilung von realem Internetdatenverkehr.

Damit beträgt die durchschnittliche Größe eines IP-Pakets 402 Byte, die eines Ethernetframes folglich 420 Byte. Ausgehend von dieser durchschnittlichen Framegröße kommt es nur zu minimalen Verlusten für Datenpfade mit 4 oder sogar 8 Gbit/s. Die durchschnittliche Latenz der Datenframes in den vorgestellten FMs beträgt etwa 130 µs.

4.3.5 IPclip

Speziell im letzten Jahrzehnt hat sich das Internet in ein Massenmedium gewandelt [Kle03]. Die radikale Änderung der angebotenen Dienste stellt Forderungen an die Netzwerkinfrastruktur, auch im Teilnehmerzugangsbereich. Große Nachteile des Internets sind der

eklatante Mangel an Sicherheit, die Anonymität sowie die Komplexität des Netzwerks selbst. Insbesondere das sogenannte „Trust-by-Wire“ (TbW) ist in IP-Netzwerken nicht gegeben.

Unter TbW versteht man das Wissen um die Vertrauenswürdigkeit bzw. die Identität eines Kommunikationspartners, das sich implizit aus der genutzten Kommunikationsverbindung ergibt. TbW beschreibt einen direkten Zusammenhang zwischen einer Art Nutzeridentifikation, z.B. einer Netzwerkadresse, einem Login-Namen oder einer Telefonnummer und einer physikalischen Verbindung bzw. einer geographischen Position. Im Bereich der Telefonie ist TbW gegeben. Über die Telefonnummer der Teilnehmer, welche bei ISDN und Mobiltelefonen im Allgemeinen übertragen wird, ist die Telefonleitung identifiziert. Mit ihr ist ebenfalls der Teilnehmer bzw. der Anschlussinhaber identifizierbar. Sollte eine Telefonnummer unterdrückt werden, ist ein gewisses Misstrauen angebracht. Man kann sich einer solchen Kommunikation verweigern. Es ist also relativ leicht, Kontakt mit wenig vertrauenswürdigen Partnern zu vermeiden. In IP-Netzwerken besteht diese Verbindung nicht mehr. Der einzige Weg, Partner zu identifizieren, ist die IP-Adresse des Senders (SRC-IP). Diese kann jedoch einfach gefälscht werden, ist zufällig vergeben oder verschleiert worden. Bei einer Kommunikation über das Internet, das ein IP-Netzwerk ist, kann ein Teilnehmer nie ganz sicher sein, mit wem er kommuniziert. Dieses Nichtvorhandensein von TbW ist ein entscheidendes Problem im Internet. Es behindert Dienste wie Onlinebanking oder stört Dienstgüten. Das ist zum Beispiel bei E-Mail der Fall. Der überbordende Empfang von Spam, von nicht erwünschten E-Mails, ist ein Nachteil dieses eigentlich sehr sinnvollen Dienstes. Voice-over-IP (VoIP) ist ein weiteres interessantes Beispiel eines Dienstes, der von TbW profitieren könnte. Zukünftig wird erwartet, dass ein Großteil der Telefonkommunikation auf VoIP umgestellt wird. Damit treten allerdings neue Probleme in Erscheinung. Die Anruferrückverfolgung ist nicht mehr ohne weiteres möglich. Aus den Daten der IP-Pakete kann weder genau auf den Anrufer noch dessen Position geschlossen werden. Klassische Dienste, wie die Notrufweiterleitung an die zuständige Leitstelle, sind nicht mehr möglich.

Um die genannten Probleme zu lösen, wurde mit der Architektur der Internet Protocol-Calling Line Identification Presentation (IPclip) ein Mechanismus entwickelt, der TbW in IP-Netzwerken bieten kann [KWD+08a, WKD+08].

4.3.5.1 IPclip-Mechanismus

Der Begriff IPclip leitet sich vom CLIP (Calling Line Identification Presentation) Mechanismus in ISDN-Telefonnetzen ab. Im herkömmlichen Gebrauch ist CLIP eine optionale Funktion. Mit CLIP kann die Telefonnummer des Anrufers an den Angerufenen übertragen werden. Das ermöglicht eine genaue Identifikation des Anrufers. Im Falle von IP kann die IP-Adresse nicht als Äquivalent zur Telefonnummer betrachtet werden, denn IP-Adressen verweisen nicht auf physische Leitungen. Des Weiteren sind aus IP-Adressen keine verlässlichen Schlüsse auf die geographische Herkunft eines Datenpakets zu ziehen. Das gilt nicht für Telefonnummern. Diese haben eine genau definierte Herkunft, einen Hausanschluss.

Damit ist eine genaue Verortung möglich. IPclip verwendet ausgewählte Prinzipien der ISDN CLIP Funktionalität in IP-Netzwerken, um die oben angesprochenen Dienste zu ermöglichen bzw. zu verbessern.

Durch IPclip ist ein Nutzer und vor allem seine aktuelle geographische Position identifizierbar. IPclip verwendet dazu ein Tupel, das die aktuelle Position des Nutzers und zusätzliche Informationen verwendet. Die IP-Adresse kann unter Umständen bereits einen Nutzer identifizieren. Seine Position, die Ortsinformation (OI), muss aber in jedem Fall Teil der Zusatzinformationen sein.

Um derartige OI in einem globalen Maßstab einsetzen zu können, fügt IPclip sie als IP-Option in jedes IP-Paket ein. IP-Optionen sind optionale Felder im Header des IP-Protokolls. Das IP-Protokoll wurde gewählt, weil es das gesamte Internet umspannt und eine Ende-zu-Ende Kommunikationsverbindung zwischen Teilnehmern bzw. ihren Geräten anbietet. Die Struktur und die Größe von IP-Optionen sind im Protokollstandard definiert [Usc81b]. IPclip ist durch die Verwendung von IP-Optionen eine standardkonforme Lösung, um die zusätzliche OI mit IP-Paketen zu transportieren. IP-Optionen dürfen als Teil des IP-Headers maximal 40 Byte lang sein. Diese Datenfelder können frei für die angestrebte OI verwendet werden. Netzwerkgeräte, die sich standardkonform verhalten, können IPclip-Informationen verarbeiten, wenn sie „IPclip-capable“ sind. Ansonsten wird die unbekannte IP-Option ignoriert. In jedem Fall jedoch sind sie in der Lage, den IP-Header zu verarbeiten, ob mit oder ohne IPclip-Option. Das ist notwendig, damit auch nach einer Einführung von IPclip Interoperabilität gewährleistet ist. IPclip bietet darüber hinaus die Möglichkeit, in IP-Optionen befindliche IPclip-Optionen zu entfernen, damit diese nicht bis zu einem angeschlossenen Endnutzer weitergeleitet werden. Das kann aus datenschutzrechtlichen Gründen notwendig sein.

4.3.5.2 Validierung von Ortsinformationen

Es ist möglich, dass Nutzer bereits auf ihren Endgeräten OI in Datenpakete einfügen. In diesem Fall überprüft IPclip diese auf Vertrauenswürdigkeit. Das geschieht durch den Vergleich der OI aus der Option mit der konfigurierten Position des IPclip-Moduls. Liegt die OI innerhalb eines Vertrauensbereichs[19] um die Position von IPclip, kann die OI als vertrauenswürdig betrachtet werden. Die Größe des Rechtecks kann konfiguriert werden. Entsprechend der Ergebnisse dieser Überprüfung setzt IPclip zwei Flags in der IPclip-Option und überschreibt gegebenenfalls die vom Nutzer eingetragenen OI mit der eigenen Position. Das geschieht nur, wenn IPclip entsprechend konfiguriert und die OI nicht vertrauenswürdig ist.

4.3.5.3 Position im Netzwerk

IPclip soll Ortsinformationen in Pakete einfügen oder diese validieren. Dazu muss das entsprechende Hardwaremodul geographisch möglichst nah an der Position der Nutzer platziert

[19] Der Vertrauensbereich ist ein Rechteck, die sogenannte Subscriber Catchment Area – SCA.

werden. Das ermöglicht genaue OI. Gleichzeitig muss IPclip außerhalb des Zugriffs von Endnutzern liegen, damit Manipulationen der OI verhindert werden. Deshalb ist IPclip auf den Linecards oder dem Zentralknoten des IP-DSLAMs zu implementieren. Vorausgesetzt, dass der den IP-DSLAM kontrollierende ISP als vertrauenswürdig betrachtet wird, kann an dieser Stelle im Netzwerk das erste Mal vertrauenswürdige Ortsinformation in IP-Pakete eingefügt werden. Vertauen in den ISP bildet die Grundlage für einen sinnvollen Einsatz von IPclip in IP-Netzwerken.

4.3.5.4 IPclip-Optionsformat

IPclip-Optionen, nicht zu verwechseln mit IP-Optionen, enthalten Ortsinformationen und werden mittels IP-Optionen in IP-Pakete eingebettet. Eine IPclip-Option ist ein möglicher Wert einer IP-Option. Abgesehen von Sonderfällen, sind IP-Optionen im TLV (Type-Lenght-Value) Format kodiert. Nähere Informationen sind [Usc81b] zu entnehmen. Als Ortsinformation, die in einer IPclip-Option vorhanden sein kann, sind im Moment vier unterschiedliche Formate definiert:

- OI als Koordinaten des *Global Positioning Systems* (GPS) [NME02]
- OI als *Geospacial Location Information* (GLI) [PSL04]
- OI als GPS mit zusätzlichen Informationen des Teilnehmerzugangsknotens auf dem IPclip implementiert ist (Knoten ID der IP-DSLAMs und Portnummer)
- OI als GLI mit Knoten ID und Portnummer

Zusätzlich zu den OI enthält eine IPclip-Option Flags, die Informationen über die Quelle der OI und deren Vertrauenswürdigkeit enthalten.

Der genaue Inhalt einer IPclip-Option und der resultierende Aufbau einer IP-Option ist Anhang C zu entnehmen.

4.3.5.5 Hardwareimplementierung

Die Architektur der IPclip-Hardware [WKD+08] unterscheidet sich etwas von der bereits vorgestellten generischen Architektur für FMs. Um die oben erläuterten Aufgaben zu erfüllen, wurden verschiedene Submodule für IPclip entwickelt. Das Blockschaltbild des FMs als Kombination der Module ist Abbildung 51 zu entnehmen. Das System besteht aus:

- IPoE MTU Adaptation Module (MAM)
- PPPoE MTU Adaptation Module (PAM)
- Parser
- Option Verification Module (OVM)
- Additional Information Adder (AIA)
- Additional Information Remover (AIR)

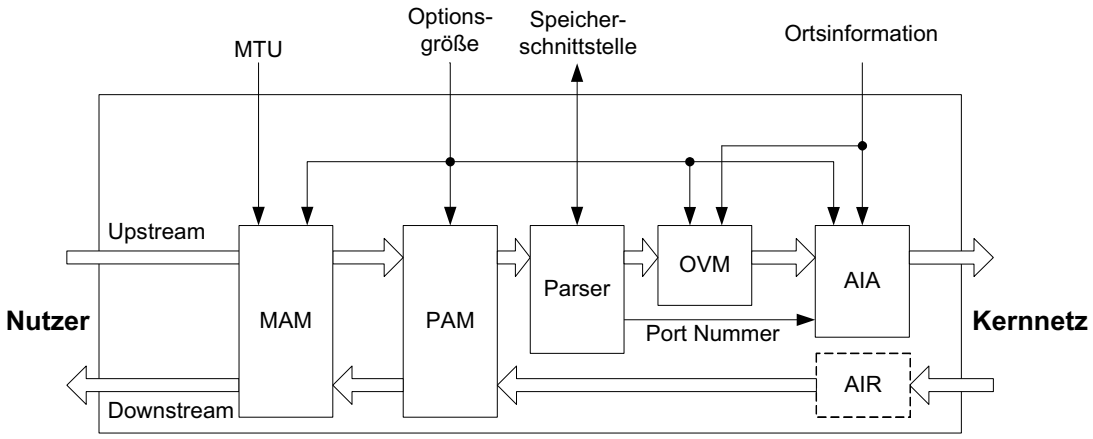


Abbildung 51 - Architektur von IPclip.

IPoE MTU und PPPoE Adaptation Module

IPclip fügt IP-Optionen in IP-Pakete ein, wodurch sich die Größe der IP-Pakete vergrößert. Das hat zur Folge, dass ausgehende Datenpakete die erlaubte Größe (engl. Maximum Transmission Unit – MTU) überschreiten und dann auf ihrem Weg durch das Netzwerk entweder verworfen oder fragmentiert werden. Das sollte vermieden werden [Cis06]. Deshalb sorgen die Module MAM und PAM dafür, dass die MTU auf der Übertragungsstrecke um die Größe einer IP-Option herabgesetzt wird. Damit gewährleistet IPclip, dass die erlaubte MTU auch mit eingefügter IPclip-Option nicht überschritten wird. Im IP-DSLAM, in dem IPclip eingesetzt werden soll, treten zwei Protokolle auf. Werden IP-Pakete über Ethernet übertragen, spricht man von IP over Ethernet (IPoE). Bei diesem Protokoll sorgt MAM für das Setzen der korrekten MTU. Bei Verkapselung der IP-Pakete im Point-to-Point Protocol (PPP), das dann über Ethernet übertragen wird, spricht man von PPPoE. Bei diesem Protokoll sorgt PAM für ein korrektes Setzen der MTU.

Parser

Der Parser entspricht in seiner Funktionalität weitgehend dem Framebuffer, der in Abschnitt 4.2.1 vorgestellt wurde. Er hat die Aufgabe, für jedes Paket den Eingangsport zu ermitteln, damit dieser als Teil der IPclip-Option eingetragen werden kann. Die Portinformation ist für zwei der vier definierten Optionsformate notwendig. Für IPclip nutzt der Parser die SRC-IP und einen VLAN-Tag als Schlüssel. Die ermittelte Portnummer übergibt der Parser zusammen mit dem IP-Paket an das OVM Modul.

Option Verification Module

Das Option Verification Module überprüft für jedes ankommende Paket die Vertrauenswürdigkeit der OI, falls es solche enthält. Dazu wird überprüft, ob die angegebene Position innerhalb eines Vertrauensbereichs (Subscriber Catchment Area – SCA) um die Position von IPclip liegt. Da eine IPclip-Option OI in unterschiedlichen Formaten enthalten kann (GPS-OI oder GLI-OI), müssen alle Formate unterstützt werden. Damit der notwendige Hardwareaufwand begrenzt ist, wandelt das Option Verification Module alle ankommenden Formate in ein einziges um. Dieses wird dann verwendet, um mit der anliegenden eigenen Position des IPclip-Moduls verglichen zu werden. Um auf jedem beliebigen Breitengrad eine SCA bestimmter Ausdehnung zu validieren, sind umfangreiche mathematische Operationen notwendig. Details dazu können [WKD+08] entnommen werden. Die Operationen erfordern einige Multiplikationen. Um Ressourcen zu schonen, ist ein sequentieller Multiplizierer implementiert. Eine schematische Darstellung der Abläufe in der Hardware kann Abbildung 52 entnommen werden.

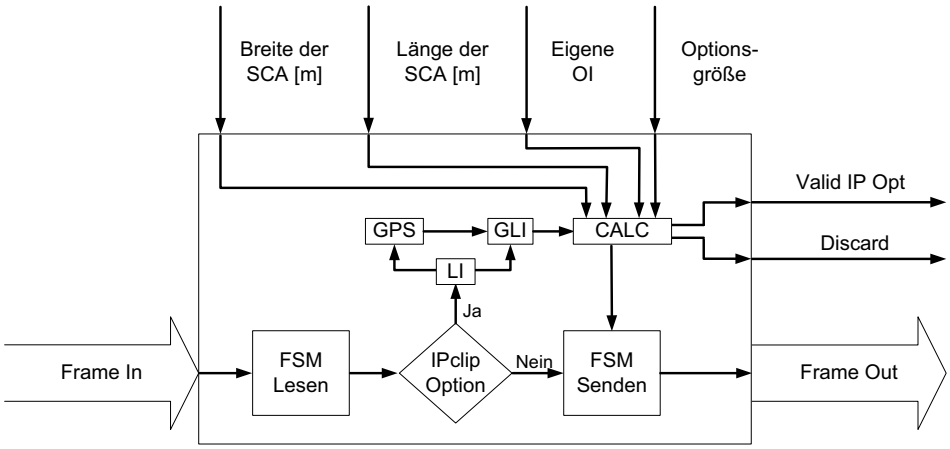


Abbildung 52 - Validierung von Ortsinformationen durch das Option Verification Module.

Additional Information Adder

Der Additional Information Adder stellt die eigentliche Ausführungseinheit von IPclip dar. AIA erzeugt für jedes ankommende Paket die OI und die IPclip-Option. AIA tut das nur für den Fall, dass noch keine gültige OI im IP-Paket vorhanden ist. Alternativ ist AIA auch in der Lage, nur die Portnummer und die ID des DSLAMs in die IP-Option einzufügen. In Abbildung 53 ist der Ablauf der Paketbehandlung durch AIA schematisch dargestellt.

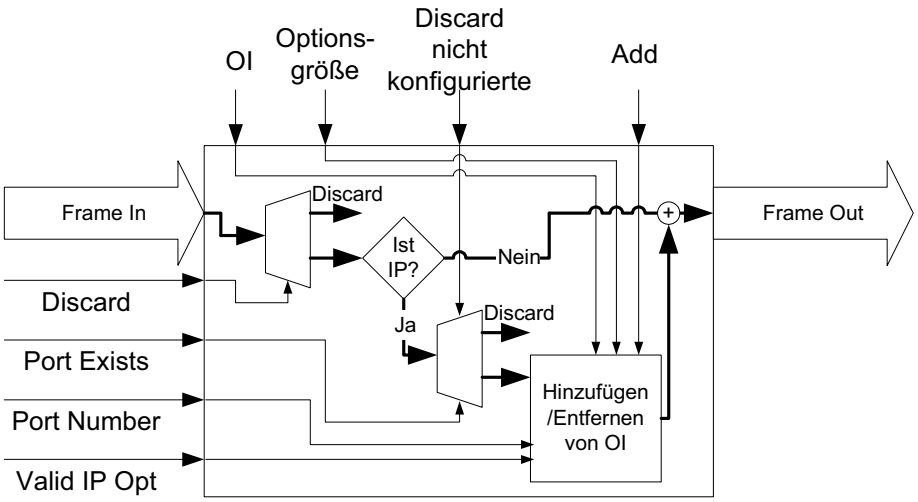


Abbildung 53 - Verarbeitung von IP-Paketen durch AIA.

Wenn bereits mindestens eine IPclip-Option im Paket vorhanden ist, gibt es vier Möglichkeiten der Paketbehandlung:

- Eine IPclip-Option mit OI wurde erfolgreich validiert: Das Paket wird nicht verändert. In diesem Fall wird entweder keine zusätzliche Information hinzugefügt, oder die DSLAM-ID und die Portnummer werden eingefügt. Das ist abhängig von der Konfiguration von AIA.
- Eine IPclip-Option mit OI wurde nicht erfolgreich validiert: Die OI wird entfernt und durch die OI des IPclip-Moduls ersetzt.
- Mehr als eine IPclip-Option mit OI befindet sich im Paket und zumindest eine wurde erfolgreich validiert: Die letzte validierte OI wird beibehalten. Alle anderen Optionen werden aus dem Paket entfernt. Optional werden die DSLAM-ID und die Portnummer eingefügt.
- Mehr als eine IPclip-Option mit OI befindet sich im Paket und keine wurde erfolgreich validiert: Alle Optionen werden entfernt und durch eine IPclip-Option mit der eigenen OI des IPclip-Moduls ersetzt.

Additional Information Remover

Der Additional Information Remover ist ein sehr einfaches Modul. Er hat die Aufgabe, in einem IP-Paket vorhandene IPclip-Optionen zu entfernen. Er wird im Downstream des IPclip-Moduls eingesetzt. Damit verhindert er, dass Endnutzer die OI ihrer Kommunikationspartner ermitteln. Das kann in verschiedenen Anwendungsszenarien gewollt sein. Es ist aber grundsätzlich als datenschutzrechtlich problematisch zu betrachten. Es ist deshalb möglich, per Konfiguration festzulegen, ob der Additional Information Remover im Downstream eingesetzt werden soll oder nicht.

Hardwarekosten und Performance von IPclip

Wird IPclip in der vorgestellten Form implementiert, ist das System in der Lage, Daten mit Raten von 1 GBit/s zu verarbeiten. Ein voll funktionsfähiger Prototyp ist auf Basis der Virtex-4FX20 FPGAs auf einem ML405 Entwicklungsboard implementiert worden. Der Prototyp benötigt insgesamt 7486 Slices an Logikressourcen und 55 BRAMs. Der eingesetzte FPGA ist damit weitgehend ausgefüllt. Die genaue Verteilung der Ressourcen der Submodule kann Tabelle 6 entnommen werden.

Tabelle 6 - Ressourcenbedarf des IPclip-Prototyps.

Modul	Slices	BRAMs
IPoE MTU Adaptation Module	786	1
PPPoE Adaptation Module	163	0
Parser	832	11
Additional Information Adder	1019	4
Option Verification Module	2491	2
Additional Information Remover	519	6
EMAC + Verbindungslogik + Logik für den Prototypen	1700	31
Gesamtsystem des IPclip-Prototyps	7486	55

Es fällt auf, dass insbesondere das Option Verification Module sehr viele Ressourcen benötigt. Das ist auf Grund seiner komplexen Funktion nachvollziehbar.

Um die Leistungsfähigkeit des Systems zu evaluieren, wurde das worst-case Szenario mittels einer ModelSim Simulation des VHDL-Codes untersucht. IP-Pakete unterschiedlicher Größe wurden in das System bei einer Eingangsdatenrate von 100% induziert. Daten wurden sowohl mit als auch ohne IPclip-Optionen versendet. Wie aus Abbildung 54 zu entnehmen ist, weist das System mit minimalen Frames eine Verlustrate von 25% auf, wenn OI in die Frames eingefügt werden muss. Das ist unvermeidbar, da die Framegröße jedes Frames von 64 auf 89 Byte anwächst. Wenn realistischer Datenverkehr gesendet wird, reduziert sich die Verlustrate auf 6%. Wenn jedoch alle ankommenden Frames bereits OI enthalten, die Framegröße mithin nicht ansteigt, gibt es keinerlei Verluste. Im Durchschnitt erzeugt IPclip eine Verzögerung von 700 Takten. Das einspricht bei 125 MHz Frequenz einer zeitlichen Verzögerung von 5,6 µs. Da IPclip nicht alle Daten verarbeiten kann, wenn OI einzufügen ist, erhöht sich in diesem Fall die Verzögerung, da alle internen Puffer gefüllt werden. Dann erreicht die Verzögerung bis zu 15,2 µs. Bei der Simulation von Verkehr mit 50% Pakten, die bereits OI enthalten, betrug die Verzögerung im Mittel 5,5 µs. Im VoIP-Bereich sind beispielsweise Verzögerungen von bis zu 30 ms hinnehmbar. Zusätzliche Verzögerungen im niedrigen µs-Bereich fallen nicht ins Gewicht.

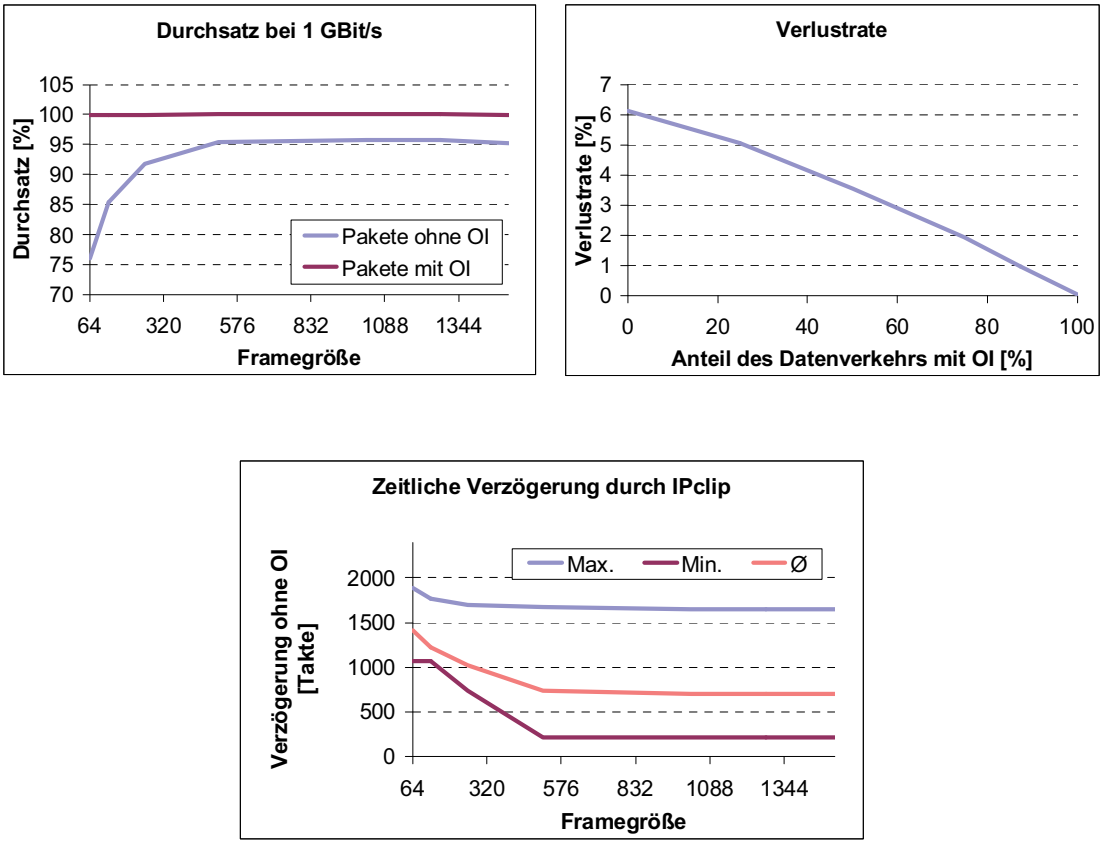


Abbildung 54 - Performance des IPclip-Prototyps. Dargestellt sind der Durchsatz des Systems bei voller Last (1 GBit/s), die Verlustrate in Abhängigkeit vom Anteil der Datenframes, die bereits über OI verfügen und die durch IPclip in den Datenpfad eingefügte Verzögerung.

4.3.5.6 Anwendungsfälle

Die Anwendungsmöglichkeiten für IPclip und damit die Verwendung von vertrauenswürdigen Ortsinformationen im Netzwerk sind mannigfaltig. Im Folgenden sind drei untersuchte Anwendungsfälle dargestellt: VoIP-Notrufe [KWD+08a], die Bekämpfung von Phishing [KWD+08b] und die Bekämpfung von Spam [KWD+08c]. Alle Szenarien sind mit dem entwickelten Prototyp abbildbar.

VoIP-Notrufe

VoIP-Notrufe sind ein sehr wichtiges Thema in der aktuellen Forschung im VoIP-Bereich [NNL06, RP06]. VoIP-Dienste haben den großen Vorteil, dass auf sie von überall zugegriffen werden kann, wo eine Verbindung zum Internet besteht. Auf der anderen Seite stellt die starke Mobilität der Nutzer ein großes Problem für die Bereitstellung korrekter Ortsinformationen dar. Im Falle eines Notrufes ist diese Information jedoch essentiell. Für herkömmliche Telefonnetze

ist die Herkunft eines Anrufes bekannt. Dieses TbW-Modell ist im nomadischen VoIP-Umgebungen nicht gegeben. Um VoIP-Notufe zu unterstützen, gibt es unterschiedliche Ansätze, wie sie in [HLP06] beschrieben sind. Es gibt einige regionale Ansätze, die hauptsächlich auf Datenbanksuchen und der manuellen Pflege der korrekten Ortsinformation durch Nutzer und/oder Service Provider basieren. Einige Service Provider bieten überhaupt keine Notrufdienste über VoIP an.

In den meisten VoIP-Anwendungen wird das Session Initialization Protocol (SIP) [HSS+99] zum Verbindungsaufbau verwendet. Im Moment müssen Nutzer ihrem Provider ihre aktuelle Position zukommen lassen. Diese Information muss immer auf dem aktuellen Stand sein. Im Falle eines Notrufs empfängt dann der Nutzer seine eigene OI mittels DHCP [Dro97]. DHCP bietet Mechanismen an, um OI in Form von Adress- oder GLI-Form zu nutzen [PSL04, SCH97]. OI kann auch von externen Quellen, wie Location Information Servern, bezogen werden [NEN05]. Dabei bilden Location-to-Service Translation (LoST) Server die OI auf den Uniform Resource Identifier (URI) der zuständigen Notrufzentrale ab. Dadurch wird eine Verbindung hergestellt. Diese Systeme sind in [RP06] und [RSP+08] erläutert. Zusammenfassend ist festzustellen, dass alle aktuellen Systeme auf die Mitarbeit und die Zuverlässigkeit der Endnutzer angewiesen sind [NNL06, MRS+05]. Entweder die Nutzer stellen ihre OI einem Dienst zur Verfügung oder fügen sie bei einem Notruf selbst hinzu. Beide Alternativen sind zwar genau aber nicht sehr zuverlässig.

IPclip ist in der Lage, das Notrufproblem zu lösen. IPclip, auf einem Teilnehmerzugangssystem (IP-DSLAM) implementiert, übernimmt die Zuordnung der OI. IPclip kann so konfiguriert werden, dass entweder jedem IP-Paket oder nur den Paketen, die SIP enthalten, OI hinzugefügt wird. Dadurch werden alle anderen Ansätze obsolet. Grundlegend gibt es zwei mögliche Quellen von Ortsinformationen. Falls das VoIP-Gerät, das ein Nutzer verwendet, ein GPS-Modul oder ähnliches besitzt, kann die genaue Nutzerposition zusammen mit dem Notruf übertragen werden. Die GPS-Koordinaten können dann von der Notrufleitstelle verwendet werden, um die richtigen Maßnahmen einzuleiten. Wenn der Nutzer keine oder offensichtlich inkorrekte OI versendet (Falsche OI kann identifiziert werden, falls die angegebenen Koordinaten außerhalb der SCA des IPclip-Moduls liegen.), kann IPclip nach der Überprüfung der Pakete seine eigene Position zusammen mit der ID des Teilnehmerzugangssystems und dem Eingangsport in die IP-Pakete eintragen und übertragen.

In jedem Fall liegen bei der Notrufleitstelle korrekte und vertrauenswürdige Daten vor, um einen Notrufer zu lokalisieren. Verglichen mit dem Stand der Technik bietet jede der zwei Möglichkeiten Notrufern eine substantielle Verbesserung von Notrufdiensten, wobei die von Nutzern bereitgestellten Daten potentiell genauer sind, als die durch IPclip im Teilnehmerzugangssystem. Weitere und ausführliche Informationen zu VoIP-Notrufen können [KWD+08a, DKW+08a, WKD+08] entnommen werden.

Bekämpfung von Phishing

Ein zweites interessantes Anwendungsgebiet von IPclip stellt die Bekämpfung des sogenannten Phishings dar [KWD+08b]. Phishing hat das Ziel, private Daten wie Logins, Passwörter oder Kreditkartennummern von Internetnutzern zu stehlen. Die Daten werden dann direkt missbraucht oder zum Zwecke des Missbrauchs an Dritte veräußert. Die häufigste Form des Phishings wird mittels E-Mail initialisiert [Lea07]. In diesen befinden sich dann Links zu Webseiten, deren URL (Uniform Resource Locator) und Layout einer Bankwebsite zum Verwechseln ähneln. Gibt ein Kunde dann seine Zugangsdaten ein, werden diese protokolliert und später missbräuchlich verwendet. Es gibt die unterschiedlichsten Maßnahmen, um Phishing zu begegnen. In [Cal05, CEH+06] und [PZW+07] sind einige beschrieben. Keine der Maßnahmen kann als perfekte Lösung betrachtet werden. Erfolgreich kann Phishing nur auf globaler Ebene bekämpft werden [ITU07, Ant07]. Ein Beispiel dafür ist das Konzept für Domain Keys Identified Mail (DKIM) [ADL+07]. Yahoo und Ebay nutzen DKIM bereits.

Der Einsatz von IPclip gegen Phishing soll im Folgenden am Onlinebanking-Dienst erläutert werden: Während eines Phishingangriffs werden Nutzer durch E-Mails dazu gebracht, auf eine falsche, zum Verwechseln ähnliche Bankwebsite zu gehen. Dort geben sie ihre Zugangsdaten ein, welche gestohlen werden. IPclip kann dazu verwendet werden, derartige Angriffe bedeutend zu erschweren. Die OI und die Flags in der IPclip-Option der Datenpakete werden genutzt, um die Vertrauenswürdigkeit der angesteuerten Bankseite zu überprüfen. Wenn eine Bank die geographische Position ihres Webservers veröffentlicht, muss jede Webseite, die vorgibt, die Seite der Bank zu sein, genau diese Daten in ihrer IPclip-Option haben. Würde nun der Angreifer die eigenen Pakete mit gefälschten OI versehen, würde das beim Validierungsprozess erkannt werden (siehe Kapitel 4.3.5.2). In diesem Fall ersetzt IPclip die OI und setzt die Statusflags auf `network provided/untrusted`. Der Browser des Bankkunden kann diese Informationen erkennen und informiert den Bankkunden, dass die angesteuerte Webseite potentiell gefährlich ist. In diesem Szenario müssen die Statusflags, wie in Tabelle 7 angegeben, interpretiert werden. Nur in dem Fall, dass die OI vom Webseitenbetreiber eingetragen und sie validiert wurde, muss die Webseite nicht geblockt werden. Das Validieren ist möglich, indem die korrekten OI der Bankseite in einer Datenbank hinterlegt werden, mit der die OI verglichen wird. Aus diesem Grund muss ein vertrauenswürdiger Webseitenbetreiber immer gültige OI bereitstellen. Das Szenario ist in [KWD+08b] umfassender und genauer ausgeführt. Es kann natürlich argumentiert werden, dass Banken aus Sicherheitserwägungen kein Interesse haben, die Koordinaten ihrer IT-Infrastruktur öffentlich zu machen. In diesem Fall ist es möglich, die Ortstinformationenen zu verschlüsseln, indem man z.B. eine geeignete Hashfunktion verwendet. Die Verschlüsselung muss im IPclip-Modul stattfinden. Die Bank selber muss ihre OI weiter unverschlüsselt versenden, damit IPclip diese validieren kann, bevor sie verschlüsselt werden. Die öffentlich zugängliche Datenbank muss dann nur für jede Bank den verschlüsselten Wert der OI enthalten. Stellt der Browser fest,

dass die OI `user provided/trusted` ist, vergleicht er die OI mit den abgelegten Daten. So kann sichergestellt werden, dass sich ein Bankkunde tatsächlich auf der richtigen Webseite befindet. Gleichzeitig kann vermieden werden, dass verwertbare Ortsinformationen öffentlich werden. Sogar das Aufspüren von Betrügern, die sich des Phishings bedienen, wird durch IPclip ermöglicht. [KWD+08b] beleuchtet diesen Aspekt des Anwendungsfalls ebenfalls näher.

Tabelle 7 - Flags der IPclip-Option.

Flag	Quelle/ Vertrauenswürdigkeit	Beschreibung
00	`user provided untrusted`	Die IPclip-Option wurde von einem Webserver-betreiber eingetragen und durch IPclip als nicht vertrauenswürdig validiert. Die Webseite muss geblockt werden.
01	`user provided/ trusted`	Die IPclip-Option wurde von einem Webserver-betreiber eingetragen und durch IPclip als vertrauenswürdig validiert. Die Webseite ist vertrauenswürdig
10	`network provided/ untrusted`	Die IPclip-Option wurde von einem Webserver-betreiber eingetragen und durch IPclip als nicht vertrauenswürdig validiert und deshalb durch eine korrekte IPclip-Option ersetzt. Die Webseite muss geblockt werden.
11	`network provided/ trusted`	Es wurde keine IPclip-Option durch den Webseitenbetreiber eingetragen. Im Teilnehmerzugangsknoten wurde eine neue IPclip-Option in das IP-Paket eingefügt. Die Webseite muss geblockt werden

Anti-Spam Mechanismen

Einen dritten Anwendungsfall stellt die Bekämpfung von E-Mail-Spam dar. Die prominenteste Form ist die unerwünschte Massen-E-Mail (Unsoliceted Bulk E-Mail – UBE). Sie wird unaufgefordert an eine sehr große Zahl von Empfängern versendet. Eine weitere Form ist die mittelbare Spam-E-Mail (Collateral Spam). Derartige Mails entstehen durch das Antworten auf Spam, dessen Absenderadresse manipuliert ist. Dadurch erhalten unbeteiligte Dritte die Antwort, welche für diese wiederum Spam darstellt. Diese und andere Spamformen und deren Verteilung sind in [Sym08] und [GCA+04] genauer beschrieben. Zur Erzeugung von Spam wird hoher technischer Aufwand betrieben [Bon04, Won05].

Auf der anderen Seite gibt es auch eine Vielzahl von Maßnahmen, die von Endnutzern, ISPs und E-Mail Dienstanbietern unternommen werden, um die Spamproblematik zumindest einzugrenzen. Zu diesen Maßnahmen gehören die Inhaltsanalyse [Apa08, CH06], die Drosselung der Mailauslieferung [Egg07] oder andere aktive Maßnahmen. [KWD+08c] gibt eine gute Übersicht über die unternommenen Anstrengungen.

Der IPclip-Mechanismus kann auch bei der Spambekämpfung einen wertvollen Beitrag leisten. Bei der Spambekämpfung können die Flags in der IPclip-Option als zusätzliche Information verwendet werden, um die Vertrauenswürdigkeit der E-Mail-Quelle zu bewerten. Wichtiger als die Erkennung von Spam ist für die Anwendung von IPclip die Möglichkeit der Rückverfolgung zur Quelle. Spammer verschleiern die wahre Herkunft der E-Mail, indem ihr Inhalt manipuliert wird. Ein Spammer hat jedoch nur begrenzten Einfluss auf den Inhalt der IPclip-Option. Er hat drei Möglichkeiten:

- Korrekte OI: Der Urheber der E-Mail kann als OI die korrekten Informationen angeben. Bei der Überprüfung ergibt für die Flags `user provided/trusted`. Anhand der OI kann er dann aber gefunden werden. Das ist keine Option für einen Spammer.
- Keine OI: Gibt der Urheber der E-Mail keine OI in seinen IP-Paketen an, setzt IPclip die eigen OI zusammen mit Knoten ID und Portnummer des IP-DSLAMs ein. Anhand dieser Informationen kann dann jedoch der Internetanschluss des Spammers sehr genau eingegrenzt werden.
- Falsche OI: Gibt der Urheber der E-Mail falsche OI an, werden diese durch die OI des Zugangsknotens ersetzt und die Flags werden auf `network provided/trusted` gesetzt. Anhand dieser Informationen kann dann jedoch sein Internetanschluss ebenfalls sehr genau eingegrenzt werden.

Verglichen mit dem generellen IP-Szenario ist im Falle von Spam auf eine Besonderheit hinzuweisen. Es besteht keine Ende-zu-Ende Beziehung zwischen dem E-Mail Versender und dem Empfänger. IP-Verbindungen terminieren an den jeweiligen Mailservern der Mailprovider. Dementsprechend muss die Information aus der IPclip-Option, damit sie den Mailempfänger auch erreicht, bewahrt werden. Dazu kann sie vom Mail Transfer Agent (MTA), der die erste IP-Strecke terminiert, aus der IPclip-Option übernommen und als zusätzliche Information in den SMTP-Header [Pos82] der E-Mail übernommen werden. Nur so ist es möglich, beim Empfänger die OI des Senders auszuwerten und adäquate Maßnahmen einzuleiten.

4.4 Kombination funktionaler Module

Die Komponentenbibliothek der vorgestellten Funktionalitäten zu der im Moment MAT, MPLS-UNI, TM und IPclip gehören, kann als Basis für die Entwicklung von leistungsfähigen und gleichzeitig variablen Paketprozessoren dienen, und somit die Vorteile von Hard- und Softwarelösungen miteinander vereinen.

In einer leistungsstarken Paketprozessorarchitektur muss es möglich sein, unterschiedliche funktionale Module effizient miteinander zu verbinden. Das heißt, die Module sollten durch die Art der Kombination und daraus folgender Intermodulkommunikation und Prioritäten nicht in ihrer Leistung eingeschränkt sein. Gleichzeitig sollten der Implementierungsaufwand und die zusätzlichen Kosten nicht zu groß ausfallen.

4.4.1 Pipelining am Beispiel von MATMUNI

Bei der Paketverarbeitung, vor allem im Teilnehmerzugangsbereich, sind die Gegebenheiten günstig, um eine verhältnismäßig effiziente Verbindung von Funktionalitäten zu ermöglichen. Viele Funktionen müssen im Datenpfad abgearbeitet werden. Das geschieht voneinander unabhängig. Alle Funktionen können im Datenpfad in Form einer Pipeline miteinander verbunden werden. Wegen der funktionalen Ähnlichkeiten der Module wurde im Rahmen der Untersuchungen prototypisch ein Paketprozessor entwickelt, der die Ausführungseinheiten von MAT, MPLS-UNI und TM miteinander kombiniert und somit alle drei Funktionalitäten bieten kann. Seine Bezeichnung ist MATMUNI [WKT+06]. In Abbildung 55 ist die Architektur der integrierten Lösung dargestellt. Sowohl in Up- als auch Downstream enthält das System alle Funktionalitäten von MAT, TM, und MPLS-UNI:

- Upstream (Kunde → ISP)
  - MAT: Ersetzung von CMAC durch PMAC
  - MPLS-UNI: Verkapselung jedes Frames mit MPLS Label Stack
  - TM: Farbmarkierung aller Datenframes mit drei Farben zur Verkehrsflusskontrolle im Kernnetz
- Downstream (ISP → Kunde)
  - MAT: Rückersetzung der PMAC durch die CMAC
  - MPLS-UNI: Entfernung des MPLS Label Stacks aus Frames und Wiederherstellung der originalen Framestruktur
  - TM: Funktionalitäten des Trafficmanagements für den Downstream sind nicht notwendig.

Die Funktionalitäten, die nahezu identisch in allen FMs implementiert sind, teilen sich die AEs global. Das sind FB, CA und das Speichersystem. Das Teilen der Module soll die Effizienz der Implementierung erhöhen. Die FBs zum Beispiel könnten für den Upstream bis zu 2160 Slices benötigen, würden sie einzeln implementiert. Durch die Integration sind maximal 720 Slices notwendig. Dabei wird die Vereinigungsmenge der einzelnen Schlüssel für die Funktionalitäten aus jedem Datenframe extrahiert und an das CA-Modul übertragen. Das bedeutet, dass die Verbindung der AEs nicht zu Leistungseinbußen des Systems führt. Bei entsprechender paralleler Speicherimplementierung kann das System die Speichersuchen für alle AEs parallel durchführen. Dann überträg der CA das Ergebnis der Suche an die AEs und der Datenframe traversiert nacheinander alle AEs, die ihre jeweilige Funktion ausführen. Unabhängige Speicherblöcke sind durch die Nutzung der internen BRAMs des FPGAs

realisierbar. Das stellt eine Möglichkeit der Speicherrealisierung dar, vorausgesetzt, es stehen genug BRAM Ressourcen zur Verfügung. Ist das nicht der Fall, muss externer Speicher, SRAM oder sogar DRAM, verwendet werden. Die Speicherentscheidung ist von großer Bedeutung für die Speicherkapazität, die Kosten und die Leistungsfähigkeit des Systems.

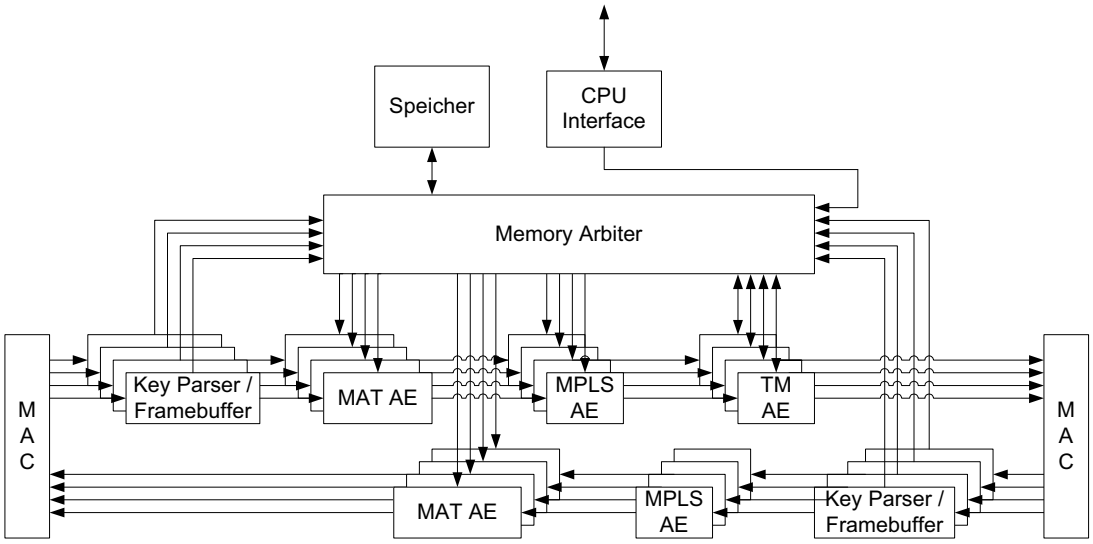


Abbildung 55 - Architektur von MATMUNI mit allen Funktionalitäten von MAT, TM und MPLS-UNI in Up- und Downstream.

Interner BRAM

In dem Fall, dass interner BRAM genutzt wird, ist es möglich, die Speicher für die einzelnen AEs voneinander zu separieren. Das hat den großen Vorteil, dass jeder Speicher mit einem eigenen Interface und einer eigenen Suchlogik parallel angesprochen werden kann. Dann entspricht die Suchperformance der von drei nicht integrierten unabhängigen AEs.

Externer RAM

Muss externer Speicher verwendet werden, besteht keine Möglichkeit der Parallelisierung von Speicheranfragen. Sowohl bei einer integrierten Implementierung im Rahmen von MATMUNI als auch bei Nutzung unabhängiger FMs müssen sich alle Elemente ein einzelnes Speicherinterface teilen. Die drei Schlüssel des Upstreams müssen seriell verarbeitet werden. Eine Möglichkeit die Speichersuche zu beschleunigen, wäre der Einsatz von Double Data Rate DRAM oder SRAM. Dennoch ist die Speicherperformance der limitierende Faktor eines solchen Systems.

Implementierung

Wie bereits vorher beschrieben, ist die Implementierung eines Prototypen auf einem FPGA von Xilinx (Virtex-4 XCVFX20) erfolgt [KWD+07]. Die notwendigen Ressourcen für die Umsetzung des Systems hängen von der Konfiguration von MATMUNI ab. Es bestehen verschiedene Möglichkeiten zur Optimierung des Ressourcenbedarfs der Konfiguration zur Synthesezeit:

- Auswahl der notewendigen AEs für eine Implementierung (MAT, TM, MPLS-UNI) in Up- Und Downstream
- Auswahl der Zusammensetzung der Schlüssel aller eingesetzten AEs.
- Auswahl der Anzahl notwendiger paralleler Datenpfade (1-4).

Diese Konfigurationen können durch die Änderung einiger Parameter in einer Konfigurationsdatei vorgenommen werden. Für die Prototypen wurde die vollständige Funktionalität für einen Datenpfad synthetisiert. Abhängig von der Zusammensetzung der Schlüssel benötigte MATMUNI zwischen 2300 und 4300 Slices (Tabelle 8). Dazu kommt etwas Zusatzlogik zur Synchronisation ankommender Daten. Wird MATMUNI mit einem typischen Schlüsselsetup und für einen Datenpfad konfiguriert, hat es eine Größe von 3300 Slices. Das gleiche Setup erfordert bei unabhängiger Implementierung der FMs 6000 Slices (MAT: 2100, MPLS-UNI 1500, TM 2400). Mit der integrierten Lösung können fast die Hälfte der Hardwareressourcen gespart werden. Es ist allerdings festzustellen, dass die Komplexität des CA stark zunimmt. Ein System mit mehr FMs erfordert Erweiterungen des CA, die einen großen Implementierungs- und auch wachsenden Hardwareaufwand zur Folge haben. Die bei der Vorstellung der FM-Architektur propagierte einfache Verbindung von FMs ist bei der MATMUNI-Architektur nicht ohne weiteres möglich. Ein weitgehender Ausbau dieser Architektur über zahlreiche weitere AEs im Datenpfad hinaus ist nicht zu empfehlen.

Tabelle 8 - Ressourcenbedarf von MATMUNI.

Modul	Slices	
	Min.	Max.
MAT	210	
MPLS	220	
TM	240	
Speicherinterface	282	1023
Controller & Arbiter	600	
Upstream FB	336	723
Downstream FB	336	723
Glue Logic	160	
MATMUNI	2300	4300

Das prototypisch implementierte System erreichte auch mit Maximalkonfiguration eine Geschwindigkeit von über 125 MHz, womit die Verarbeitung von GBit-Ethernet problemlos möglich ist.

Performance

Zur Evaluierung der Performance wurde MATMUNI mit MAT, TM, MPLS-UNI im Upstream und MPLS-UNI und MAT im Downstream für vier Datenpfade und damit einer Bandbreite von 4 GBit/s in jeder Richtung untersucht. Genutzt wurde wiederum eine ModelSim Simulation des erstellten VHDL-Codes. Die Ergebnisse können den Graphen in Abbildung 56 entnommen werden. Wie zu erkennen ist, ist die Verlustrate insbesondere bei sehr kleinen Framegrößen hoch. Erst mit Framegrößen über 256 Byte kommt es zu keinen Verlusten mehr. Damit ist MATMUNI, was die Leistungsfähigkeit betrifft, etwas schlechter als das MAT-Modul.

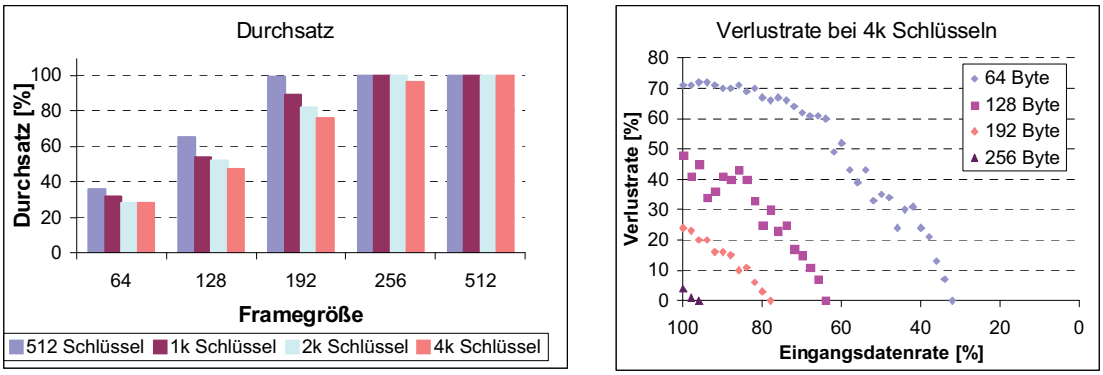


Abbildung 56 - Performance von MATMUNI in der Konfiguration, wie sie in Abbildung 55 dargestellt ist.

Das erklärt sich aus dem gestiegenen internen Kommunikations- und Steuerungsaufwand des Systems. Dieser tritt insbesondere im CA von MATMUNI zu Tage. Es bleibt dennoch zu konstatieren, dass typischer Datenverkehr mit seiner mittleren Framegröße von 416 Byte ohne Probleme durch MATMUNI zu bewältigen ist. MATMUNI kann also in einem Teilnehmerzugangssystem wie einem DSLAM oder GPON zum Einsatz kommen.

4.4.2 Blockorientierte Verarbeitung

Die bereits angedeutete Alternative zur Integration der FMs in ein System wie MATMUNI ist die Konstruktion des paketverarbeitenden Systems als Kombination der existenten FMs mit ihrer vollen Funktionalität. MATMUNI aus Abbildung 55 entspräche mit dieser Methode dem System, das in Abbildung 57 dargestellt ist. Alle FMs sind seriell miteinander verbunden. Als Ergebnis können auch hier alle Funktionalitäten auf ankommende Datenframes angewendet werden. Funktional besteht kein Unterschied zu MATMUNI.

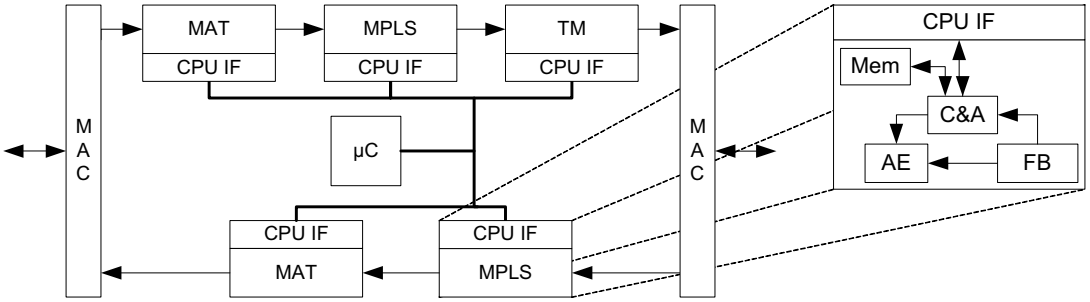


Abbildung 57 - Architekturvorschlag eines blockorientierten Paketprozessorsystems mit dem gleichen funktionalen Umfang wie MATMUNI.

Diese Methode der Implementierung hat den Vorteil, dass sie leistungsfähig aber vor allem sehr leicht zu implementieren und damit extrem flexibel ist. Es gibt architektonisch keine Begrenzung, die Anzahl der FMs betreffend. Die definierten Interfaces zwischen den FMs ermöglichen es, die Module miteinander zu kombinieren. Für den Slow-Path ist es möglich, einen Mikrocontroller oder eine CPU bereitzustellen, der/die mittels eines Bussystems an die CPU Interfaces aller FMs angeschlossen ist und die entsprechenden Aufgaben ausführen kann. Sollten zu viele FMs verwendet werden, sind auch mehrere CPUs verwendbar, um die Aufgaben des Slow-Path zu erfüllen. Es ist ohne weiteres denkbar, die Kombination von FMs aus einer Komponentenbibliothek zu automatisieren. Dazu wurden bereits einige Entwicklungen gemacht, die in [Krö06] dargestellt und ausgeführt wurden. Damit wird es dann sehr einfach möglich, paketverarbeitende Systeme, die genau bestimmen Leistungs- und Funktionsanforderungen genügen, zu erzeugen. Nachteilig an einer solch einfachen Kombination ist die Größe des ganzen Systems. Im Vergleich zu MATMUNI ist die reine Kombination der FMs doppelt so groß.

Performance

Durch den modularen Aufbau ist das blockorientierte System zwar fast doppelt so groß wie MATMUNI, es ist aber, was die Leistungsfähigkeit betrifft, leicht besser. Die in Simulationen ermittelte Leistungsfähigkeit des Systems ist in Abbildung 58 dargestellt. Es ist zu erkennen, dass die Unterschiede in der Leistungsfähigkeit zu MATMUNI nur sehr gering ausfallen. Im Gegensatz zu MATMUNI kommt es allerdings bei Framegrößen von 256 Byte nicht mehr zu Paketverlusten. Mit seiner Leistungsfähigkeit gleicht das System den Leistungsdaten des MAT-Moduls, welches auch das kritische FM im Datenpfad darstellt. Hier werden die Paketverwürfe verursacht. Die Architektur selbst erzeugt, wie zu erwarten, keine nennenswerten Reibungsverluste.

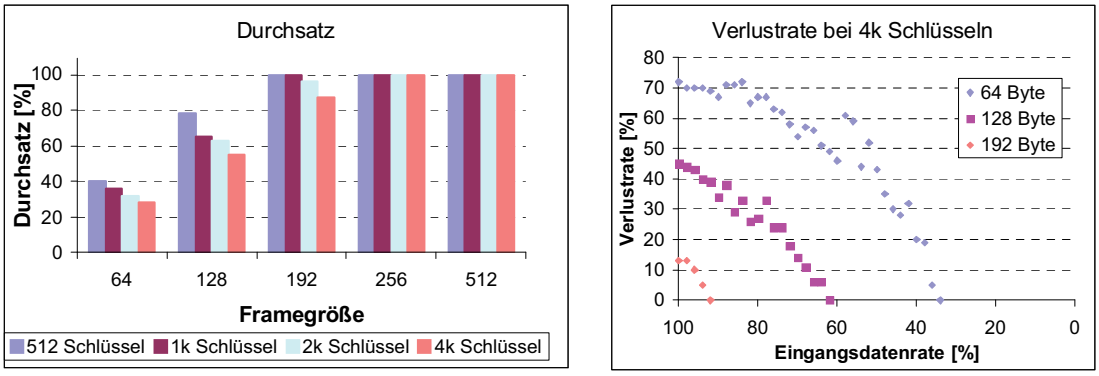


Abbildung 58 - Leistungsfähigkeit eines Systems, das in seiner Funktionalität MATMUNI entspricht, aber als einfache Kombination von FMs implementiert ist.

4.4.3 Networks-on-Chip

Eine weitere Möglichkeit ist die Integration eines paketverarbeitenden Systems, das ein Network-on-Chip (NoC) zur Modulkommunikation bereitstellt. Dabei sind Varianten unterschiedlicher Granularität denkbar. Zum einen könnten ganze FMs als Verarbeitungselemente an die einzelnen Router angeschlossen werden. Zum anderen könnten auch FB, AE, CA und Speicher jeweils als Verarbeitungselemente mit einzelnen Routern verbunden werden. Abbildung 59 stellt die Abbildung von MATMUNI auf ein NoC dar.

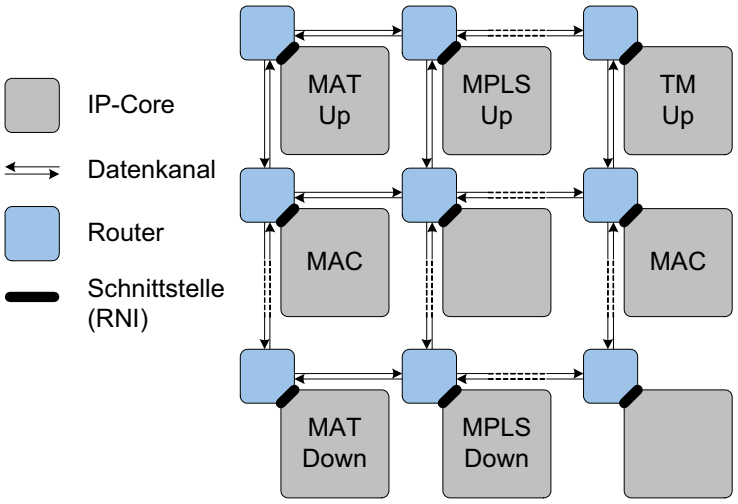


Abbildung 59 - Abbildung der Funktionalität von MATMUNI auf ein NoC (Abbildung aus [Kub08]).

Die Leistungsfähigkeit einer Abbildung von MATMUNI auf ein NoC ist jedoch nicht Inhalt dieser Arbeit. Es sei an dieser Stelle auf [Kub08] verwiesen, wo dieser Sachverhalt neben anderen gründlich untersucht und beleuchtet wird.

4.5 Zusammenfassung

In diesem Kapitel der Arbeit wurde eine Architektur vorgestellt, die es ermöglicht, Paketverarbeitung leistungsstark, hardwareeffizient und vor allem flexibel durchzuführen. Diese wurde in [WKT07] veröffentlicht. Auf Basis dieser Architektur wurden mehrere funktionale Elemente entwickelt und auf einem FPGA-Entwicklungsboard prototypisch implementiert. Diese richten sich in ihrem funktionalen Umfang an den Anforderungen aus, die in modernen TZNs heute herrschen. MAT [KWD+06], MPLS-UNI [WKD+06] und TM [WKD+06] sind in der Lage, die Funktionalität von DSLAMs und GPON-Systeme zu erweitern und zu flexibilisieren. Damit konnte gezeigt werden, dass die vorgestellte Architektur flexibel und leistungsfähig genug ist, um für verschiedene Aufgaben im Bereich der Paketverarbeitung genutzt zu werden.

Eine Erweiterung der den FMs zugrunde liegenden Architektur stellt IPclip [WKD+08, DKW+08a, DKW+08b] dar. IPclip ist ein Hardwaremodul, das in der Lage ist, „Trust-by-Wire“ in IP-basierte Datennetze einzuführen. Dadurch sind Anwendungen im Bereich der VoIP Notrufe [KWD+08a], der Verhinderung von Phishing [KWD+08b] und der Verbesserung des E-Mail-Dienstes [KWD+08c] möglich. Zum Einsatz kommt IPclip, wie auch die zuvor vorgestellten Module, im TZN.

Außerdem wurden einige Möglichkeiten der Kombination von FMs betrachtet. MATMUNI als integriertes System von FMs wurde prototypisch implementiert und evaluiert [WKT+06, KWD+07]. Alternativen der Kombination von FMs, wie die blockorientierte Verarbeitung oder die Nutzung von NoCs, wurden ebenfalls aufgezeigt. Dabei wurde deutlich, dass die Flexibilität und die Performance von MAMTUNI im Vergleich begrenzt sind, die Hardwarekosten jedoch weitaus geringer ausfallen als das bei der blockorientierten Verarbeitung der Fall ist. Der Autor favorisiert deshalb, ausreichende Hardwareressourcen vorausgesetzt, die blockorientierte Verarbeitung.

5 Evolvierbare Paketklassifizierung in Hardware

Ein leistungsstarker Paketklassifizierer bildet eine entscheidende Kernkomponente zur Entwicklung leistungsfähiger Paketprozessoren. Wie in Abschnitt 2.1 verdeutlicht wurde, basiert die Verarbeitung von Paketen in verschiedenen funktionalen Elementen eines Paketprozessors darauf, jedes Paket dem korrekten Flow zuzuordnen. Das bedeutet nichts anderes, als dass für jedes Paket geklärt werden muss, was mit ihm zu geschehen hat. Diese Aufgabe, die Klassifizierung von Paketen, übernimmt der Paketklassifizierer. In Kapitel 3.2 wurde der aktuelle Stand der Technik im Bereich Paketklassifizierung insbesondere mit Hashfunktionen dargelegt. Dieses Kapitel befasst sich nun mit der Entwicklung eines einfachen Paketklassifizierers auf Basis einer evolvierbaren Hashfunktion und dessen Umsetzung in eine funktionsfähige Hardwarearchitektur. Ziel der Entwicklung ist das Finden einer sehr guten Hashfunktion, die in verschiedenen Systemen zur Paketklassifizierung zum Einsatz kommen kann, um diese zu optimieren.

5.1 Ein evolvierbarer Hash-basierter Paketklassifizierer

Der hier vorgestellte evolvierbare Paketklassifizierer (Evolvable Packet Classifier - EPC) basiert auf adaptiven Hashfunktionen. Der jeweils vorliegende Hashalgorithmus passt sich den aktuellen Gegebenheiten der zu klassifizierenden Daten mittels eines genetischen Algorithmus (GA) direkt in Hardware an. Da sowohl der Prozess der Evolution des GA in Hardware ausgeführt als auch die Klassifizierung selbst in Hardware implementiert wird, handelt es sich um eine vollständig intrinsische, evolvierbare Hardware. Die Autoren von [TH99] prägen für ein solches System, das sowohl die Zielarchitektur als auch den GA auf einer Hardwareplattform vereint, den Begriff Complete Hardware Evolution.

5.1.1 Gesamtsystem des evolvierbaren Paketklassifizierers

Das Klassifizierersystem ist eine reine Hardwarelösung. Die Zielarchitektur, auf deren Grundlage die Hardwareentwicklung vorgenommen wurde, ist ein FPGA der Firma Xilinx. Der verwendete Virtex-4FX20 ist der Kernbaustein auf dem zur Evaluierung des Prototyps zur Verfügung stehenden ML405 Entwicklungboard.

Wie in [Wil01] dargestellt, können paketverarbeitende Systeme im Allgemeinen in einen Daten- und Kontrollpfad (Fast-Path, Slow-Path [Awe99]) getrennt werden. Das gilt auch für den EPC. Im Datenpfad findet die Klassifizierung der Pakete statt. Funktional entspricht der Ablauf dem in Abbildung 60 dargestellten Ablaufdiagramm. Aus jedem ankommenden Paket ist ein Schlüssel zu extrahieren. Anhand dieses Schlüssels wird die Klassifizierung vorgenommen. Um zum Schlüssel die korrekte Regel (die zugehörige Nutzinformation, Zuordnung zu einem Flow, Next Hop Information) effizient zu ermitteln, hasht das System den Schlüssel und beginnt

mit der so ermittelten Speicheradresse die Suche nach den Daten. Die Suche erfolgt mittels linearer Kollisionsauflösung. Nach Ermittlung der Regel erscheint diese zusammen mit dem Frame am Ausgang des Klassifizierers. Sie kann dann in einem funktionalen Modul weiter verarbeitet werden. Die Leistungsfähigkeit des EPCs hängt entscheidend von der Qualität der Hashfunktion ab. Ist sie perfekt, dann wird jeder mögliche Schlüssel auf eine andere Speicheradresse abgebildet. Somit kann mit jedem Speicherzugriff die passende Regel zu einem Schlüssel gefunden werden.

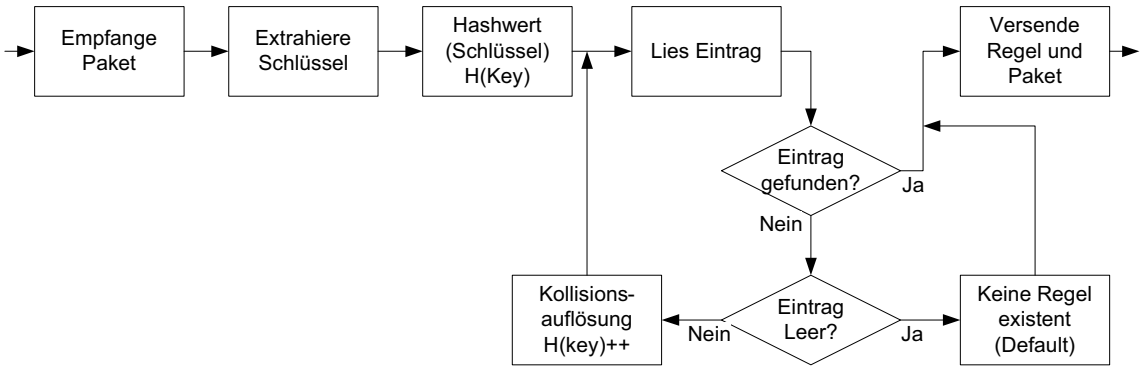


Abbildung 60 - Datenpfadoperation.

Im Kontrollpfad wird das Ziel verfolgt, eine möglichst leistungsfähige, im optimalen Fall eine perfekte Hashfunktion zu ermitteln. Der Kontrollpfad hat primär die Aufgabe, die Hashfunktion und damit den gesamten EPC zu optimieren. Dies geschieht mittels eines GA. Es handelt sich um einen in Hardware implementierten evolutionären Prozess. Zur Implementierung des in Abbildung 60 dargestellten Verlaufs ist eine Hardwarearchitektur des Datenpfades gewählt worden, wie sie in Abbildung 61 zu sehen und in den folgenden Unterabschnitten beschrieben ist.

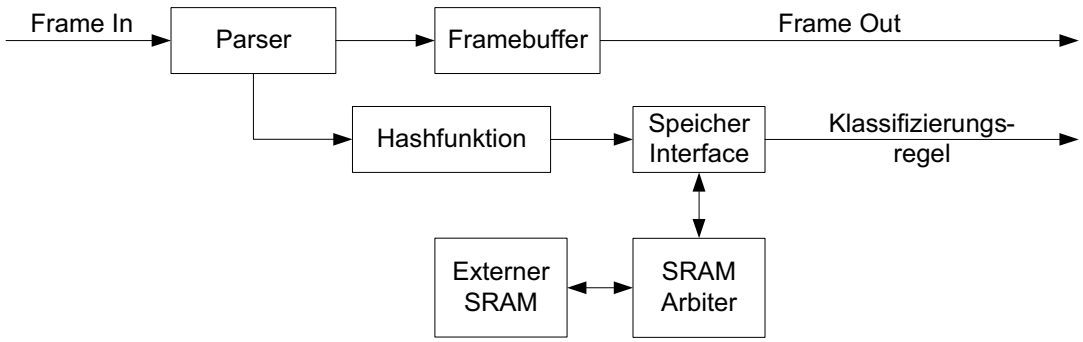


Abbildung 61 - Struktur des Datenpfades des EPCs.

5.1.2 Datenpfad

Jedes Paket durchläuft die einzelnen Module entlang des Pfades. Die Aufgaben der einzelnen Funktionsblöcke werden im Folgenden erläutert.

5.1.2.1 Parser und Framebuffer

Die Pakete erreichen zuerst den Parser. Dieser extrahiert den Schlüssel aus jedem Paket und übergibt ihn dann der Hashfunktion. Der Aufbau des Parsers weicht von dem in Abschnitt 4.2.1 für FMs vorgestellten ab, denn welche Teile der ankommenden Pakete als Schlüssel extrahiert werden, ist zur Laufzeit des Systems konfigurierbar. Eine Bitmaske definiert, ob ein spezielles Bit des Pakets Teil des Schlüssels ist. Zur Konfiguration der Bitmaske dient ein BRAM, der 2048 Byte abspeichern kann. Damit enthält der Speicher für jedes einzelne Bit eines Pakets die Information, ob es Teil des Schlüssels ist, da die MTU von IP-Paketen 1500 Byte nicht überschreitet. Hat der Speichereintrag zu einem Bit den Wert '1', so ist es als Teil des Schlüssels zu verwenden. Eine Einschränkung bei der Konfigurierbarkeit des Parsers existiert: Die Größe des Schlüssels, die Anzahl der Bits, die er umfasst, ist im VHDL-Code des Parsers zu definieren. Für eine Änderung muss er demzufolge neu synthetisiert werden. Es ist eine freie Konfigurierbarkeit zur Synthesezeit, nicht zur Laufzeit, gegeben. Der Parser speichert die extrahierten Schlüssel zwischen und übergibt sie an das Hashfunktionsmodul zur weiteren Verarbeitung. Der Datenframe selbst wird einem Framebuffer übergeben und dort gespeichert, bis die Klassifizierung über den Schlüssel erfolgt ist.

5.1.2.2 Hashfunktion

Die Hashfunktion als Kernelement des EPCs verarbeitet die ankommenden Schlüssel. Sie hasht die Schlüssel und übergibt sie dann zusammen mit dem ermittelten Hashwert an die nachfolgenden Elemente. Sie ist zur Synthesezeit konfigurierbar. Auch der innere Aufbau, die Breite der Eingangsdaten und die Breite der Hashwerte sind konfigurierbar. Die genauen Architekturen und Funktionsweisen verschiedener, im Rahmen dieser Arbeit vom Autor entwickelter, evolvierbarer Hashfunktionen sind in Kapitel 5.2 ausführlich beschrieben. Die Breite der Eingangsdaten entspricht der Breite der im Parser extrahierten Schlüssel. Die Breite der Hashwerte hängt von der Größe der Schlüsselmenge S der im EPC nutzbaren Schlüssel ab. Jeder Hashwert adressiert einen Eintrag in einem Schlüsselspeicher.

Die Qualität einer Hashfunktion, bzw. die Anzahl der erzeugten Kollisionen hängt unter anderem vom Belegungsfaktor des Speichers ab [Knu05]. Eigene Untersuchungen ergaben, dass ein Belegungsfaktor von $\alpha = 0{,}5$ für den Hashspeicher gewählt werden sollte, um die Anzahl der Kollisionen gering zu halten. Damit ergibt sich für die Ausgangsbitbreite M der Hashfunktion:

$$M = \log_2\left(\frac{|S|}{\alpha}\right) \tag{14}$$

Um 32.768 IP-Adressen als Schlüssel zu speichern, ist demzufolge eine Hashfunktion notwendig, die 32 Bit breite Werte auf 16 Bit breite Werte abbildet. Die so ermittelte Speicheradresse übernimmt dann ein Speicherinterface zusammen mit dem Schlüssel, um die notwendige Suche nach der passenden Regel im Speicher durchzuführen.

5.1.2.3 Speicherinterface

Das Speicherinterface übernimmt Schlüssel und Hashwert und übergibt den Hashwert als Suchadresse an ein Speicherelement. Jedes Datum hat die gleiche Struktur. Ein Eintrag besteht aus der Klassifizierungsregel generischer Länge, dem Schlüssel generischer Länge und zwei Bit, die die Gültigkeit des Speichereintrags definieren. Ein Bit (existent – E) gibt an, ob ein Eintrag an der Stelle existiert. Das zweite Bit (valid – V) gibt an, ob der Eintrag gültig ist, also zur Klassifizierung verwendet werden kann. Diese Datenstruktur ist auf die jeweilig existierende Speicherhardware abzubilden (Abbildung 62).

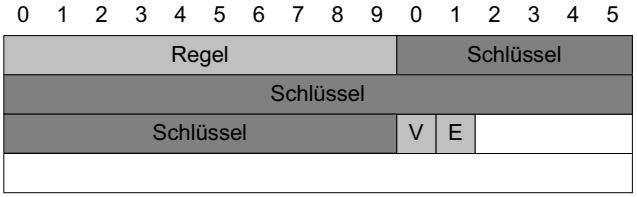


Abbildung 62 - Abbildung der Struktur eines Speichereintrags auf einen 16 Bit breiten Speicher. Der Schlüssel ist in dem Beispiel 32 Bit breit, die Klassifizierungsregel hat 10 Bit. Damit sind vier[20] Speicheradressen (64 Bit) zu verwenden.

Es sind drei Zugriffsarten auf den Speicher zu unterscheiden. Schreibzugriffe, Löschzugriffe und Suchzugriffe.

Suchzugriffe

Das Datum, das an der Stelle gelesen wird, welches durch den Hashwert definiert ist, muss auf Gültigkeit überprüft werden. Dabei können drei Fälle unterschieden werden

- An der Stelle existiert ein Eintrag und er ist gültig. In diesem Fall wird der Schlüssel s im Speicher mit dem vorliegenden Schlüssel verglichen. Bei Gleichheit ist die passende Regel gefunden und die Klassifizierungsregel verlässt zusammen mit dem Datenpaket aus dem Framebuffer den Datenpfad, um von nachfolgenden Modulen verarbeitet werden zu können. Bei Ungleichheit der beiden Schlüssel ist eine Kollision aufgetreten. Der EPC löst diese durch eine lineare Kollisionsauflösung auf. Der Hashwert wird

[20] Die Größe des Eintrags erfordert nur drei Speicheradressen. Um jedoch eine einfache Abbildung der logischen Adresse eines Eintrags auf die physikalischen Adressen im Speicher zu ermöglichen, sind für jeden Eintrag vier Adressen vorzusehen.

inkrementiert und eine neue Speicherabfrage wird an der Adresse $h(s)+1$ durchgeführt.

- An der Stelle existiert ein Eintrag. Er ist aber ungültig. Dies kommt vor, wenn ein Eintrag im Speicher gelöscht wurde. In diesem Fall wird der Hashwert ebenfalls inkrementiert und eine neue Speicherabfrage an der Adresse $h(s)+1$ durchgeführt.
- An der Stelle existiert kein Eintrag. Wenn kein Eintrag im Speicher existiert, so war die Suche erfolglos. Der gesuchte Schlüssel kommt nicht im Speicher vor. Es kann keine Regel gefunden werden.

Schreibzugriffe

Soll ein neuer Speichereintrag vorgenommen werden, muss der Speicher solange durchsucht werden, bis einer der folgenden Fälle eintritt:

- Es existiert ein Eintrag, er ist gültig und der Schlüssel entspricht dem des zu schreibenden Eintrags. Der gefundene Eintrag wird vom aktuellen überschrieben (Update).
- Es existiert ein Eintrag und er ist ungültig. Ist der Eintrag der erste ungültige muss sich die Adresse gemerkt werden. In jedem Fall wird die Suche fortgesetzt.
- An der Stelle existiert kein Eintrag und im Verlauf der Suche wurde ein ungültiger Eintrag gefunden. Der neue Wert überschreibt den gefundenen ungültigen Eintrag.
- An der Stelle existiert kein Eintrag und im Verlauf der Suche wurde kein ungültiger Eintrag gefunden. Der neue Wert wird an die freie Stelle geschrieben.

Löschzugriffe

Um die Konsistenz des Speichers nicht zu gefährden, kann beim Löschen nicht einfach ein Eintrag gesucht und dann die Speicherstelle frei gegeben werden. Der Grund hierfür ist die verwendete lineare Kollisionsauflösung. Ein Beispiel. Drei Einträge haben den gleichen Hashwert von 5. Sie sind dann an den Adressen 5, 6 und 7 abgelegt. Soll nun der Eintrag an Adresse 6 gelöscht werden, kann die Speicherstelle nicht frei gegeben werden, da sonst der Eintrag an Adresse 7 nicht mehr auffindbar wäre. Dessen Suche beginnt bei Adresse 5 und würde im Falle einer Löschung bei Adresse 6 erfolglos enden. Deshalb müssen Einträge, die gelöscht werden sollen, immer als existent, jedoch ungültig markiert werden. Mit diesem Mittel ist die Suche, wie oben beschrieben, erfolgreich und jeder gültige Eintrag wird gefunden.

5.1.2.4 Speicherimplementierung

Bei der Implementierung des Speichers für die Schlüsseldaten sind die Randbedingungen zu beachten, die die Zielarchitektur vorgibt. Aus diesen ergeben sich zwei mögliche Implementierungsvarianten. Die Implementierung aus internen BRAM-Elementen und die Nutzung eines externen statischen RAMs (SRAMs). Die Nutzung des vorhandenen DDR-

SDRAM ist trotz dessen großer Kapazität von 64 MB ausgeschlossen, da die Zugriffslatenz mindestens 29 Taktzyklen bei wahlfreiem Speicherzugriff beträgt [Dan06].

Implementierung aus BRAM

Die Nutzung des FPGA-eigenen BRAMs zur Implementierung stellt die leistungsfähigste Variante dar. Die BRAMs, welche frei konfigurierbar aus hart verdrahteten Zellen mit jeweils 18 KBit Kapazität erzeugt werden können, besitzen vier wichtige Eigenschaften:

- Durch die freie Kombinierbarkeit kann ein Speicher erzeugt werden, dessen Breite exakt der notwendigen Breite für einen Speichereintrag entspricht. Es ist keine Anpassung der Einträge an die physikalischen Gegebenheiten eines Speichermoduls notwendig. Damit ist ein Speichereintrag mit einem Speicherzugriff les- bzw. schreibbar.
- BRAMs können mit dem vollen Systemtakt des EPC auf dem FPGA betrieben werden.
- BRAMs haben eine Lese- und Schreiblatenz von lediglich einem Takt.
- BRAMs können als echte Dualport-RAMs betrieben werden, was den Speicherdurchsatz erhöht.

Daraus kann man folgern, dass nichts gegen eine Implementierung des Speichers aus BRAM spricht. BRAMs sind jedoch direkt auf dem Chip des FPGAs implementiert und stehen nur in sehr begrenzter Zahl zur Verfügung. Das für den Prototyp verwendete Virtex-4FX 20 FPGA besitzt insgesamt 68 Blöcke mit einer Kapazität von je 18 KBit. Damit ergibt sich ein Gesamtspeicher von nur 154 KByte. Das ist zu gering für die Implementierung des Speichers. Sogar das größte kommerziell verfügbare FPGA der Virtex-4 Serie hat nur wenig mehr als 1 MByte Speicherkapazität [Xil04b]. Einige der BRAMS werden von unterschiedlichen Modulen der EPCs benötigt (z. B. vom Framebuffer) was die Menge des effektiv zur Verfügung stehenden Speichers verringert. Aus diesem Grund kommt eine Implementierung des Speichers mit BRAM nur für sehr kleine Schlüsselmengen in Frage.

Implementierung aus SRAM

Für größere Schlüsselmengen müssen externe Speicherbausteine Verwendung finden. In Frage kommen nur SRAMs. SRAMs haben eine Latenz von nur wenigen Takten, weshalb die Wahl beim Aufbau des Prototypsystems auf den verbauten SRAM fiel. Er steht mit einer Kapazität von $32 \cdot 256$ KBit auf dem Board zur Verfügung. Bei Speichereinträgen, deren Größe 32 Bit übersteigt, sind immer zwei Speicherstellen für jeden Eintrag notwendig. Das ist bei dem entwickelten und untersuchten System der Fall. Jeder Eintrag enthält neben dem 32 Bit breiten Schlüssel eine Regel variabler Länge und die zwei Kontrollbits. Damit ergibt sich eine maximale Kapazität des SRAMs von $128k = 131.072$ Einträgen.

5.1.3 Kontrollpfad

Der Kontrollpfad des Paketklassifizierers umfasst den gesamten genetischen Algorithmus (GA). Er ist vollständig in Hardware implementiert (Abbildung 63). Es handelt sich hier um eine evolvierbare Hardware. Es stellt sich die Frage, ob eine Hardwareimplementierung des GA notwendig ist. Viele paketverarbeitende Systeme haben einen in Software implementierten Kontrollpfad, da im Allgemeinen sowohl die zu verarbeitenden Datenmengen, als auch die an die Verarbeitung gestellten Zeitanforderungen gering sind. Das ist hier aus zwei Gründen nicht der Fall. Die Leistungsfähigkeit der sich evolvierenden Hashfunktion verbessert sich ähnlich einer Wurzelfunktion (siehe Abbildung 21). Das heißt, zu Beginn der Entwicklung ist sie verhältnismäßig schlecht. Der erreichte Durchsatz ist sehr unbefriedigend. Es ist deshalb überaus wichtig, schnell einen Zustand „guter“ Leistungsfähigkeit zu erreichen. In vorangegangenen Untersuchungen mit einer Softwareimplementierung des GA hat sich gezeigt, dass eine solche Implementierung überaus rechenintensiv ist. Sogar die hier vorgestellte Hardwareimplementierung benötigt einen nicht unerheblichen Zeitraum zur Berechnung einer hinreichend großen Anzahl von Generationen im GA. Dies verdeutlichen Untersuchungen, auf die im weiteren Verlauf der Arbeit ausführlich eingegangen wird.

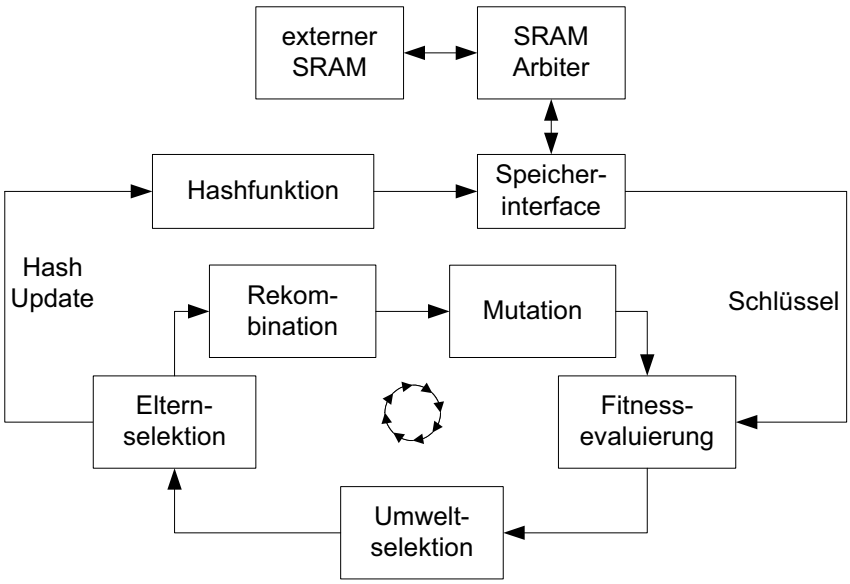


Abbildung 63 - Architektur des Kontrollpfades des EPC.

5.1.4 Verschränkung von Daten- und Kontrollpfad

Um ein funktionierendes EPC-System zu erhalten, müssen der Daten- und der Kontrollpfad miteinander verbunden werden. Das beste Genom der Hashfunktion, die der GA im Kontrollpfad ermittelt, muss im Datenpfad konfiguriert werden, damit der EPC die Paketklassifizierung optimal durchführen kann. Ein neues Genom bedeutet stets, dass eine

vollkommen veränderte Hashfunktion Verwendung findet. Folglich ist ein totales Rehashing aller Einträge im Speicher notwendig. Das kann nicht auf einem einzelnen Speicher durchgeführt werden, wenn die Datenpfadoperationen nicht unterbrochen werden sollen. Ein verbundenes System enthält aus diesem Grund zwei parallele Pfade aus Hashfunktion, Speicherinterface und Speicher im Datenpfad (Abbildung 64). Diese sind miteinander verschränkt.

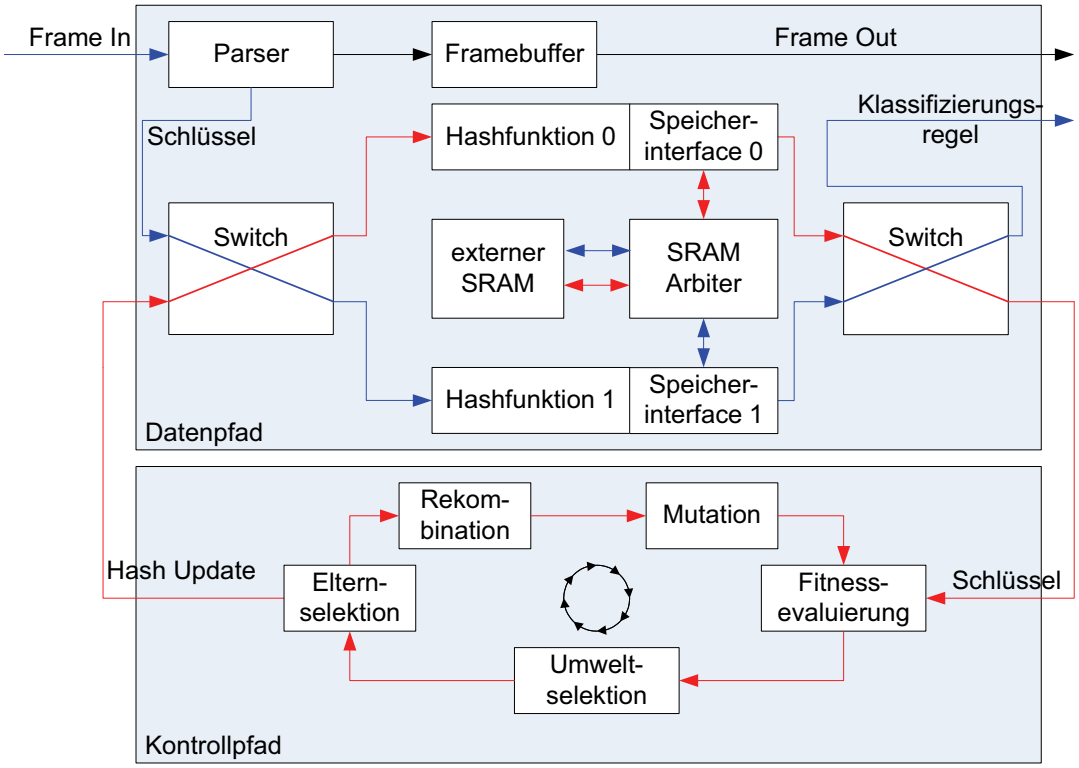


Abbildung 64 - Aufbau des Gesamtsystems des evolvierbaren Paketklassifizierers mit implementiertem Speicher im externen SRAM. Die parallelen Pfade im Kontrollpfad müssen sich den Speicherzugriff über einen Arbiter teilen.

In einem der beiden Pfade laufen die normalen Datenpfadoperationen ab. Das ist in Abbildung 64 blau dargestellt. Den anderen Pfad verwendet der GA des Kontrollpfades um die gesamten Schlüssel aus dem Speicher auszulesen. Diese werden zur Berechnung der Fitness der Lösungskandidaten im GA benötigt. Das ist in der Abbildung rot dargestellt.

Kommt es zu einem Update der Hashfunktion im Datenpfad, rekonfiguriert das System die Hashfunktion, die vom Kontrollpfad verwendet wurde. Das neue Genom verändert die Hashfunktion. Ein vollständiger Rehash im zugehörigen Speicher ist notwendig. Dazu wird er gelöscht. Danach werden alle gespeicherten Schlüssel aus dem anderen Speicher, den der Datenpfad nutzt, nacheinander ausgelesen. Diese Schlüssel durchlaufen die neue Hashfunktion und werden im Speicher abgelegt. Das Auslesen aus dem Datenpfad erfolgt mit niedrigerer Priorität als normale Datenpfadoperationen. Dadurch ist gewährleistet, dass die eigentliche

Paketklassifizierung nicht beeinträchtigt wird. Nachteil der niedrigen Priorität ist allerdings, dass die Dauer der Rekonfiguration maßgeblich vom Datenaufkommen im Datenpfad beeinflusst wird. Bei hohem Datenaufkommen und einem Betrieb an der Kapazitätsgrenze des Speichers kann die Rekonfiguration unter Umständen sehr lange dauern. In der Zeit, in der das System rekonfiguriert wird, ruht der genetische Algorithmus im Kontrollpfad. Er wird erst nach dem Abschluss des Rehashings fortgesetzt.

Die Abbildung des Gesamtsystems weist nur einen Speicher auf, den externen SRAM. Da als Basis zur Implementierung nur ein Entwicklungsboard mit einem verbauten SRAM zur Verfügung stand, wurden die beiden benötigten Speicher des Datenpfades logisch getrennt physisch auf einem SRAM implementiert. Wie bereits oben erwähnt, fasst der SRAM 128k Einträge. Es stehen also lediglich 64k Einträge in jedem logischen Speicher zur Verfügung. Der Belegungsfaktor ist auf $\alpha = 0{,}5$ begrenzt. Insgesamt können im implementierten EPC deshalb maximal 32k Schlüssel gespeichert werden. Diese Anzahl von Schlüsseln stellt demzufolge auch die Basis dar, auf Grund derer die Untersuchungen durchgeführt wurden, die im Rahmen dieser Arbeit erläutert und dargestellt sind.

5.2 Die evolvierbare Hashfunktion

Soll eine Hashfunktion für die spezielle Aufgabe der Paketklassifizierung in einer Hardware eingesetzt werden, muss sie mehrere Anforderungen erfüllen:

- Die Qualität des Hashings muss für unterschiedlich zusammengesetzte, verschieden große und über die Zeit veränderliche Datenbasen hoch sein.
- Um sowohl Datendurchsatz als auch Latenz des EPC zu optimieren, muss jedes Eingangsdatum mit hoher Geschwindigkeit gehasht werden, möglichst rein kombinatorisch, das bedeutet innerhalb eines Taktzyklus des Klassifizierers.
- Die Kosten für eine Implementierung müssen moderat ausfallen. Das heißt, das Verhältnis von Leistungsfähigkeit zu Hardwarekosten muss die Nutzung einer solchen Hashfunktion rechtfertigen.
- Die Hashfunktion muss generisch sein. Sie muss verschiedene Bitbreiten für den zu hashenden Wert und das Hashergebnis unterstützen. Damit ist eine Anwendbarkeit auf verschiedenen Problemstellungen gegeben.

Um die aufgeführten Kriterien zu erfüllen, verbieten sich von vornherein Hardware-implementierungen der sehr leistungsstarken Hashalgorithmen MD5 und SHA1. Wie aus den Ausführungen in Kapitel 3.2.3.1 hervorgeht, ist die Berechung eines Hashwertes mit diesen Funktionen sehr aufwendig. Hardwareimplementierungen für FPGAs sind entweder überaus kostspielig, was die notwendigen Ressourcen betrifft, oder sehr langsam in ihrer Ausführung [DHV01].

Um die Möglichkeiten der Hardwareevolution in eine Hashfunktion zu integrieren, muss deren Funktion durch einen Vektor von Bits, ein Genom, steuerbar sein. In der Literatur wurde

eine optimierbare Architektur vorgestellt, welche auf die besonderen architektonischen Eigenschaften von FPGAs Rücksicht nimmt. Damiani et. al. stellen in [DLT98, DLT99] und [DT99] eine evolvierbare Hashfunktion auf Basis von 4-Eingangs Look-Up-Tables (LUTs) vor. Diese Architektur ist an die Gegebenheiten der in FPGAs existierenden Logikelemente (Configurable Logic Blocks) angepasst.

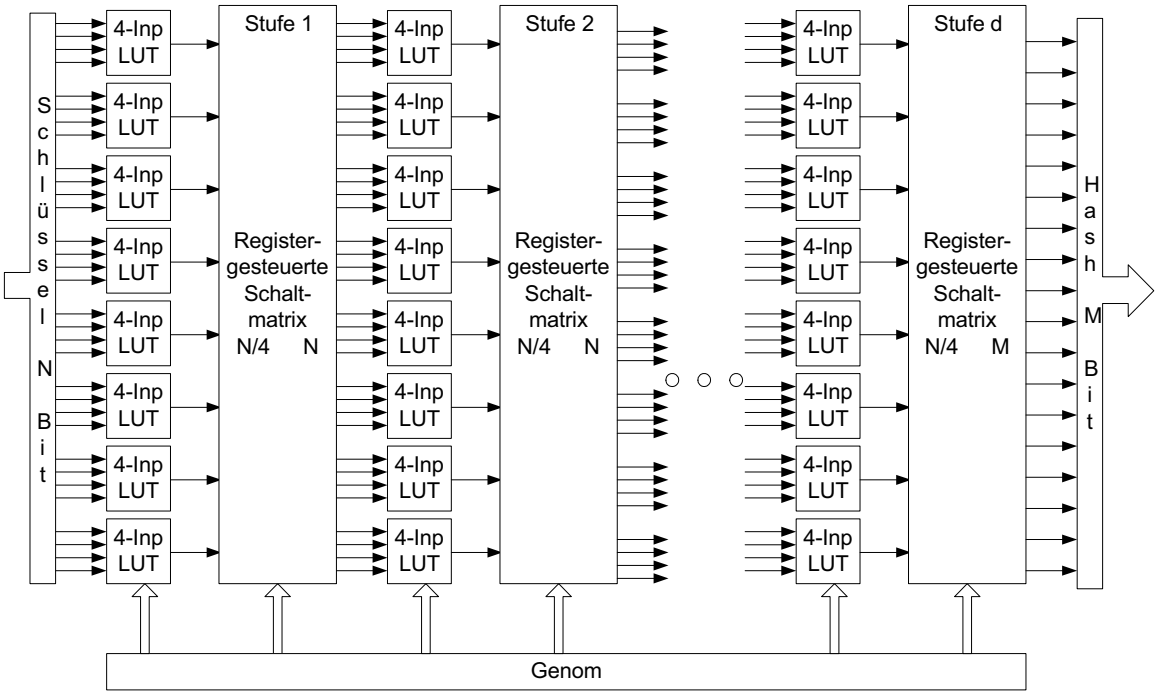


Abbildung 65 - Architektur einer Hashfunktion, die aus miteinander verknüpften 4-Eingangs-LUTs besteht. Die Anzahl der Stufen ist variabel.

Jede beliebige 4-Eingangs LUT kann mit $2^4 = 16$ Bit vollständig programmiert werden. Damit können *N* Bit breite Schlüssel auf *M* Bit gehasht werden. Jede reguläre Stufe besteht aus $N/4$ LUTs. Im Gegensatz zur Literaturquelle ist zur Steigerung der Flexibilität zwischen den einzelnen Stufen eine programmierbare Schaltmatrix angeordnet. Die Anzahl der Stufen *d* kann frei konfiguriert werden. Die Matrix enthält *N* Multiplexer. Jeder bildet $N/4$ Eingänge auf einen Ausgang ab. Dadurch ist es möglich, jeden Eingang der Folgestufe mit jedem Ausgang der aktuellen Stufe zu verbinden. Es ergibt sich für eine reguläre Stufe ein Genom der Größe l_s:

$$l_s = N \cdot \log_2\left(\frac{N}{4}\right) + \frac{N}{4} \cdot 16 \qquad (15)$$

Für die letzte Stufe muss eine Schaltmatrix aus *M* Multiplexern vorgesehen werden, um zu gewährleisten, dass die Hashfunktion *M* Ausgänge aufweist. Das Gesamtgenom hat die Länge:

$$l = (d-1)\cdot\left[N\cdot\left(\log_2\left[\frac{N}{4}\right]\right)\right] + M\cdot\log_2\left(\frac{N}{4}\right) + d4N \qquad (16)$$

Bereits in ersten Simulationen zeigte sich, dass die Leistungsfähigkeit der Architektur nicht zufriedenstellend ist. Insbesondere skaliert die Hashfunktion mit zunehmender Größe der Schlüsselmenge sehr schlecht. Das heißt, bei größeren Schlüsselmengen, wobei an dieser Stelle bereits bei $|S| = 8.192$ Schlüsseln von größer gesprochen werden muss, entsteht bereits eine sehr große Zahl von Kollisionen. Dieser an die Literatur angelehnte Ansatz ist nicht weiter zu verfolgen. Darum müssen andere Architekturen, die eine bessere Gesamtleistung und vor allem bessere Skalierungseigenschaften zeigen, gefunden werden. Dazu wurden vom Autor verschiedene Architekturen entwickelt, deren Aufbau, Funktionsweise und Eigenschaften im Folgenden untersucht und erläutert werden soll.

5.2.1 Zwei Multiplexer mit XOR-Verknüpfung

Evolvable Hardware ermöglicht die Optimierung von XOR-basierten Hashfunktionen. Bei diesen ist das Problem, die richtige Auswahl der Operanden für die XOR-Gatter vorzunehmen. Die Auswahl der Bits des Schlüssels als Operanden kann durch einen GA ermittelt werden. Davon abgeleitet wurde vom Autor eine Architektur entwickelt, die zwei Multiplexer mit einer XOR-Verknüpfung verwendet. Bei der 2-Mux-XOR-Architektur ist die Hashfunktion wie in Abbildung 66 dargestellt aufgebaut. Für jedes der *M* Bits des Hashwertes ist ein Funktionselement eingesetzt. Jedes der *M* Funktionselemente besteht aus zwei Multiplexern, die eine $N \mapsto 1$ Abbildung durchführen und abschließend mit einem XOR-Gatter verknüpft sind. *N* ist die Bitbreite der zu hashenden Schlüssel. Die Funktion jedes einzelnen Multiplexers ist durch einen Steuereingang definiert. Durch Änderung der Steuereingänge der Multiplexer kann somit die Funktionalität der Hashfunktion verändert werden.

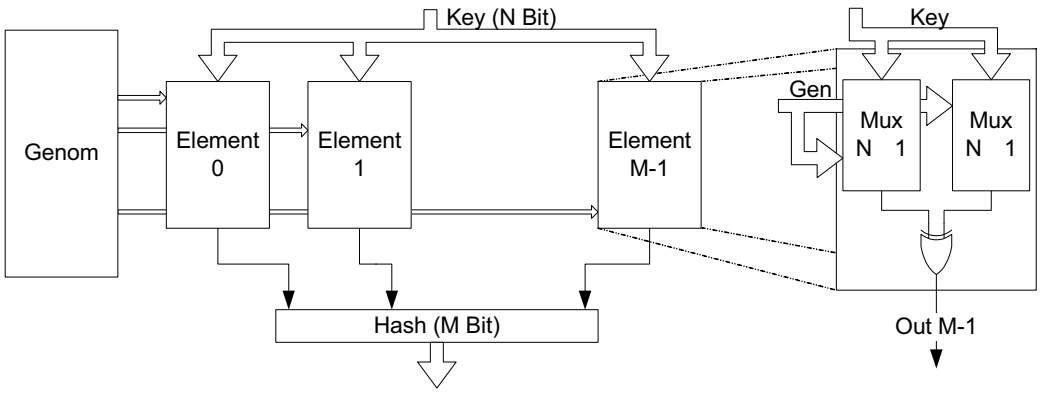


Abbildung 66 - Hashfunktion mit XOR-verknüpften Multiplexern (2Mux-XOR).

Die Gesamtheit aller Steuereingänge bildet das Genom. Um eine effiziente Hardwareimplementierung insbesondere der Multiplexer zu ermöglichen, sollte N stets eine Zweierpotenz sein. Für die Hashfunktion ergibt sich dann eine Genomgröße von:

$$l = M \cdot 2 \cdot \lceil \log_2(N) \rceil \tag{17}$$

Damit sind 2^l verschiedene Hashfunktionen umsetzbar.

5.2.2 Zwei Multiplexer in zwei Stufen XOR-Verknüpft

Um den Suchraum für eine leistungsfähige Hashfunktion zu vergrößern, wurde vom Autor eine weitere Architektur, bestehend aus zwei Stufen von Multiplexern, entwickelt. Diese entspricht in ihrer Struktur der Darstellung in Abbildung 67. Hierbei sind in einer ersten Stufe N Funktionselemente angeordnet, die in ihrem Aufbau dem im vorigen Abschnitt beschriebenen gleichen. In einer zweiten Stufe sind nun allerdings M weitere Multiplexer angeordnet, welche ebenfalls über Steuereingänge verfügen, um die N Ausgänge der ersten Stufe auf den Ausgang der Hashfunktion abzubilden. Hierbei ergibt sich nun ein Genom der Größe:

$$l = (N \cdot 2 + M) \cdot \lceil \log_2(N) \rceil \tag{18}$$

Dadurch sind weitaus mehr unterschiedliche Hashfunktionen umsetzbar, als dies für die vorherige Version der Fall ist. Für ein Beispiel mit nur $N = 16$ Bit breiten Schlüsseln und einer Ausgangsdatenbreite von $M = 12$ Bit ergeben sich für die zweistufige Funktion nun 2^{176} mögliche Funktion gegenüber 2^{96} Möglichkeiten für 2-Mux-XOR.

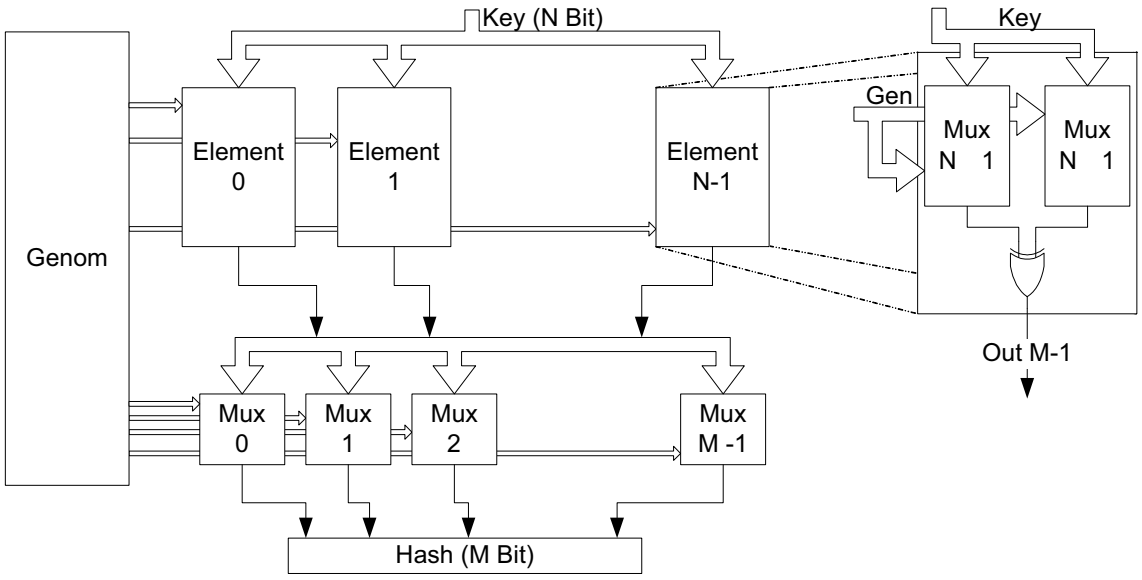


Abbildung 67 - Zweistufige Hashfunktion mit XOR-verknüpften Multiplexern (2-Mux-XOR-2-Stage).

5.2.3 Drei Multiplexer mit XOR-Verknüpfung

Eine weitere Möglichkeit zur Flexibilisierung der Hashfunktion besteht in der Vervielfachung der Multiplexer in den funktionalen Elementen. In Abbildung 68 ist eine Architektur mit jeweils drei Multiplexern, deren Ausgänge XOR-verknüpft sind, abgebildet.

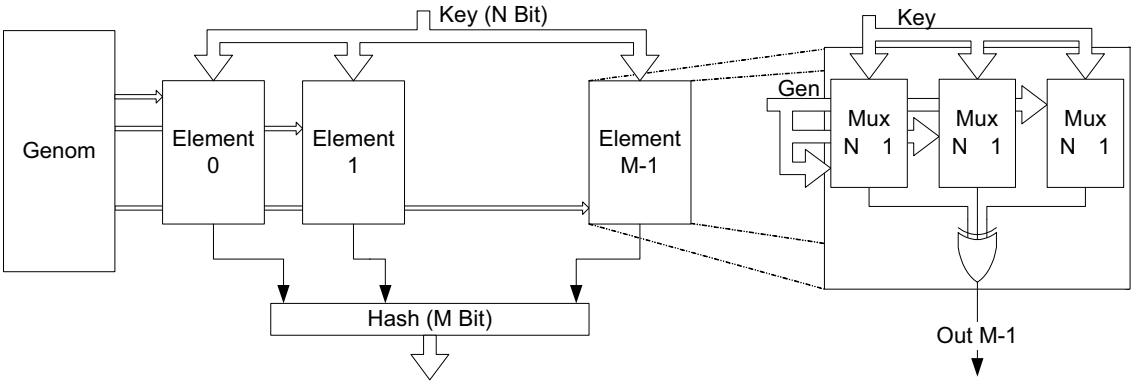


Abbildung 68 - Hashfunktion mit drei XOR-verknüpften Multiplexern (3-Mux-XOR).

Die Nutzung von drei statt zwei Multiplexern erweitert ebenfalls die Möglichkeiten der Selektion einer geeigneten Hashfunktion zum optimalen Hashing der vorliegenden Schlüssel. Die Genomgröße erweitert sich damit auf:

$$l = M \cdot 3 \cdot \lceil \log_2(N) \rceil \tag{19}$$

5.2.4 Multiplexer mit variabler Logikverknüpfung

Zusätzliche Variabilität in die Hashfunktionsarchitekturen aus verknüpften Multiplexern bringt die Einführung eines weiteren Logikblocks, der statt einer festen XOR-Verknüpfung der Multiplexersignale eine beliebige logische Funktion abzubilden vermag (Abbildung 69).

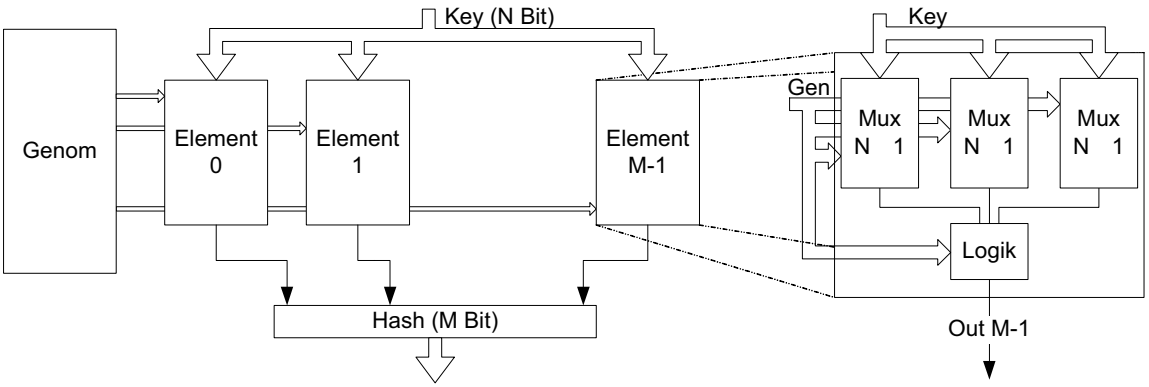


Abbildung 69 - Hashfunktion mit drei Multiplexern, welche mit einer beliebigen, variablen 3-Eingangslogikfunktion verknüpft sind

Das gilt sowohl für die Architektur aus zwei als auch für die aus drei. Dadurch ergibt sich eine Genomgröße bei zwei variabel verknüpften Multiplexern von

$$l = M \cdot \left[2 \cdot \log_2(N) + 4\right] \tag{20}$$

Für drei Multiplexer ist die Genomgröße demzufolge

$$l = M \cdot \left[3 \cdot \log_2(N) + 8\right] \tag{21}$$

Damit vergrößert sich die Variabilität der Funktion im Vergleich zu den anderen Architekturen noch weiter.

5.2.5 Evolvierbare CRC32-Berechnung

In [BM01] finden CRC-Funktionen Verwendung zur Erzeugung von Hashfunktionen. Dass CRC32 eine potentiell sehr gute Hashfunktion sein kann, zeigte Jain in [Jai92], wie bereits in Abschnitt 3.2.3.1 erläutert wurde. Ein wichtiger Vorteil gegenüber allen anderen hier vorgestellten Hashfunktionsarchitekturen ist die geringe Größe der Implementierung und der Umstand, dass zur Berechnung eines CRC kein GA in das System integriert werden muss. Der evolvierbare Paketklassifizierer beschränkt sich in diesem Fall auf den Datenpfad, der nicht doppelt ausgelegt werden muss. Mit nur 32 Slices ist die Implementierung der CRC32-Funktionalität überaus ressourcensparend. Die CRC32 Berechnung kann mittels Veränderung des CRC-Generatorpolynom ebenfalls variabel gestaltet werden, wobei in diesem Fall das Polynom selbst als Genom der evolvierbaren Hashfunktion zu betrachten ist. Simulationen haben allerdings gezeigt, dass in keinem Fall ein Polynom gefunden wurde, das bessere Eigenschaften aufweist (eine bessere Hashfunktion erzeugt) als das CRC32-Generatorpolynom. Der Ansatz wurde deshalb nicht weiter verfolgt. Darüber hinaus ist die Implementierung der Polynomdivision mit einem variablen Generatorpolynom im Vergleich zum CRC32 sehr aufwendig und kann mit zufriedenstellenden Frequenzen nur in mehreren Takten berechnet werden.

5.2.6 Evolvable H_3 Hashing

Auch Hashfunktionen der Klasse H_3 bieten sich an, mittels eines genetischen Algorithmus optimiert zu werden. Durch das Genom muss dabei die Matrix $q \in Q$ definiert werden, die die beste Hashfunktion h_q hervorbringt, um eine bestimmte Schlüsselmenge zu hashen. Die Matrix hat jeweils *N* Zeilen und *M* Spalten. Damit ergibt sich für das Genom, das den Inhalt der Matrix und damit h_q definiert, eine Größe von:

$$l = M \cdot N \tag{22}$$

Die Architektur eines evolvierbaren H_3 Hashings kann Abbildung 70 entnommen werden. Dabei enthält jedes Funktionselement eine Zeile der Matrix, die mit dem jeweiligen Bit des Schlüssels UND-verknüpft wird, bevor eine XOR-Verknüpfung mit dem Ergebnis der vorhergehenden Stufe erfolgt.

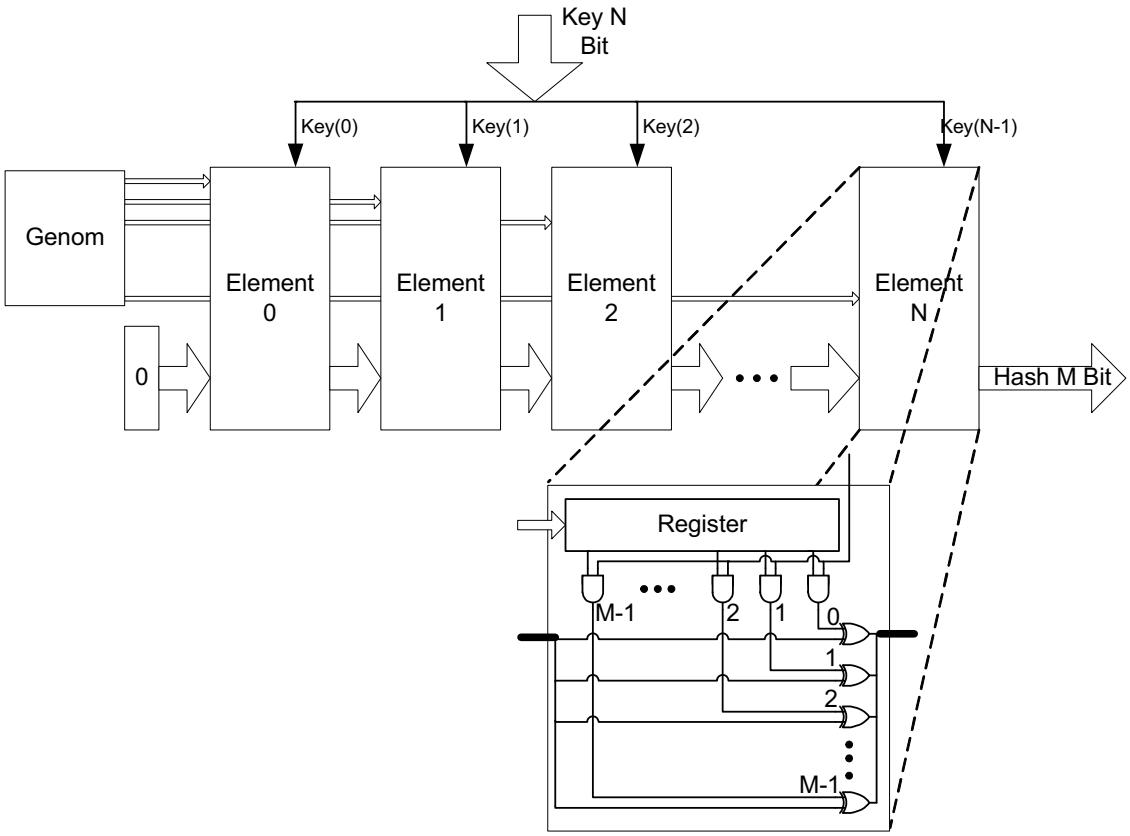


Abbildung 70 - Architektur der evolvierbaren Hashfunktion der Klasse H_3.

5.3 Der eingesetzte genetische Algorithmus

Der im EPC eingesetzte evolutionäre Algorithmus ist ein Genetischer Algorithmus. Er ist durch das folgende Tupel beschrieben:

$$GA = (P^0, \mu, \lambda, l, p, m, s, f, t) \tag{23}$$

Nach eingehender Untersuchung einer großen Zahl von GAs schlägt Grefenstette in [Gre86] die Nutzung eines Algorithmus mit den folgenden Parametern vor:

$\mu = 32$	Populationsgröße
$\lambda = 32$	Anzahl der Nachkommen
$l \in N$	Die Genomlänge ist abhängig von der eingesetzten Hashfunktion
$p_r = 0.95$	Rekombinationsoperator mit Rekombinationswahrscheinlichkeit p_r
$p_m = 0.01$	Mutationsoperator mit Mutationswahrscheinlichkeit p_m

Diese Parameter sind im GA des EPC umgesetzt. Eine genaue Beschreibung der Parameter, ihrer Funktion und ihrer Hardwareimplementierung erfolgt in den folgenden Unterabschnitten.

5.3.1 Fitnessfunktion und Abbruchkriterium

Um die Qualität einer bestimmten Konfiguration der genutzten Hashfunktion zu bestimmen, muss eine realistische Bestimmung der Fitness ermöglicht werden. Die Qualität einer Hashfunktion ist aus der Anzahl der Kollisionen ableitbar, die sie für eine bestimmte Schlüsselmenge erzeugt. Eine perfekte Hashfunktion erzeugt keinerlei Kollisionen für die zu nutzende Schlüsselmenge. Jeder Schlüssel wird an eine andere Adresse gehasht. Die schlechteste vorstellbare Hashfunktion andererseits erzeugt für jeden Schlüssel den gleichen Hashwert. Das ergibt für $|S|$ Schlüssel $\frac{|S|^2 - |S|}{2}$ Kollisionen. Es ist demzufolge naheliegend, die Summe aller Kollisionen der einzelnen Schlüssel s ($coll_s$) als Maß für die Fitness einzelner Individuen der aktuellen Generation von Hashfunktionen zu nutzen:

$$f(I) = \sum_{s=1}^{|S|} coll_s \tag{24}$$

Mit einer kleineren Anzahl von Kollisionen verringert sich auch $f(I)$. Die Hashfunktion ist besser. Daraus folgend ergibt sich das natürliche Abbruchkriterium mit:

$$t : f(I^{\lambda}) = 0 \tag{25}$$

Hardwareumsetzung

Die Struktur des Hardwaremoduls zur Evaluierung der Fitness eines Genoms ist in Abbildung 71 dargestellt. Es wird ein 1 Bit breiter Speicher bestehend aus BRAMs verwendet, der ebenso viele Einträge wie die Schlüsselspeicher im Datenpfad enthält, also doppelt so viele Einträge wie die maximale Anzahl der speicherbaren Schlüssel. Jedes Bit gibt an, ob die Speicherposition an der Adresse frei ist, oder bereits von einem Schlüssel verwendet wird.

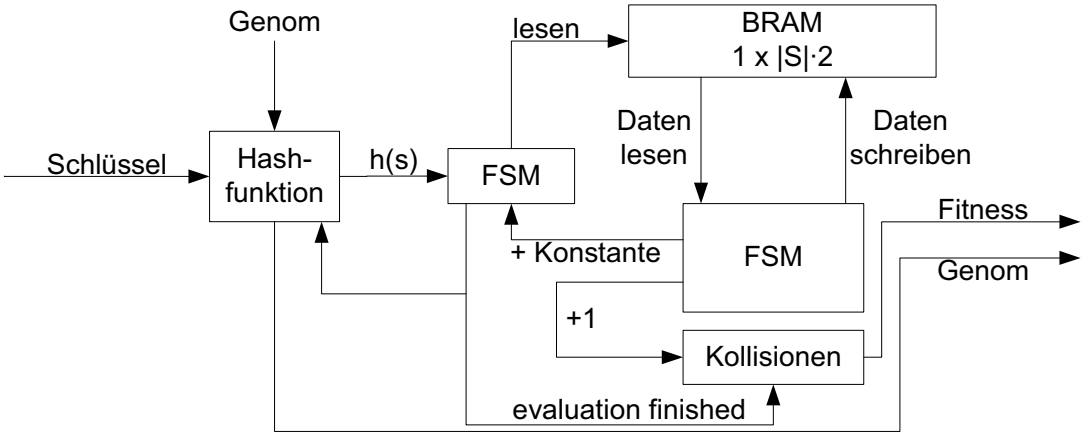


Abbildung 71 - Architektur des Fitnessevaluierungsmoduls.

Bevor die Evaluierung eines Individuums beginnt, ist der Speicher an allen Stellen auf '0' zurückzusetzen. Die Evaluierung beginnt mit der Konfiguration der Hashfunktion mit dem Genom des aktuellen Individuums. Durch die Verarbeitung aller Schlüssle wird die vollständige Evaluierung. Jeder Schlüssel *s* wird gehasht. Dann wird der Speicher an Adresse $h(s)$ gelesen. Ist er frei, d.h. der Eintrag ist '0', so wird er auf '1' (belegt) gesetzt und mit dem nächsten Schlüssel fortgefahren. Ist der Eintrag jedoch belegt, tritt eine Kollision auf. Der Kollisionszähler ist zu inkrementieren. Gleichzeitig muss nun die nächste Adresse bei $h(s) = h(s) + c$ gelesen werden. Die Kollisionsauflösung wird durchgeführt. Dies ist solange zu wiederholen, bis ein freier Speichereintrag gefunden wurde. Nachdem dieser Algorithmus für alle Elemente von *S* durchgeführt wurde, beinhaltet der Kollisionszähler die Summe aller Kollisionen. Dies ist gleichzeitig der Fitnesswert für das Genom des aktuellen Individuums. Eine perfekte Hashfunktion, die keine Kollisionen erzeugt, hat den Fitnesswert 0. Je kleiner also der Fitnesswert, desto besser ist die Hashfunktion mit dem aktuellen Genom.

5.3.2 Variationsoperatoren

Im Folgenden wird die Abbildung der verwendeten Operatoren des GA auf die evolvierbare Hardware dargestellt. Zur Bereitstellung einer größtmöglichen Flexibilität wurde neben den Standardoperatoren für GAs auch die Umweltselektion als Selektionsoperator implementiert. In vielen Funktionen des GA werden Zufallszahlen benötigt. Diese sind in allen Fällen als Linear Feedback Shift Register (LFSR) implementiert (siehe Abschnitt 2.3.2.3).

5.3.2.1 Elternselektion

Zur Implementierung der Elternselektion (parent selection) wurde die Wettkampfselektion, wie in Kapitel 2.3.1.1 beschrieben, gewählt. In μ Wettkampfrunden werden aus *k* Individuen der Elternpopulation, wobei *k* ein konfigurierbarer Parameter ist, jeweils die zwei Eltern mit der besten Fitness ausgewählt und an das Rekombinationsmodul übergeben. Mittels eines LFSRs als Pseudozufallszahlengenerator wählt das Modul die *k* Eltern aus den μ möglichen aus. Ein Vergleich der Fitnesswerte, die zusammen mit dem Genom in einem BRAM abgelegt sind, ermöglicht die Auswahl der besten Eltern zur Übergabe an das Rekombinationsmodul.

5.3.2.2 Rekombination

Das Rekombinationsmodul implementiert das 1-Point Crossover, wie in Kapitel 2.3.1.1 beschrieben. Hierbei findet, ebenso wie bei der Elternselektion und allen Hardwareelementen, die eine Zufallszahl benötigen, ein LFSR Verwendung. Zum einen wird mittels des LFSR entschieden, ob eine Rekombination stattfindet. Dies geschieht durch einen Vergleich des LFSR-Signals mit einem konfigurierbaren Parameter (recomb_probability). Soll eine Rekombination erfolgen, entscheidet der Wert des LFSR wiederum über die Position an der die genetische Information zwischen den beiden Eltern ausgetauscht wird, um zwei neue Individuen zu erzeugen. Die rekombinierten Individuen übergibt das Modul dann an das Mutationsmodul.

5.3.2.3 Mutation

Das Mutationsmodul mutiert einzelne Bits des Genoms eines Individuums. Das heißt, es invertiert sie. Dazu wird für jedes einzelne Bit des Genoms eines Elters eine Zufallszahl erzeugt. Liegt diese unter einer konfigurierbaren Schwelle, wird das jeweilige Bit nicht mutiert. Wenn alle Bits eines Genoms überprüft sind, überträgt das Modul das mutierte Genom an das Modul zur Fitnessevaluierung, das bereits in Unterabschnitt 5.3.1 erläutert wurde.

5.3.2.4 Umweltselektion

Nachdem im Fitnessevaluierungsmodul die Fitness jedes einzelnen Individuums ermittelt wurde, wird gegebenenfalls eine Umweltselektion durch das entsprechende Modul durchgeführt. In herkömmlichen GA-Implementierungen findet keine Umweltselektion statt. Sie ist nicht notwendig, da aus einer Population von μ Individuen einer Generation genau $\lambda = \mu$ Individuen für die Folgegeneration entstehen (siehe Abschnitt 2.3.1.1). Sollte jedoch eine Nachkommenschaft von $\lambda > \mu$ erzeugt werden, selektiert das Umweltselektionsmodul die μ fittesten der λ Nachkommen, um eine neue Elterngeneration zu bilden.

5.4 Leistungsvergleich verschiedener Hashfunktionen

Um die unterschiedlichen hier vorgestellten Hashfunktionen miteinander und mit den in der Literatur vorgestellten Hashfunktionen zu vergleichen, wurden alle Architekturen zusammen mit den anderen Elementen des EPCs als VHDL-Beschreibung für eine FPGA-Plattform implementiert. Die verwendete Evaluierungsplattform, das ML405-Entwicklungsboard von Xilinx enthält neben einem Virtex-4FX20 FPGA einen SRAM mit einer Kapazität von 1 MByte. Der Belegungsfaktor des Speichers liegt bei $\alpha = 0{,}5$. Außerdem müssen sich die beiden logischen Speicher von Daten- und Kontrollpfad den einen SRAM teilen. Unter diesen Randbedingungen können maximal 32.768 Speichereinträge vorgenommen werden und somit ist auch die Größe der verwendbaren Schlüsselmenge auf diesen Wert begrenzt.

5.4.1 Testdaten

Um die Qualität der Hashfunktionen zu bewerten, wurden realistische Schlüsselmengen verwendet. Diese entstammen einer BGB-Routingtabelle aus dem „Route Views Project“ der Universität von Oregon [Rou07]. Die verwendete Routingtabelle enthält insgesamt 167.254 Einträge. Sie ist auf dem Stand vom November 2007. Die Verteilung der Präfixe in der Tabelle kann Abbildung 72 entnommen werden. Wie zu sehen ist, entspricht die Verteilung der Präfixe den aus der Literatur bekannten Verteilungen. Es existieren keine Präfixe mit weniger als 8 Bit. Außerdem gibt es keine Präfixe, die 31 Bit umfassen.

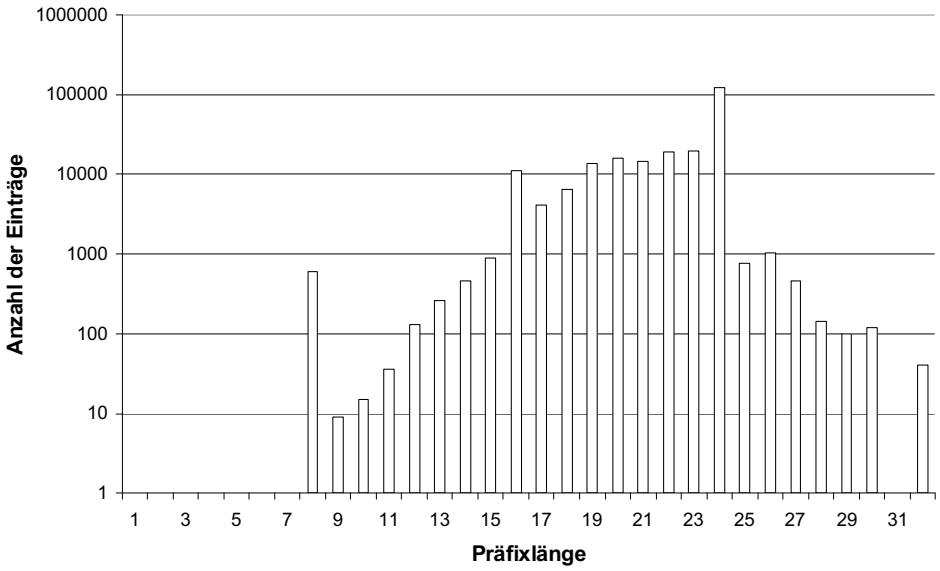


Abbildung 72 - Präfixverteilung der verwendeten BGP-Routingtabelle (logarithmische Skalierung).

Die Nutzung dieser Tabelle erscheint geeignet, um Aussagen zur Qualität der Hashfunktionen machen zu können. Es gibt andere Routingtabellen wie MAE-EAST und -WEST oder AADS. Sie alle zeigen ein ähnliches Verteilungsmuster der Präfixe [Wal00].

Um aussagekräftige Untersuchungsdaten zu erfassen, wurden 20 unterschiedliche Datensätze aus den zur Verfügung stehenden Routinginformationen erzeugt. Alle Hashfunktionen wurden mit diesen Datensätzen überprüft. Wegen der oben beschriebenen Einschränkungen der genutzten Hardwareplattform wurden die meistern Untersuchungen mit Schlüsselmengen der Größe 32.768 durchgeführt. Abweichungen werden an entsprechender Stelle beschrieben. Die Hashfunktionen wurden mittels des GA über 2000 Generationen optimiert. Danach wurde deren Qualität evaluiert und eine Bewertung vorgenommen.

5.4.2 Merkmale der verschiedenen Hashfunktionen

Die in den Abschnitten 5.2.1 bis 5.2.6 vorgestellten Architekturen besitzen unterschiedliche Eigenschaften. In Tabelle 9 sind die unterschiedlichen Genomgrößen der aktuellen Implementierung für das im Prototyp verwendete Virtex-4FX20 FPGA für unterschiedlich große Schlüsselmengen aufgeführt.

Tabelle 9 - Genomgrößen der verschiedenen Hashfunktionsarchitekturen bei unterschiedlich großen Schlüsselmengen [Bit].

Schlüsselmenge / Architektur	8k	16k	32k	64k	128k
Zwei Multiplexer XOR-verknüpft	130	140	150	160	170
Zwei Multiplexer zweistufig	385	390	395	400	405
Drei Multiplexer XOR-verknüpft	195	210	225	240	255
Zwei Multiplexer variabel verknüpft	196	210	224	238	252
Drei Multiplexer variabel verknüpft	203	226	249	272	295
Variable CRC-Generierung	32	32	32	32	32
Evolvable H_3 Hash	448	480	512	544	572

Tabelle 10 enthält analog dazu die Informationen zu den Hardwarekosten der Hashfunktionen, die in Hardware implementiert wurden. Dabei ist zu beachten, dass die Architekturen, die mit Hilfe von Multiplexern implementiert wurden, potentiell sehr viel kleiner sein können, wenn die partielle Rekonfigurierbarkeit von FPGAs weitere Fortschritte macht. Die Multiplexer dienen der Auswahl der Schlüsselbits. Wenn man in der Lage ist, Teile eines FPGAs zu rekonfigurieren, ist ein solcher Multiplexer nicht mehr notwendig. Die gewählten Signale können direkt als Draht an die XOR-Gatter geroutet werden. Die Hardwarekosten wären dann praktisch auf diese Gatter beschränkt. Da partielle Rekonfiguration noch erforscht wird, wurden hier die Multiplexer verwendet.

Tabelle 10 - Ressourcenverbrauch der verschiedenen Hashfunktionsarchitekturen bei unterschiedlich großen Schlüsselmengen [Slices].

Schlüsselmenge / Architektur	8k	16k	32k	64k	128k
Zwei Multiplexer XOR-verknüpft	352	376	402	429	456
Zwei Multiplexer zweistufig	934	946	964	958	963
Drei Multiplexer XOR-verknüpft	540	545	571	612	645
Zwei Multiplexer variabel verknüpft	388	419	452	489	516
Drei Multiplexer variabel verknüpft	600	641	686	723	731
CRC32	32	32	32	32	32
Variable CRC-Generierung	360	367	375	384	393
Evolvable H_3 Hash	506	535	561	608	641

Bei der Abschätzung der Kosten für die Implementierung der evolvierbaren Hashfunktionen auf einem FPGA sollten aus dem genannten Grund die Kosten für die Hashfunktionen selbst nicht überbewertet werden.

5.4.3 Performance der Hashfunktionen

Um die Leistungsfähigkeit der vorgeschlagenen Architekturen von evolvierbaren Hashfunktionen zu überprüfen, wurden sie zusammen mit den aus der Literatur bekannten Hashfunktionen untersucht. An dieser Stelle soll beantwortet werden, welche Hashfunktionsarchitektur die geringste Anzahl von Kollisionen erzeugt und ob sich der Einsatz eines GA zur Optimierung der Hashfunktionen lohnt. Die Kollisionsanzahl entspricht dem Fitnesswert des GAs. Die Ergebnisse der Untersuchungen können Abbildung 73 entnommen werden. Dargestellt ist die erreichte Fitness der unterschiedlichen Hashfunktionen bei einer Schlüsselmenge mit $|S| = 32.768$. Es ist zu erkennen, dass von den in der Literatur bekannten Funktionen CRC32 mit einer Fitness von im Mittel 15.215 die besten Ergebnisse aufweist. Es zeigt sich allerdings auch sehr deutlich, dass die vom Autor entwickelten und mittels des GA optimierten Hashfunktionen zum Teil um einiges bessere Fitnesswerte aufweisen.

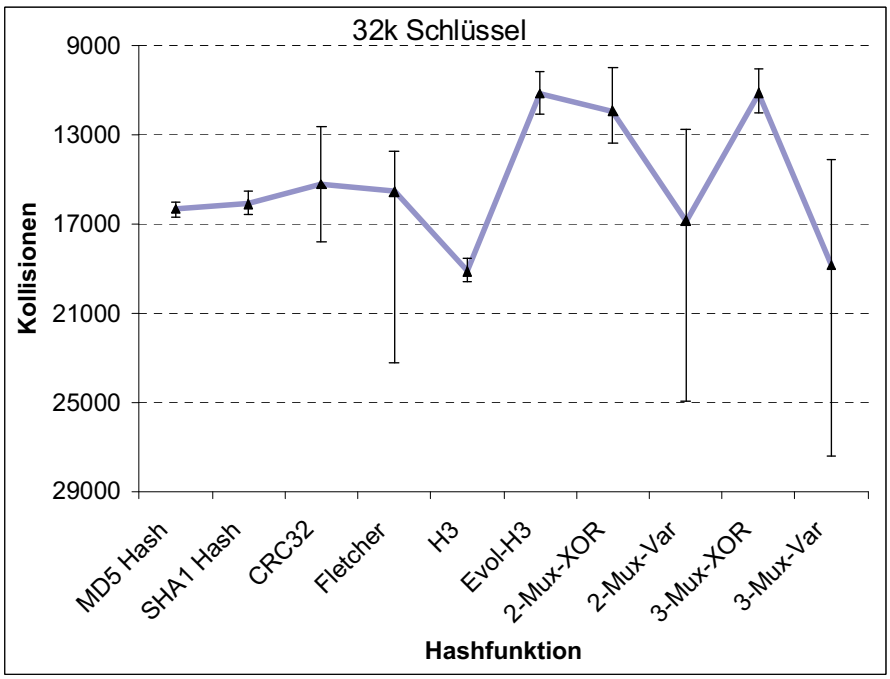
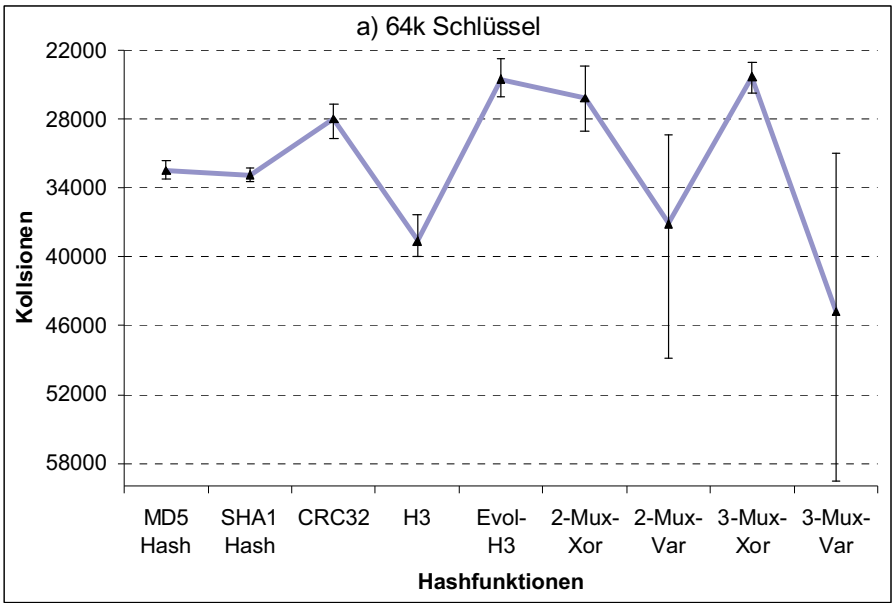


Abbildung 73 - Mittlere Fitness (Kollisionsanzahl) unterschiedlicher Hashfunktionen mit 32k Schlüsseln aus realistischen Schlüsseldaten. Es ist jeweils der Fitnesswert über der Architektur aufgetragen. Je kleiner der Fitnesswert ist, desto besser ist die Leistungsfähigkeit der jeweiligen Hashfunktion.

2-Mux-Var und 3-Mux-Var bleiben hinter den Erwartungen zurück. Offenbar sind die variablen Ausgänge weniger gut geeignet als feste XOR-Verküpfungen. 2-Mux-XOR und 3-MUX-XOR zeigen mit 11.921 bzw. 11.109 sehr gute Fitnesswerte. Sie sind um 22% bzw. 27% besser als die CRC32 Hashfunktion. Auch das evolvable-H_3 Hashing zeigt ausgezeichnete Ergebnisse. Mit einer Fitness von im Mittel 11.152 ist diese Architektur besser als 2-Mux-XOR und nur leicht schlechter als 3-Mux-XOR.



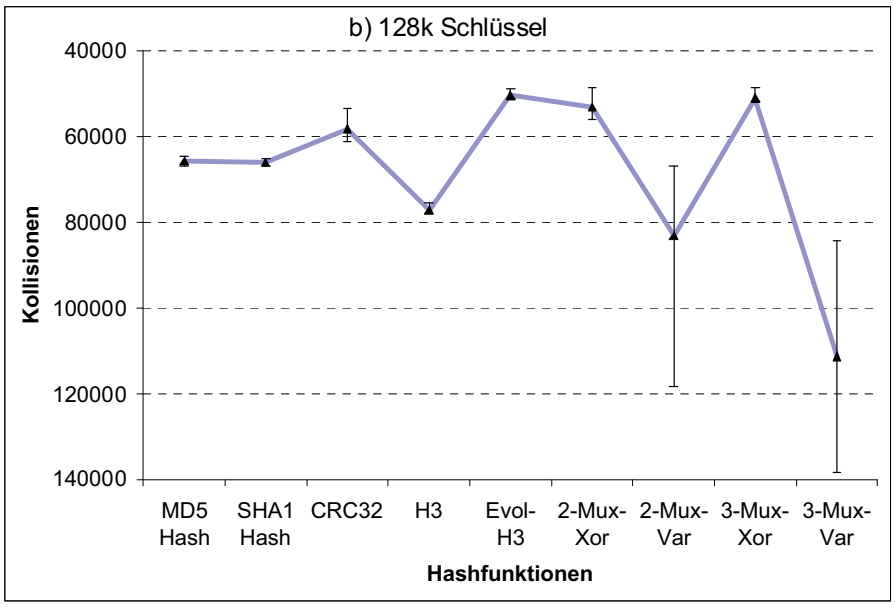


Abbildung 74 - Mittlere Kollisionsanzahl unterschiedlicher Hashfunktionen mit 64k und 128k Schlüsseln aus realistischen Schlüsseldaten.

Um auch Schlüsselmengen mit $|S| = 65.536$ und $|S| = 131.072$ untersuchen zu können, wurde die Hardwarearchitektur angepasst. Dazu wurde nur der Kontrollpfad implementiert. Abbildung 74 ist das Verhalten der Hashfunktionen für diese größeren Schlüsselmengen zu entnehmen. Es ist zu erkennen, dass die relative Leistungsfähigkeit der Hashfunktionen zueinander auch für 64k und 128k Schlüssel gleich bleibt. 3-Mux-XOR weist für 131.072 Schlüssel im Mittel 51.027 Kollisionen auf. Das bedeutet, dass pro Schlüssel 0,39 Kollisionen auftreten. Ein beliebiger Eintrag kann damit mit durchschnittlich 1,39 Speicherzugriffen gefunden werden. CRC32 hat im Mittel 58.201 Kollisionen. Damit ist auch in diesem Fall 3-Mux-XOR besser als CRC32, jedoch nur noch um 13%.

Wie Abbildung 75 verdeutlicht, ist das Verhalten der Hashfunktionen für unterschiedlich große Schlüsselmengen vergleichbar. Es wurden Schlüsselmengen von 32k, 64k und 128k Größe untersucht. Die Gegenüberstellung im Balkendiagramm belegt, dass die Anzahl der Speicherzugriffe für alle Schlüsselmengengrößen nahezu konstant bleibt bzw. nur sehr moderat mit zunehmender Schlüsselmengengröße zunimmt.

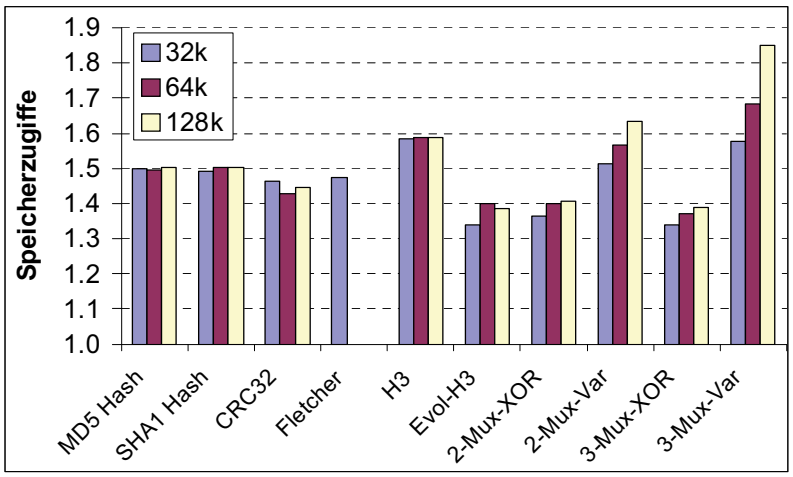


Abbildung 75 - Speicherzugriffe pro Schlüssel für unterschiedlich große Schlüsselmengen der verschiedenen Hasharchitekturen. Je weniger Speicherzugriffe notwendig sind, desto besser ist die Hashfunktion. 1,0 Speicherzugriffe stellen das theoretische Minimum dar. Dieses wäre bei einer perfekten Hashfunktion erreicht. Für die Flechtercheckssumme liegen nur Werte für 32k Schlüssel vor, da der Hashwert maximal 16 Bit breit ist.

Die beiden besten Architekturen sind 3-Mux-XOR und evolvable-H_3. Sie haben sehr ähnliche Leistungen erreicht. Deshalb wurden diese Architekturen noch einmal über einen längeren Zeitraum untersucht. In Abbildung 76 sind die Ergebnisse dargestellt. Auch nach 100.000 Generationen liegen die Resultate beider Architekturen sehr nah zusammen. Es zeigt sich aber die Tendenz, dass die 3-Mux-XOR Architektur leichte Vorteile aufweist, je länger der genetische Algorithmus arbeitet, wobei Unterschiede in der Fitness auch nach 100.000

Generationen mit 10.781 (evolvalble-H_3) zu 10.728 (3-Mux-XOR) sehr gering ausfallen. Gleiches gilt für die notwendigen Hardwareressourcen. 3-Mux-XOR benötigt 571 Slices und evolvalble-H_3 565 Slices. Es ist jedoch zu beachten, dass die Implementierung von 3-Mux-XOR aus bereits genannten Gründen (partiell, dynamische Rekonfigurierbarkeit) sehr viel effizienter erfolgen kann. Perspektivisch sind die Hardwarekosten der 3-Mux-XOR-Architektur verschwindend gering. Mit 10.781 gegenüber den 15.215 vergrößert sich der Abstand der 3-Mux-XOR Architektur von CRC32 aber mit jeder Generation immer weiter. Nach 100.000 Generationen erzeugt 3-Mux-XOR um 29% weniger Kollisionen.

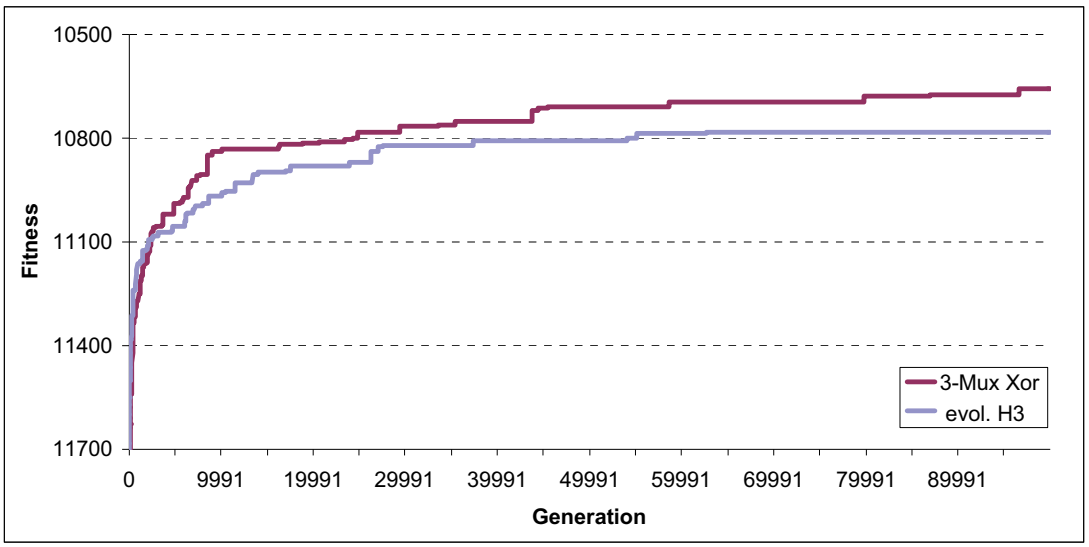


Abbildung 76 - Verhalten von 3-Mux-XOR und evolvable-H3 über 100.000 Generationen (32k Schlüssel).

5.4.4 Alternative Schlüssel

Es wurden auch Untersuchungen mit anderen Schlüsselmengen als denen aus dem „Route Views Project“ durchgeführt. Zum einen wurden Datensätze mit einfachen zufälligen Schlüsseln erzeugt. Zum anderen wurde eine alternative Routingtabelle, nämlich die des deutschen Internet Exchange Knotens in Frankfurt/Main (DE-CIX) verwendet [Rip08].

Es zeigt sich, dass die Ergebnisse mit zufälligen Zahlen nicht ganz so gut sind, wie die Ergebnisse, die mit realen Daten erreicht werden. Offenbar wirkt sich bei den realen Daten die größere Korrelation der Daten im Vergleich zu zufällig erzeugten Daten positiv auf das Ergebnis aus. Bei den realen Daten sind beispielsweise viele Schlüssel vorhanden, bei denen die letzten 8 oder 16 Bit Null sind. Diese Bitpositionen werden vom Algorithmus dann nicht mehr oder seltener zur Erzeugung der Hashfunktion verwendet. Die zufällig mittels eines Zufallszahlengenerators erzeugten Schlüssel sind nicht korreliert. Wie Abbildung 77 zu entnehmen ist, bleiben die Fitnesswerte für die zufälligen Daten hinter den erreichten Werten

für die realen Schlüsseldaten zurück. Der Fitnessunterschied beträgt bei 32k Schlüssel 35% und verringert sich bis auf 26% bei 128k Schlüsseln. Die Daten aus den beiden realen Datensätzen zeigen sehr ähnliche Ergebnisse. CRC32 verhält sich analog zur 3-Mux-XOR Architektur. Die Ergebnisse bleiben sowohl bei den realen Datensätzen als auch bei zufälligen Schlüsseln hinter 3-Mux-XOR zurück.

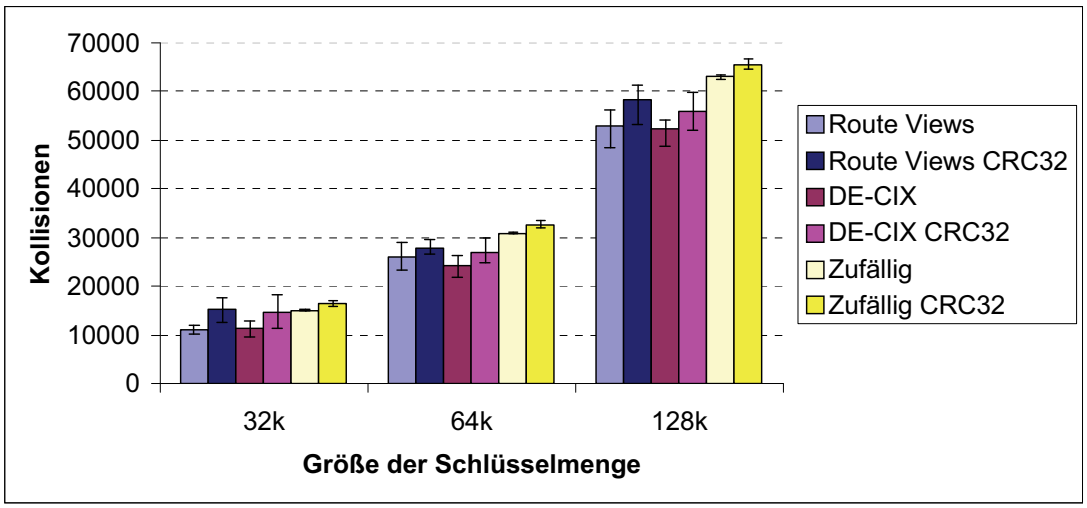


Abbildung 77 - Fitness der Architektur mit 3-Mux-XOR und CRC32 bei zufällig generierten Schlüsseln und unterschiedlichen realen Schlüsseln.

5.4.5 Veränderliche Schlüsselmengen

Der Autor untersuchte ebenfalls die Fähigkeit der evolvierbaren Hashfunktion, auf veränderliche Schlüsselmengen zu reagieren. Dazu wurde jeweils eine Schlüsselmenge mit $|S| = 32.868$ Schlüsseln verwendet und evaluiert. Nach einer festen Anzahl von Generationen wurden mehrmals 12,5% der Schlüssel ausgetauscht, bis ein vollständiger Austausch der Schlüsselmenge vollzogen war. Dabei wiesen die alte und die neue Schlüsselmenge keine gemeinsamen Schlüssel auf. Die Fitness im Verlauf der Evaluierung wurde in jeweils zehn unterschiedlichen Simulationsläufen untersucht.

Wie aus Abbildung 78 hervorgeht, verbessert sich die Fitness nach einem Austausch von Teilen der Schlüsselmengen wieder. Auch ist der Rückgang der Fitnesswerte nicht sehr stark, da nur jeweils 12,5% der Schlüssel verändert wurden. Es ist allerdings auch festzustellen, dass das erreichte Fitnessniveau bis zu einem Austausch von 50% der Schlüssel abnimmt, um danach wieder anzusteigen. Wenn die gesamte Schlüsselmenge ausgetauscht ist, wird annähernd, wenn auch nicht ganz, das Anfangsniveau erreicht. Die Untersuchungsergebnisse verdeutlichen nach Meinung des Autors, dass die erst ab- und dann wieder zunehmende Korrelation der Daten großen Einfluss auf die Performance der Hashfunktion hat.

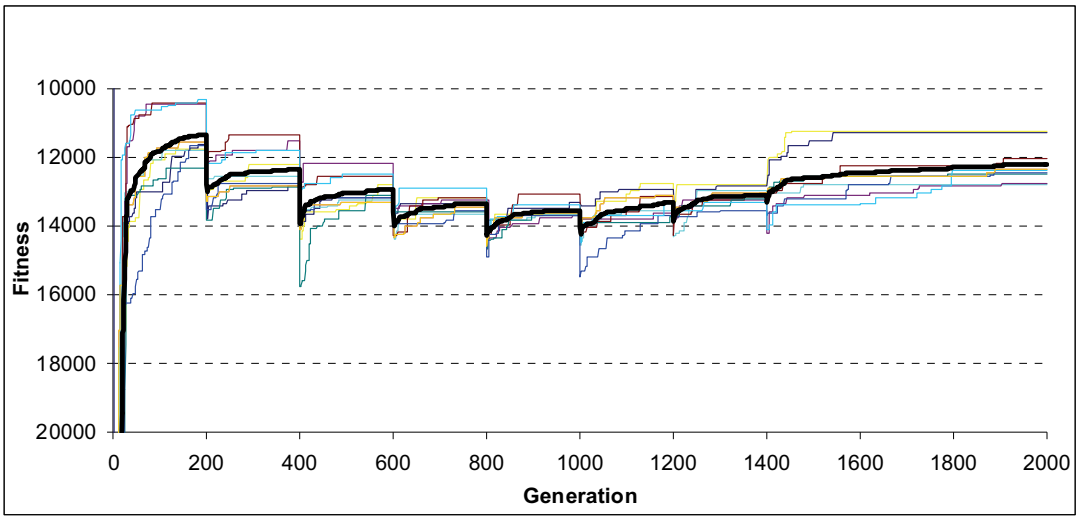


Abbildung 78 - Fitnessverlauf des GA mit 32k Schlüsseln über 2000 Generationen. Es wurden alle 200 Generationen 4k Schlüssel ausgetauscht, bis ein kompletter Schlüsselaustausch sattfand.

Auch veränderte Simulationsdauern, deren Ergebnis in Abbildung 79 dargestellt sind, zeigen die gleichen Ergebnisse. Auch über insgesamt 5000 Generationen, wobei immer alle 500 Generationen 12,5% der Schlüssel ausgetauscht wurden, zeigen sich die gleichen Ergebnisse. Dennoch bleibt in jedem Fall die Anzahl der Kollisionen mit weniger als 14.000 sehr gering.

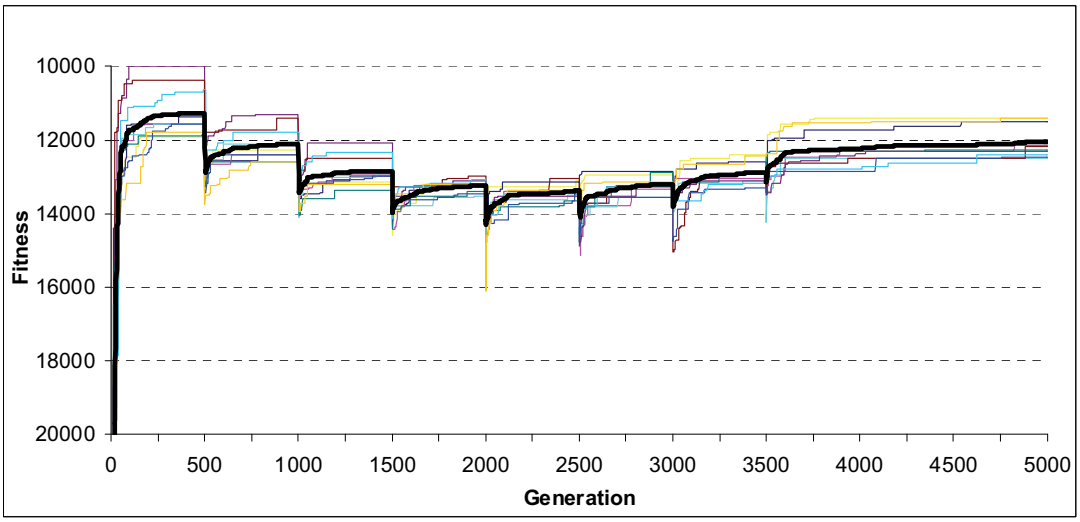


Abbildung 79 - Fitnessverlauf des GA mit 32k Schlüsseln über 5000 Generationen. Es wurden alle 500 Generationen 4k Schlüssel ausgetauscht, bis ein kompletter Schlüsselwechsel stattfand.

5.5 Zusammenfassung

In diesem Kapitel wurde ein vom Autor entwickelter einfacher Paketklassifizierer vorgestellt, der auf der Basis einer evolvierbaren Hashfunktion arbeitet [WHT05, WST06, SWT06]. Um eine leistungsfähige Paketklassifizierung zu ermöglichen, wurden verschiedene Hardwarearchitekturen für Hashfunktionen entwickelt, die allesamt mittels eines GA an eine konfigurierte Schlüsselmenge angepasst werden können. Des Weiteren wurde der notwendige GA entwickelt und in Hardware implementiert.

Die verschiedenen Architekturen wurden mit Daten aus einer realistischen Schlüsselmenge auf ihre Leistungsfähigkeit gegenüber herkömmlichen und aus der Literatur bekannten Hashfunktionsarchitekturen hin untersucht. Dabei ergab sich, dass die 3-Mux-XOR-Architektur die besten Ergebnisse liefert. Sie erzeugt bei der Nutzung von 32.768 Schlüsseln durchschnittlich 11.109 Kollisionen. Das sind 4106 Kollisionen weniger aufgetreten als bei CRC32, die beste dem Autor bekannte Hashfunktionsarchitektur. Das entspricht einer Verminderung von 27%. Damit ist es möglich, mit im Mittel 1,34 Speicherzugriffen den richtigen Speichereintrag zu finden. Die vorgestellte Architektur benötigt allerdings weitaus mehr Hardwareressourcen als CRC32. Es ist jedoch zu erwarten, dass mit der Unterstützung von partieller Rekonfiguration durch die FPGA-Hersteller, die Hardwarekosten für die 3-Mux-XOR-Architektur stark absinken.

Die 3-Mux-XOR-Architektur kann beliebig angepasst werden, was die Eingangs- und auch die Ausgangswortbreite betrifft. Sie kann deshalb als universelle Hashfunktion auch in die in Abschnitt 3.2.3.2 vorgestellten hash-basierten Klassifizierarchitekturen integriert werden. Im Unterschied zum EPC unterstützen die aus der Literatur bekannten Architekturen auch „Prefix Match“. Der EPC selbst kann dennoch vor allem im Bereich der TZNs sinnvoll eingesetzt werden. Die in Abschnitt 4.3 vorgestellten Funktionalitäten von MAT, TM, MPLS-UNI und IPclip benötigen nur das „Exact Match“, um Teilnehmer zu identifizieren. Bei diesen Funktionalitäten ist die Unterstützung von „Prefix Match“ nicht notwendig.

6 Systemoptimierungen

Wie die Untersuchungen aus Abschnitt 5.4.4 gezeigt haben, ist die Systemperformance unter realen Bedingungen im Vergleich zu Zufallszahlen besser. Dennoch ist der Durchsatz im Datenpfad insbesondere am Anfang des evolutionären Prozesses gering, da die durchschnittliche Anzahl von Speicherzugriffen hier sehr groß ist. Zum Anderen ist eine recht lange Zeit notwendig, damit sich ein guter Klassifizierer für einen bestimmten Schlüsselsatz entwickelt. Des Weiteren vermindert sich die Dynamik, mit der der Klassifizierer im Betrieb auf veränderliche Schlüsselmengen reagiert. Es ist demzufolge notwendig, die Systemleistung zu optimieren. Lösungen wurden hierbei für folgende Teilaspekte untersucht und entwickelt:

- Start in die Entwicklung mit besonders vielversprechenden Genomen
- Optimierung der Parameter des GA
- Reinitialisierung des GA, nachdem dieser konvergierte
- Verbesserung der Geschwindigkeit der Fitnessevaluierung
- Verbesserung der erreichbaren Fitnesswerte
- Beschleunigung der Klassifizierung im Datenpfad

Alle im Folgenden vorgestellten Optimierungen wurden in Hardware implementiert. Die erreichten experimentellen Ergebnisse wurden mit dem auf einem ML405 implementierten Prototypsystem ermittelt und basieren nicht auf Simulationen mit Softwaremodellen.

6.1 Beschleunigung der Fitnessevaluierung (ET, PFE, MI)

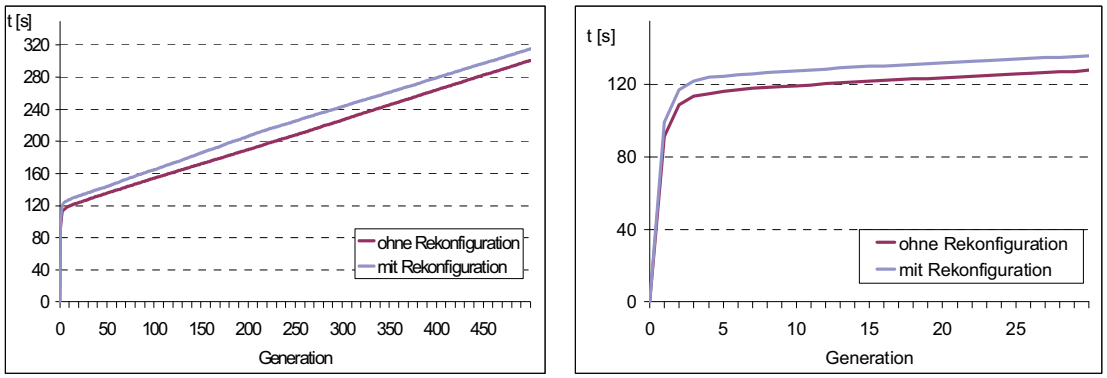
Wie einleitend erläutert, ist ein schneller Ablauf des GA wichtig, um möglichst schnell eine brauchbare Konfiguration für den EPC zu finden. Die Fitnessevaluierung ist der Teilprozess im Verlauf des GA, der die weitaus meiste Rechenzeit benötigt. Für die Evaluierung einer kompletten Generation von Hashfunktionen müssen alle Nachkommen unabhängig voneinander evaluiert werden. Aus der Architektur der Fitnessfunktion und ihrer Implementierung als Summe aller Kollisionen, die beim Hashing des kompletten Schlüsselsatzes auftreten, ergibt sich für den *i*-ten Nachkommen eine Evaluierungszeit T_i von:

$$T_i = \sum_{s=1}^{|S|} \tau \cdot (1 + coll_s) \tag{26}$$

Bei Berechung der λ Nachkommen unabhängig voneinander ergibt sich die Evaluierungszeit T_g einer Generation. Dazu kommt die Zeit T_{reconf}, die notwendig ist, um den Datenpfad zu rekonfigurieren:

$$T_g = \sum_{i=1}^{\lambda} (T_i) + T_{reconf} \tag{27}$$

Die Rekonfiguration der Hashfunktionen im Datenpfad, wozu ein komplettes Rehashing gehört, findet nicht in jeder neuen Generation statt. Eine Rekonfiguration wird nur dann erforderlich, wenn die neue Generation der Hashfunktionen einen besseren Fitnesswert aufweist, als die alte. Nur dann T_{reconf} auf. Die Rekonfigurationszeit ist von der Geschwindigkeit der Fitnessevaluierung unabhängig. Da sie parallel zu den Datenpfadoperationen stattfindet, hängt T_{reconf} vor allem vom Datenaufkommen im Datenpfad ab. Wie Abbildung 80 verdeutlicht, ist T_{reconf} jedoch auch im optimalen Fall (keine Daten im Datenpfad vorhanden) für die Dauer der Gesamtberechnung relevant. Es ist allerdings anzumerken, dass der Einfluss in späteren Generationen abnimmt, da die Häufigkeit der Rekonfigurationen abnimmt und sich die Zeit für die einzelne Rekonfiguration mit besseren Fitnesswerten verkürzt.



a) Berechnungszeit des GA mit und ohne Rekonfiguration

b) vergrößerte Darstellung

Abbildung 80 - Einfluss von T_{reconf} auf die Gesamtberechnungszeit des genetischen Algorithmus. Im Mittel steigt die Berechnungszeit bei 500 Generationen um 4,7%.

Die verhältnismäßig schlechte Fitness der Hashfunktion gerade zu Beginn der Hardwareevolution, beziehungsweise nach einer Neuinitialisierung des evolutionären Prozesses, hat zur Folge, dass besonders zu Beginn der Evolution pro Generation eine sehr lange Rechenzeit aufzuwenden ist. Das macht es erforderlich, den Vorgang der Fitnessevaluierung signifikant zu beschleunigen. Um eine realistische Einschätzung der Einflüsse der verschiedenen im Folgenden vorgestellten Beschleunigungsstrategien [WTS+07] zu gewinnen, wird das EPC-System so konfiguriert, dass keine Rekonfiguration der Hashfunktionen im Datenpfad stattfindet. Die eigentliche Fitnessevaluierung wird demzufolge nicht unterbrochen. Es ergibt sich die vereinfachte Gleichung für die Evaluierungszeit einer Generation von:

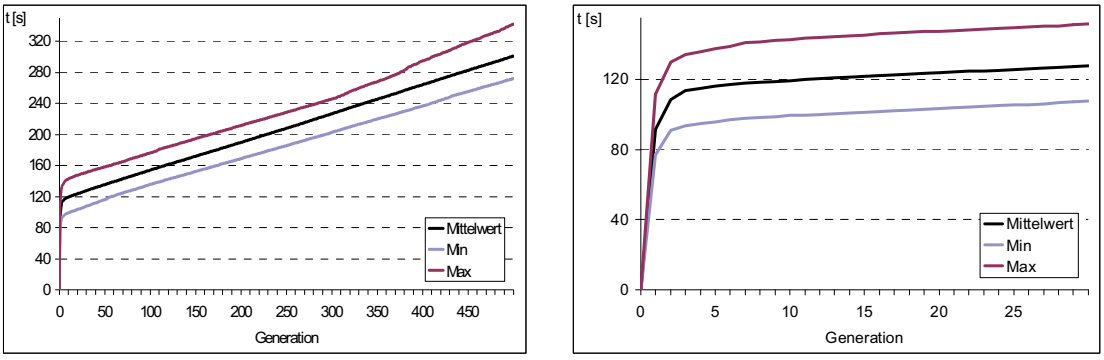
$$T_g = \sum_{i=1}^{\lambda} T_i \tag{28}$$

Durch das Verhindern der Rekonfiguration des Datenpfades sind genauere Aussagen über die tatsächliche Verbesserung des beeinflussbaren Teils der Evaluierungszeit möglich. Die restliche Konfiguration des EPC entspricht den Angaben in Tabelle 11. Mit dieser Konfiguration soll der Einfluss von drei unterschiedlichen Maßnahmen zur Beschleunigung der Fitnessevaluierung überprüft werden.

Tabelle 11 - Konfiguration des GA zur Überprüfung der Beschleunigung der Fitnessevaluierung durch Vorzeitigen Abbruch, Parallele Fitnessevaluierung und Speicherverschränkung.

Populationsgröße	32
Nachkommen	64
Elternselektion	Wettkampf (k = 4)
Mutationsrate	0.01
Reproduktionsrate	0.95
Umweltselektion	64 → 32
Hashfunktion	3-Mux-XOR
Schlüsseldaten	BGP-Routingtabelle (32.768 Einträge)

Aus Abbildung 81 geht die Rechenzeit des GA für die Berechnung von nur 500 Generationen hervor. Zur Berechnung von 500 Generationen werden im Mittel mehr als 5 Minuten auf der Hardwareplattform benötigt.



a) 500 Generationen

b) Vergrößerte Darstellung der ersten 30 Generationen

Abbildung 81 - Zeitverhalten des genetischen Algorithmus. Mittelwert, obere und untere Grenze bei 10 Untersuchungen.

Es ist zu erkennen, dass insbesondere die ersten Generationen immens viel Rechenzeit benötigen, da die Rechenzeit direkt von der Qualität der Hashfunktionen abhängt. Je mehr Kollisionen auftreten, desto länger benötigt der Algorithmus zur Berechnung einer Generation.

6.1.1 Vorzeitiger Abbruch (ET)

Wird für den GA eine Umweltselektion implementiert, wie sie in Kapitel 5.3.2.4 beschrieben wurde, können von λ Nachkommen nur die μ besten in die nächste Generation übernommen werden. Gibt es bereits μ fittere Individuen als das aktuell zu evaluierende, kann dieses Individuum keinesfalls mehr in die neue Generation übernommen werden. Es besteht demzufolge kein Grund, mit der Evaluierung der Fitness dieses Individuums fortzufahren. Es kann ein vorzeitiger Abbruch (Early Termination – ET) der Fitnessevaluierung vorgenommen werden. Um die ET zu implementieren, sind an der Architektur des GA-Moduls leichte Veränderungen vorgenommen worden. Das Umweltselektionsmodul stellt nun den Fitnesswert des µ-ten Nachkommens – die Terminierungsgrenze – zur Verfügung. Sollten während der Evaluation weniger als μ Nachkommen evaluiert worden sein, liegt dieser Wert bei seinem möglichen Maximum. Wie Abbildung 82 zu entnehmen ist, ist zur Feststellung der Terminierungsgrenze ein weiterer Dateneingang in das Fitnessevaluierungsmodul notwendig. Nach jeder Inkrementierung des Kollisionszählers vergleicht das Modul den aktuellen Fitnesswert nun mit dem Termination Limit. Sollte dieses überschritten worden sein, wird die Evaluierung abgebrochen und das entsprechende Ausgangssignal (*Evolutionsabbruch*) für einen Takt auf '1' gesetzt, um dem Speicherinterface zu signalisieren, dass die Evaluierung abzubrechen ist und keine Schlüssel mehr auszulesen sind.

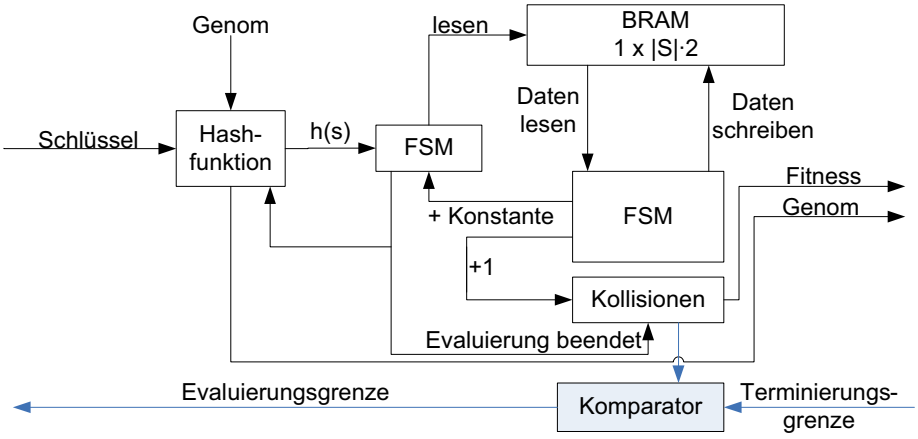


Abbildung 82 - Architektur der Fitnessevaluierung mit integrierter ET. Die Änderungen zur originalen Fitnessevaluierung sind farblich hervorgehoben.

Im Vergleich zur Implementierung der originalen Fitnessevaluierung ist nur ein sehr geringer zusätzlicher Hardwareaufwand notwendig. Die Gesamtgröße des EPCs erhöht sich um nur 0,82% von 6058 auf 6135 Slices.

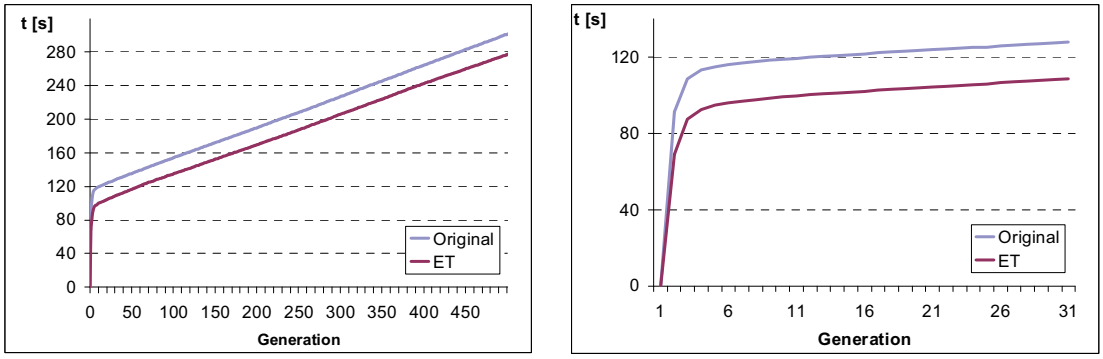
Die Geschwindigkeit der Fitnessevaluierung auf der anderen Seite steigert sich signifikant. Dabei sind zwei Grenzfälle zu betrachten. Im günstigsten Fall werden die µ besten Individuen zuerst evaluiert. Alle schlechteren Nachkommen können dann von der ET profitieren. Ihre

Evaluierung bricht bei erreichter Fitness des μ-ten Individuums ab. Daraus ergibt sich eine Evaluierungszeit von:

$$T_g = \sum_{i=1}^{\mu} T_i + (\lambda - \mu) \cdot \max_{i=1:\mu}(T_i) \qquad (29)$$

Im ungünstigsten Fall allerdings evaluiert das Modul die schlechtesten Individuen zuerst und die besten Nachkommen erst am Ende der Fitnessevaluation. In diesem Fall ist jedes Individuum vollständig zu evaluieren. Es ergibt sich keinerlei Verbesserung von T_g. Die Wahrscheinlichkeit, dass tatsächlich die μ schlechtesten von λ Individuen zuerst evaluiert werden, liegt bei $\mu!/\lambda!$. Je nachdem, wie μ und λ gewählt werden, ist nahezu immer eine Verbesserung durch ET zu erzielen. Schon bei einem (4,12)-GA liegt die Wahrscheinlichkeit keiner Verbesserung bei nur $1/19.958.400$.

Um die real zu erzielende Verbesserung der Evaluierungsgeschwindigkeit zu ermitteln, wurden vom Autor mit einem Hardwareprototyp mehrere Untersuchungen durchgeführt. Dabei wurde die Zeit gemessen, die benötigt wurde, um 500 Generationen des GA zu evaluieren. Basis ist die in Kapitel 5.2.3 vorgestellte leistungsfähigste Architektur in der Standardkonfiguration (siehe Kapitel 5.3). Abweichend davon ist die Anzahl der Nachkommen, die in jeder Generation erzeugt werden, von 32 auf 64 erhöht worden. Abbildung 83 stellt den Grad der Verbesserung über die Zeit dar, wenn 32k Schlüssel zur Evaluierung der Fitness verwendet werden.



a) 500 Generationen

b) Vergrößerte Darstellung der ersten 30 Generationen

Abbildung 83 - Einfluss des vorzeitigen Abbruchs auf die Rechenzeit des GA.

Wie dem Graph zu entnehmen ist, benötigt die originale Fitnessevaluierung im Schnitt 301 Sekunden. Wird ET eingesetzt, verringert sich die benötigte Evaluierungszeit auf nur noch 276 Sekunden. Das entspricht einer Erhöhung der Evaluierungsgeschwindigkeit um 9%. Das kann man als nicht zu vernachlässigende Verbesserung betrachten, zumal die zusätzlichen Hardwarekosten praktisch nicht ins Gewicht fallen. Es ist zu erwarten, dass der Grad der

Verbesserung mit einer Zunahme des Verhältnisses von λ zu μ weiter steigt, da eine größere Anzahl Evaluierungen vorzeitig abgebrochen werden kann. Ist $\lambda = \mu$ (es findet keine Umweltselektion statt), muss ET nicht eingesetzt werden.

6.1.2 Speicherverschränkung (MI)

Eine weitere Möglichkeit, von den Vorteilen einer Hardwareentwicklung zu profitieren, ist die parallele Überprüfung von Kollisionen. Im ursprünglichen System wird eine Speicherstelle überprüft, um festzustellen, ob diese bereits verwendet wird oder ob sie noch frei ist. In dem Fall, dass sie benutzt ist, ist der Speicher solange sukzessiv zu durchsuchen, bis eine freie Stelle gefunden wird, um den aktuellen Wert einzutragen. Dieser Prozess kann, zumindest teilweise, parallelisiert werden. Wenn nicht nur die Speicherposition an der Stelle $h(s)$, sondern auch die p-1 folgenden möglichen Einträge parallel überprüft werden, kann eine große Zahl von Speicherzugriffen gespart werden. Durch diese Speicherverschränkung (Memory Interleaving – MI) reduziert sich die Evaluierungszeit, die für ein Individuum notwendig ist, von

$$T_i = \sum_{s=1}^{|S|} \tau \cdot (1 + coll_s) \tag{30}$$

auf

$$T_i = \sum_{s=1}^{|S|} \tau \cdot \left(1 + \left\lfloor \frac{coll_s}{p} \right\rfloor\right) \tag{31}$$

wobei p den Grad der Parallelität, bzw. die Anzahl der gleichzeitig zu überprüfenden Einträge darstellt. Diese Verbesserung ist nahezu ohne zusätzlichen Hardwareaufwand zu erreichen (Abbildung 84).

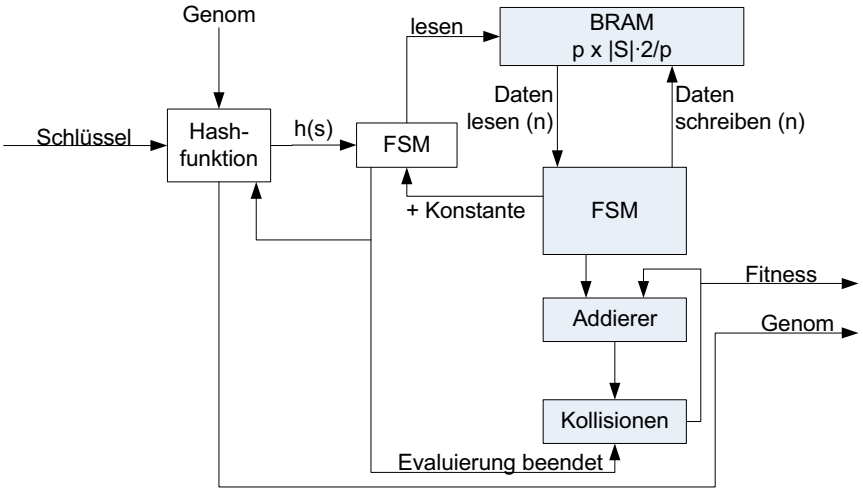
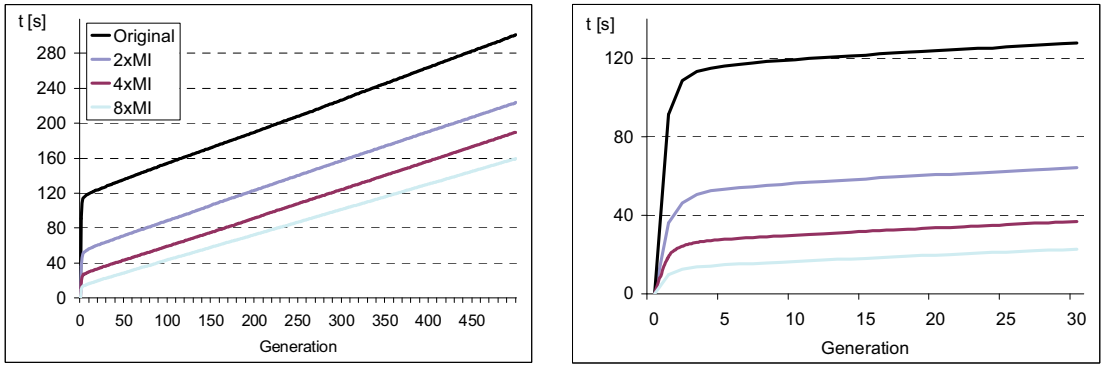


Abbildung 84 - Architektur der Fitnessevaluierung mit implementiertem Memory Interleaving.

Der notwendige BRAM ist lediglich von einem Speicher des Formats $1 \times 2 \cdot |S|$ zu einem $p \times \frac{2 \cdot |S|}{p}$ Speicher umzukonfigurieren. Die zusätzlichen Logikressourcen beschränken sich auf einen Addierer und einen p Bit breiten Komparator. In der Implementierung benötigen die Erweiterungen für MI maximal 4,57% zusätzliche Logik ($8 \times$ MI). Die Beschleunigung der Evaluierung auf der anderen Seite ist im Vergleich zu den Kosten beeindruckend.



a) Zeitverlauf über 200 Generationen

b) Zeitverlauf über die ersten 30 Generationen

Abbildung 85 - Zeitverlauf des GA. Einfluss der parallelen Fitnessevaluierung.

Wie aus Abbildung 85 hervorgeht, beschleunigt bereits ein $2 \times$ MI die Fitnessevaluierung um 35%. Die Evaluierungszeit verringert sich um 26% von 301 Sekunden auf 223 Sekunden. Bei $4 \times$ MI reduziert sich die Rechenzeit auf nur noch 189 Sekunden. Wird $8 \times$ MI eingesetzt, sind nur noch 159 Sekunden notwendig, um 500 Generationen des GA zu berechnen.

6.1.3 Parallele Fitnessevaluierung (PFE)

Wie bereits beschrieben wurde, berechnet das Modul zur Fitnessevaluierung alle Nachkommen nacheinander. Die Fitness jedes einzelnen Individuums einer Population kann jedoch unabhängig von den anderen Individuen berechnet werden. Da der EPC eine reine Hardwarelösung ist, ist eine parallele Evaluierung der Fitness (Parallel Fitness Evaluation - PFE) möglich und naheliegend (Abbildung 86). Dadurch kann der Prozess der Evaluierung jeder Generation weiter beschleunigt werden. Im Extremfall könnte für jeden Nachkommen ein eigenes Fitnessevaluierungsmodul eingesetzt werden. Dann wären λ Hardware Module für die Fitnessevaluierung notwendig. Die Gesamtgröße des EPC-Systems steigt jedoch bereits um 13,7 bzw. 34,2%, wenn $2 \times$ oder $4 \times$ PFE eingesetzt wird. Die Hardwarekosten der PFE sind also nicht zu vernachlässigen. Es ist zu beachten, dass die Gesamtrechenzeit nicht von T_g auf T_g / λ sinkt, wie auf den ersten Blick vermutet werden könnte. Dafür gibt es zwei Gründe:

Erstens ist die Zeit zum Rehashing und der Rekonfiguration der Speicher im Datenpfad T_{reconf} konstant und ist durch Maßnahmen bei der Fitnessevaluierung nicht zu reduzieren.

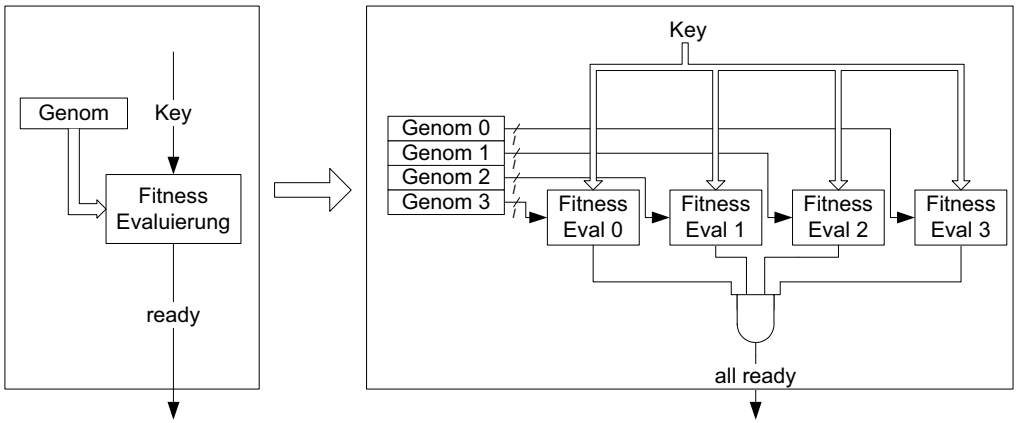


Abbildung 86 - Originale Fitnessevaluierung gegenüber PFE. Bevor ein neuer Schlüssel evaluiert werden kann, müssen alle Evaluierungsmodule die Berechnungen für den vorherigen Schlüssel abgeschlossen haben.

Zweitens basiert die Fitnessevaluierung auf dem virtuellen Hashen aller Schlüssel. Diese werden einem der beiden Speicher im Datenpfad entnommen (siehe Abbildung 64). Das Hashing läuft wie folgt ab: Nachdem ein Schlüssel zu allen Evaluierungsmodulen transferiert wurde, kann er gehasht werden. Kollisionen werden aufgelöst. Am Ende wird der Speichereintrag vorgenommen. Erst wenn der Schlüssel von allen parallelen Modulen verarbeitet wurde, kann der nächste Schlüssel übernommen und verarbeitet werden. Daraus folgt, dass die zur Evaluierung eines Schlüssels benötigte Zeit immer von dem Individuum mit den meisten Kollisionen bestimmt wird. Die Gesamtzeit der Berechnung einer Generation ergibt sich somit zu:

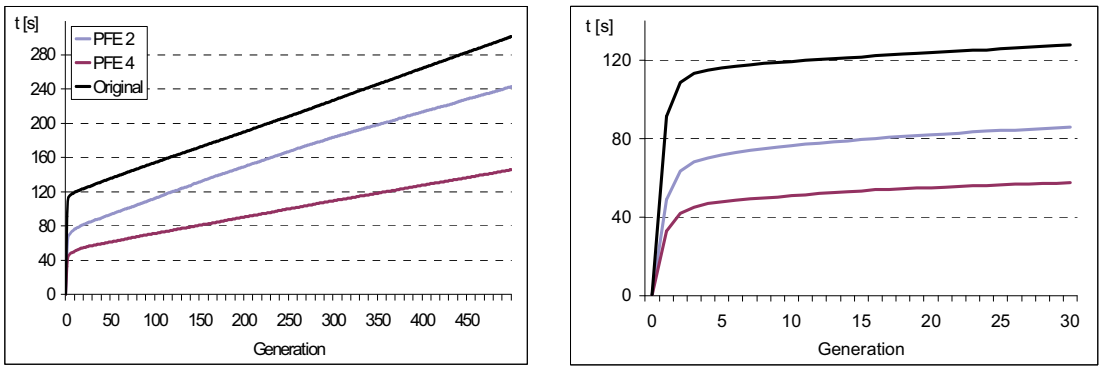
$$T_g = \sum_{i=1}^{|S|} \max_{j=1:\lambda}(T_i)_j + T_{reconf} \tag{32}$$

Es ist zu erwarten, dass der Grad der Verbesserung sich im Laufe des evolutionären Prozesses an T_g / λ annähert. Wie weit die Annährung verläuft, hängt von der Fitness jedes einzelnen Individuums der Nachkommenschaft ab:

$$\lim_{coll \to 0} \left[\max_{j=1:\lambda} \left(T_{i,j} \right) \right] = \min_{j=1:\lambda} \left(T_{i,j} \right) \tag{33}$$

Die Einführung eines Puffers für die ankommenden Schlüssel am Eingang eines jeden Evaluierungsmoduls trägt zur Entspannung der Situation bei. So kann jedes Modul eigenständig und von den anderen Modulen unabhängig auf die notwendigen Schlüssel zugreifen und diese verarbeiten. Es kommt in diesem Fall nur dann zur Notwendigkeit, auf andere Module zu warten, wenn der eigene Puffer leer, jedoch der mindestens eines anderen Moduls voll ist.

Das Verhalten der Verbesserungen in der Simulation kann Abbildung 87 entnommen werden. Sowohl für $2\times$ als auch $4\times$PFE kann eine Verbesserung festgestellt werden. Es ist festzustellen, dass die Evaluierungszeit um 20 bzw. 52% abnimmt. Das sind sehr gute Resultate, die jedoch mit großem zusätzlichem Hardwareaufwand erkauft werden. Beim Einsatz von $2\times$PFE entstehen zusätzliche Hardwarekosten von 830 Slices und 5 BRAMs. Das entspricht knapp 14% bzw. knapp 17% Zuwachs. Soll $4\times$PFE zum eingesetzt werden steigen die Hardwarekosten um 2100 Slices (34%) und 15 BRAMs (50%).



a) Zeitverlauf über 500 Generationen

b) Zeitverlauf der ersten 30 Generationen

Abbildung 87 - Einfluss der parallelen Fitnessevaluierung auf den Zeitverlauf des GA.

Es ist festzuhalten, dass, verglichen mit den Kosten, der Geschwindigkeitszuwachs unbefriedigend ist. Das gilt zumindest solange, wie PFE nicht mit MI kombiniert wird.

6.1.4 Kombination der Maßnahmen

Alle bislang untersuchten Maßnahmen zur Beschleunigung der Fitnessevaluierung (ET, MI, PFE) sind voneinander unabhängig. Es ist deshalb unproblematisch, diese Ansätze miteinander zu kombinieren. Der positive Einfluss von ET wird abnehmen, wenn der Ansatz mit der PFE kombiniert wird. Im Extremfall, wenn λ parallele Module Verwendung finden, kann es überhaupt keine Verbesserung durch ET geben. Die Nutzung von MI beeinflusst die Effektivität von PFE jedoch sehr positiv. Denn mit dem Einsatz von MI gilt für PFE, dass in weniger Fällen ein Modul auf ein anderes zu warten hat. Es gilt nun:

$$\lim_{coll_s \to p} \left[\max_{j=1:\lambda} \left(T_{i,j} \right) \right] = \min_{j=1:\lambda} \left(T_{i,j} \right) \tag{34}$$

Damit sinkt die Berechungszeit wesentlich früher auf T_g / λ. Auf der Zielplattform wurde eine Kombination von $8\times$MI, $2\times$FPE und ET implementiert. Aufgrund der Größenbeschränkungen der Hardwareplattform war eine Implementierung von $4\times$PFE nicht möglich. Die Hardwarekosten summieren sich auf 7494 Slices und 35 BRAMs. Auf der anderen

Seite reduziert sich die Zeit, um 500 Generationen zu evaluieren, von 310 Sekunden auf nur noch 67 Sekunden.

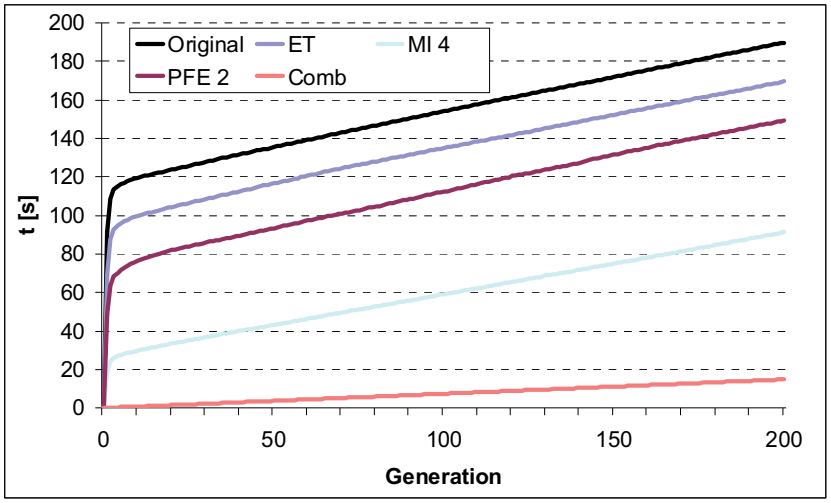


Abbildung 88 - Notwendige Zeit, um 500 Generationen mit einem (32,64)-GA und 32k Schlüsseln zu evaluieren. Die eindeutig beste Leistung erzielt die Kombination aller Ansätze.

Es ist zu konstatieren, dass alle drei vorgeschlagenen Maßnahmen zur Steigerung der Evolutionsgeschwindigkeit Wirkung zeigen. Insbesondere die Kombination der Maßnahmen ist sehr wirkungsvoll. Beim Nutzen jeder einzelnen Maßnahme ist jedoch der entstehende Hardwareaufwand zu beachten. MI ist ohne jede Einschränkung zu empfehlen. Bei einem maximalen zusätzlichen Aufwand von 4.6% steigert sich die Evaluierungsgeschwindigkeit im Vergleich immens. Das gleiche gilt für ET. Hier ist allerdings zu beachten, dass ein Einsatz nur dann sinnvoll sein kann, wenn der GA eine Umweltselektion verwendet. Wenn $\lambda = \mu$ gilt, ist der Einsatz von ET sinnlos. In jedem anderen Fall ist der Einsatz von ET sinnvoll. Zwar fällt der Gewinn nur mäßig aus, jedoch sind die Kosten mit nur 77 Slices vernachlässigbar. Die positive Beeinflussung von PFE durch MI wird deutlich, wenn man die erzielten Ergebnisse für eine Kombination aller Maßnahmen betrachtet. Mit insgesamt nur noch 67 Sekunden benötigt die Kombination aus $8\times$ MI, $2\times$ FPE und ET weitaus weniger Zeit, als die Summe der Maßnahmen vermuten ließe. Das spiegelt sich auch in dem Produkt aus $Slices \cdot Zeit$ mit einem Wert von nur 50 wieder. Die PFE hat im Vergleich zu den anderen Methoden große zusätzliche Hardwarekosten. Das ist durch die Implementierung paralleler Module zur Fitnessevaluierung leicht nachzuvollziehen. Dem stehen jedoch große Performancesteigerungen gegenüber. So vergrößert sich das Design zwar um mehr als 2000 Slices, bzw. um ein Drittel. Die Evaluierungszeit halbiert sich jedoch. Betrachtet man das Produkt aus $Slices \cdot Zeit$, zeigt $4\times$ PFE sehr gute Ergebnisse. Es ist beim Einsatz der PFE deshalb abzuwägen, inwieweit Ressourcen zur Verfügung stehen, um $2\times$ FPE oder gar $4\times$ FPE zu implementieren.

Tabelle 12 - Überblick über das Verhältnis von Hardwarekosten zu Evaluierungsbeschleunigung der verschiedenen Maßnahmen.

Modul	Ori-ginal	ET	PFE		MI			Kombi-nation
			2-fach	4-fach	2-fach	4-fach	8-fach	
Slices	6058	6135	6889	8130	6266	6259	6335	7494
Zuwachs (Abs.)	-	77	831	2072	208	201	277	1436
Zuwachs [%]	-	0,82	13,72	34,20	3,43	3,32	4,57	23,70
BRAMs	30	30	35	45	30	30	30	35
Zuwachs [%]	-	0	16,67	50,00	0	0	0	16,67
Zeit [s] (500 Generationen)	301	276	242	145	223	189	159	67
Abnahme [%]	-	8,3	19,6	51,8	25,9	37,2	47,2	77,7
$Slices \cdot Zeit$	182	169	167	118	140	118	101	50

Abbildung 89 verdeutlicht noch einmal die Bedeutung einer beschleunigten Fitnessevaluierung. Die Rechenzeit des Algorithmus steigt mit einer zunehmenden Anzahl von Schlüsseln stark an. Sollen 2000 Generationen evaluiert werden, benötigt der Algorithmus für 32k Schlüssel 590 Sekunden. Für 64k Schlüssel sind es schon 700 Sekunden. Bei einer Evaluierung von 128k Schlüsseln sind sogar 1570 Sekunden notwendig. Diese Zeiten wurden bereits mit einem optimierten Algorithmus gemessen. Bei der Datenaufnahme wurde bereits 8xMI verwendet, was bei 32k Schlüsseln schon eine Beschleunigung von 47,2% bewirkte.

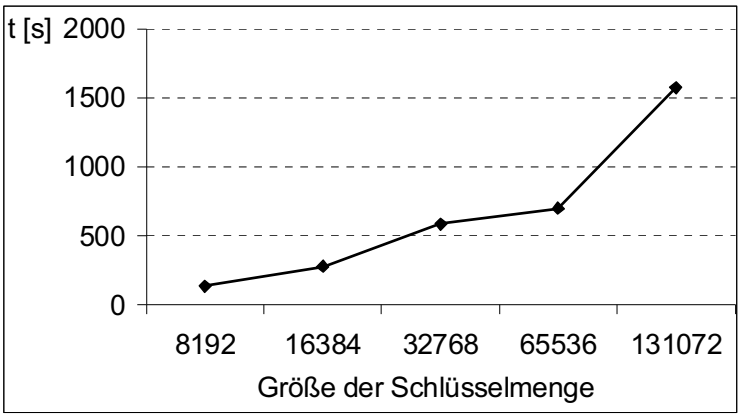


Abbildung 89 - Rechenzeit des 3-Mux-XOR Algorithmus zur Berechnung von 2000 Generationen des GA für unterschiedlich große Schlüsselmengen. Dabei ist bereits 8× MI implementiert.

Auch mit der Beschleunigung der Fitnessevaluierung existiert eine Zeitspanne zu Beginn der Ausführung des GA, in der die Leistungsfähigkeit der Hashfunktion sehr gering ist. Für den Einsatz in einem realen Klassifizierer würde dies bedeuten, dass es zu erheblichen Paketverlusten käme. Es ist daher sinnvoll, in einer Boot-Phase den EPC erst eine gewisse Zeit offline zu betreiben, damit sich ein gutes Genom entwickelt, bevor das System in Betrieb geht. Da die dafür notwendige Zeit nicht von vornherein bekannt ist, sollte bei Überschreiten einer akzeptablen Fitnessschwelle die Arbeitsbereitschaft des EPC signalisiert werden. Die Experimente mit dem Hardwareprototyp haben gezeigt, dass dies nach wenigen Minuten der Fall ist.

6.2 Parameter des Genetischen Algorithmus

Bei dem im EPC implementierten GA sind drei wichtige Parameter für dessen Leistungsfähigkeit unabhängig von der gewählten Hasharchitektur entscheidend:

- die gewählte Populationsgröße
- der Selektionsdruck
- die Mutationsrate

Im Folgenden sind die Ergebnisse der Untersuchung der Einflüsse dieser Parameter auf die erreichte Fitness des GA dargestellt. Alle Parameter wurden unabhängig voneinander untersucht. Das heißt, während der zu untersuchende Parameter verändert wurde, blieben alle anderen Parameter konstant bei den in Tabelle 13 angebenden Werten.

Tabelle 13 - Parameter des GA für die durchgeführten Experimente

Populationsgröße	32
Selektionsschema	Wettkampf (k = 4)
Mutationsrate	1/100
Hashfunktion	3-Mux-XOR
Schlüsseldaten	BGP-Routingtabelle (32.768 Einträge)
Abbruchkriterium	2000 Generationen
Anzahl der Läufe	20

6.2.1 Populationsgröße

Durch die Veränderung der Populationsgröße kann Einfluss auf die Breite der Suche genommen werden, um den Lösungsraum zu durchsuchen. Als naiven Ansatz könnte man annehmen, dass eine möglichst große Populationsgröße analog zur Natur am ehesten in der Lage ist, eine optimale Problemlösung zu finden. Es muss allerdings beachtet werden, dass die Zeit zur Evaluierung der Fitness einer Generation linear von der Populationsgröße abhängt. Somit können in einer definierten Zeit bei doppelter Populationsgröße nur halb so viele

Generationen des genetischen Algorithmus evaluiert werden. Das kann unter Umständen zu schlechteren Fitnesswerten führen. Die Gesamtleistung des GA hängt allerdings nicht ausschließlich von der Anzahl der Individuen ab. Der Einfluss der Populationsgröße ist deshalb weniger groß. Das spricht für die Wahl einer verhältnismäßig kleinen Population. Es ist deshalb notwendig zu untersuchen, welche Populationsgröße für die Lösung des Pakteklassifizierungsproblems am besten geeignet ist. Dazu wurden verschiedene Populationen mit einer Größe zwischen 8 (P8) und 128 (P128) Individuen untersucht und die Entwicklung der jeweiligen Fitness über einen definierten Zeitraum festgehalten. Es wurden keine Maßnahmen zur Beschleunigung der Evaluierung ergriffen, wie sie in Abschnitt 6.1 dargestellt sind. Aus Abbildung 90 und Abbildung 91 geht der Einfluss der Populationsgröße auf die Performance der Hashfunktion hervor.

Wie Abbildung 90 entnommen werden kann, verbessert der Einsatz einer möglichst großen Population die Fitnessentwicklung der einzelnen Generationen. Je größer die Population ist, desto weniger Generationen sind notwendig, um einen sehr guten Fitnesswert zu erreichen. Des Weiteren ist die maximal erreichte Fitness ebenfalls besser, je größer die Anzahl der Individuen in der Population ist. Allerdings nimmt auch die notwendige Rechenzeit zu, um eine Generation zu berechnen.

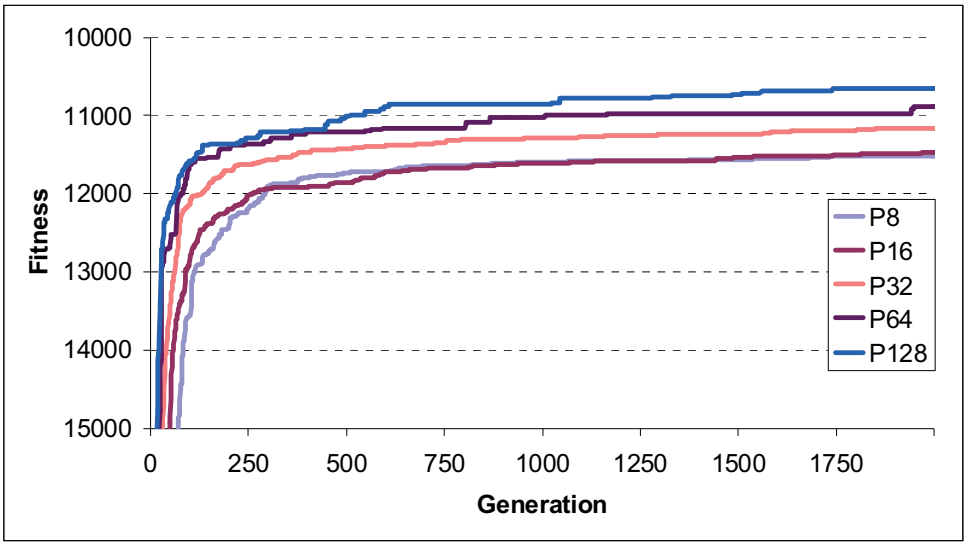


Abbildung 90 - Entwicklung der Fitness des Genetischen Algorithmus über 2000 Generationen. Es fanden Populationsgrößen zwischen 8 und 128 Individuen Verwendung.

Abbildung 91 verdeutlicht diesen Zusammenhang. Die Rechenzeit zum Erreichen eines akzeptablen Fitnesswerts steigt mit zunehmender Populationsgröße stark an. Die evolutionäre Entwicklung von P8 und P16 hat bei 200 Sekunden bereits weitgehend konvergiert, während P128 noch nicht einmal die 2. Entwicklungsgeneration hervorgebracht hat.

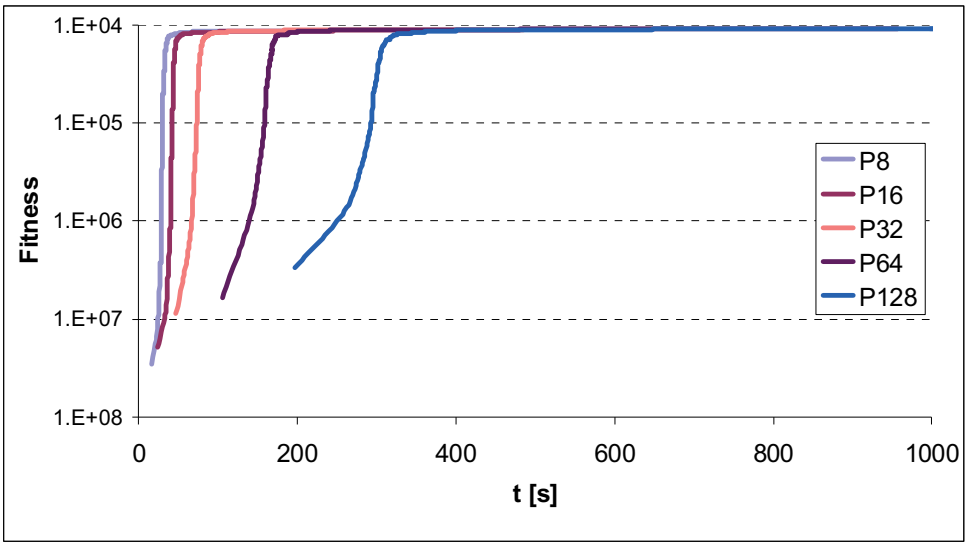


Abbildung 91 - Zeitlicher Verlauf der Entwicklung der Fitnesswerte der Hashfunktion bei Nutzung unterschiedlicher Populationsgrößen.

Abbildung 92 zeigt, dass bei einer Populationsgröße von 8 bis 32 bei 150 Sekunden bereits ausgezeichnete Fitnesswerte erreicht werden. P64 und P128 haben nur schlechte bzw. keine Fitnesswerte erreicht. Sollen 128 Individuen evaluiert werden, ist erst nach 400 Sekunden ein guter Wert erreicht. Allerdings sind die maximal zu erreichenden Fitnesswerte mit zunehmender Populationsgröße etwas besser. Eine Populationsgröße von mehr als 32 erscheint nicht sinnvoll, da die erreichten Gesamtergebnisse nach 2000 Generationen, was einer Laufzeit von fast 1000 bzw. 2000 Sekunden für P64 bzw. P128 entspricht, im Mittel nur leicht besser sind als die Ergebnisse von P32.

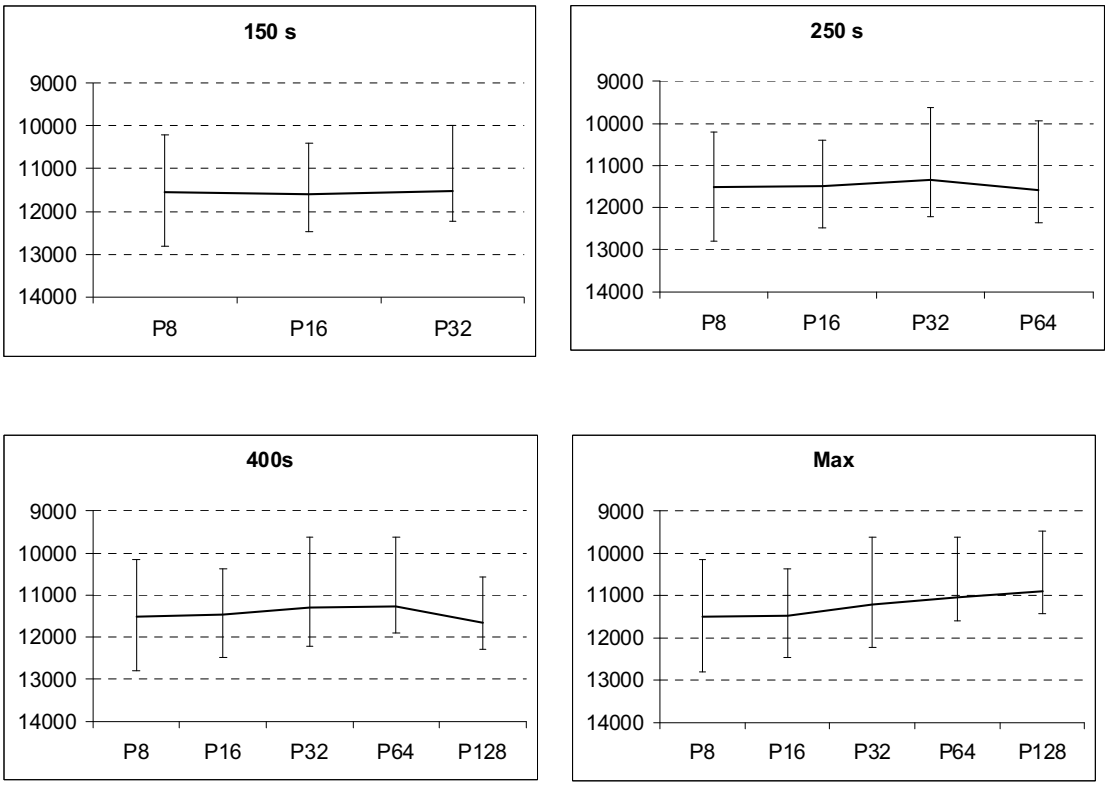


Abbildung 92 - Erreichte Fitness der Hashfunktion nach unterschiedlichen Zeiträumen bzw. dem Ablauf von 2000 Generationen.

6.2.2 Selektionsdruck

Der Selektionsdruck beschreibt die Schwierigkeit eines Individuums einer Generation, als ein Elter für die nächst Generation ausgewählt zu werden. Bei hohem Selektionsdruck bestehen tendenziell nur die fittesten Individuen einer Generation fort. Ist der Selektionsdruck niedrig, haben auch weniger fitte Individuen Chancen, ausgewählt zu werden. Die richtige Wahl des Selektionsdrucks ist ein wichtiger Faktor zur Optimierung der Effektivität des GA. Ist der Selektionsdruck zu gering, ist die Suche im Parameterraum nicht zielgerichtet. Sie gleicht dann zunehmend einer zufälligen Suche, was zur Folge hat, dass das optimale Genom im Parameterraum erst spät oder überhaupt nicht gefunden werden kann. Ist der Selektionsdruck auf der anderen Seite zu hoch, ist die Suche im Parameterraum zu eng fokussiert. Das hat zur Folge, dass der GA frühzeitig konvergiert und unter Umständen nur ein lokales und kein globales Optimum des Parameterraumes gefunden wird. In beiden Fällen ist das Ergebnis des GA suboptimal.

Der Selektionsdruck eines GA kann über die Eltern- oder die Umweltselektion gesteuert werden. Da bei klassischen GAs keine Umweltselektion verwendet wird, soll an dieser Stelle primär auf die Elternselektion eingegangen werden.

Wie bereits in Kapitel 2.3.1.1 beschrieben, wählt die Elternselektion aus der Population der aktuellen Generation die λ Nachkommen aus. Die implementierte Methode nutzt die Selektion mittels Wettkampfs. Dabei werden von den μ Eltern k zufällig ausgewählt. Die k gewählten Individuen bestreiten einen Wettkampf, den dasjenige gewinnt, das die beste Fitness aufweist. Die Wahl der Größe von k bestimmt hierbei, wie stark sich fittere Individuen gegenüber weniger fitten Individuen durchsetzen können. Ist k klein, beispielsweise 2, besteht die Chance, dass ein mittelmäßiger Kandidat auf einen noch schlechteren Kandidaten trifft. Er setzt sich folglich im Wettkampf durch und wird zum Nachkommen. Ist k groß, z.B. 10, ist die Wahrscheinlichkeit eher gering, dass ein mittelmäßiger Kandidat auf keine besseren Kandidaten im Wettkampffeld trifft. Große Wettkampfgrößen bevorzugen fittere Individuen. Sie erhöhen damit den ausgeübten Selektionsdruck.

In Abbildung 93 ist die mittlere Fitness der besten Individuen des GA dargestellt, wie sie nach einer Evaluierung von 2000 Generationen erreicht wurde.

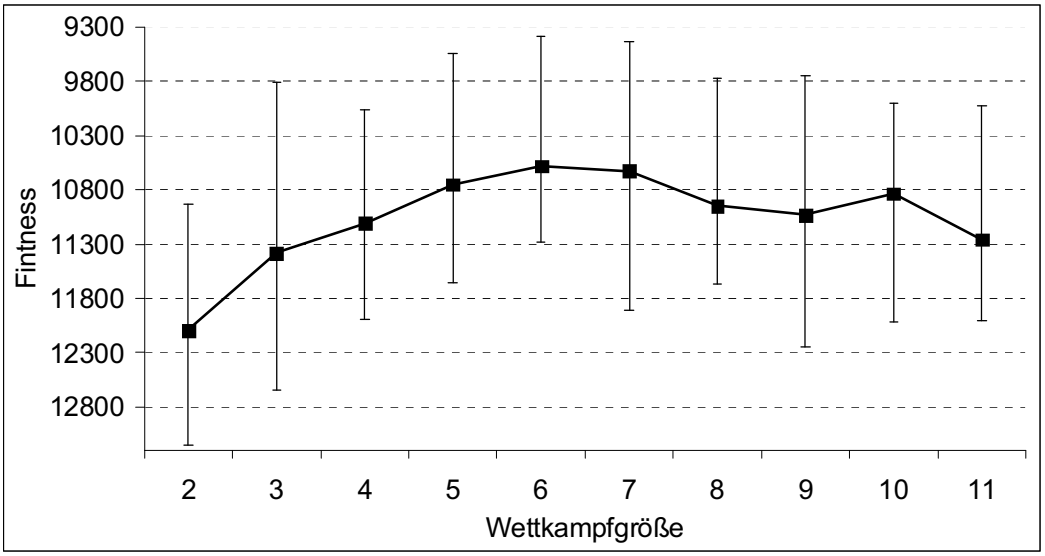


Abbildung 93 - Mittlere Fitness der besten Individuen des GA nach Ablauf von 2000 Generationen. Es wurden für jede Wettkampfgröße 20 Untersuchungen vorgenommen.

Dazu wurde die 3-Mux-XOR Architektur verwendet. Aus der Abbildung geht hervor, dass vor allem eine Wettkampfgröße von 2 einen zu geringen Selektionsdruck erzeugt. Die Fitnesswerte sind deutlich schlechter, als die Fitnesswerte der größeren Wettkampfgrößen. Die besten Ergebnisse wurden mit einer Wettkampfgröße von $k = 6$ erzielt, wobei festzustellen ist, dass die mittleren Ergebnisse von $k = 5$ und $k = 7$ nur sehr leicht abweichen. Insgesamt sind

die Abweichungen von $k = 3$ bis $k = 11$ nicht sehr groß, vor allem wenn man die verhältnismäßig große Schwankungsbreite berücksichtigt, die bei den ermittelten Ergebnissen aufgetreten ist. Eine Wettkampfgröße von $k = 6$ hat mit 10581 Kollisionen im Mittel nur 1% bessere Ergebnisse als $k = 7$, ist aber 7,6% bzw. 6,4% besser als $k = 3$ und $k = 11$. Die Wahl muss deshalb auf eine Wettkampfgröße im Bereich von 5 bis 7 fallen, um optimale Ergebnisse zu erzielen.

6.2.3 Mutationsrate

Ein wichtiger Parameter des genetischen Algorithmus ist die Mutationsrate. Das ist der Fall, weil bei allen GAs ausschließlich durch die Mutation des Genoms neues genetisches Material in die Population eingeführt wird. Nur so ist es möglich, Kandidaten zu erzeugen, die potentiell jede Stelle des Suchraumes abdecken können. Von der Mutationsrate hängt es nun ab, wie schnell neues genetisches Material eingeführt wird. Das beeinflusst den Charakter der Suche maßgeblich. Ist die Mutationsrate zu gering gewählt, ist die Population nicht in der Lage, ein lokales Minimum, das einmal gefunden wurde, wieder zu verlassen. Ist sie jedoch zu groß, degeneriert die Suche zu einer zufälligen.

An dieser Stelle soll untersucht werden, inwiefern sich verschiedene Mutationsstrategien auf die Leistungsfähigkeit des GA und damit auch auf die Qualität des EPC auswirken. Es wird sowohl der Einfluss auf die Geschwindigkeit der Evolution als auch die Entwicklung der absoluten Fitnesswerte betrachtet. Wie in Abschnitt 5.3 bereits erläutert wurde, ist die für die bisher gemachten Untersuchungen verwendete Mutationsrate 0,01. Diese wurde von Grefenstette in [Gre86] vorgeschlagen. Der Wert ist das Ergebnis von Experimenten, durchgeführt mit fünf verschiedenen genetischen Algorithmen für unterschiedliche mathematische Problemstellungen. In den folgenden Abschnitten sollen unterschiedliche Mutationsstrategien untersucht werden. Dies umfasst verschiedene konstante aber auch variable Mutationsraten. In [Och00] sind einige dieser Ansätze beschrieben und zusammengefasst. Auch wurden sie für unterschiedliche mathematische Probleme auf ihre Effektivität hin untersucht. Es soll untersucht werden, welche Strategie für den EPC optimal ist.

6.2.3.1 Konstante Mutationsraten

Weite Verbreitung hat eine konstante Mutationsrate gefunden, die gemäß $1/l$ umgekehrt proportional zur Genomlänge *l* ist. Sie wurde bereits in [BRS66] vorgeschlagen und taucht in unterschiedlichen Quellen auf [HM91, Mue92]. Aus diesem Grund soll sie an dieser Stelle untersucht werden. Auch andere Raten, die von $1/l$ abgeleitet und größer sind, wurden untersucht. Es wurde untersucht, ob sich erhöhte Raten positiv auf die Leistung des GA auswirken. Dazu wurden Raten von r/l mit $r \in \{1,2,4,6,8\}$ untersucht. Teil dieser Untersuchung ist auch die Einbeziehung von Mutationsraten, die nicht von der Genomlänge

abhängen. DeJong [DeJ75] schlug die Wahl einer sehr geringen Mutationsrate von 0,001 vor. Schaffer schlägt in [SCE+89] vor, die Mutationsrate von der Anzahl der Individuen der Population abhängig zu machen:

$$\frac{1{,}75}{\mu \cdot \sqrt{l}} \tag{35}$$

Damit liegt die Mutationsrate in einem Bereich zwischen 0,005 und 0,01. Diese wurde von Schaffer als günstig für viele Problemstellungen bewertet. Die Untersuchungen wurden für Schlüssel mit einer Anzahl von 8.192 und 32.768 durchgeführt. Das bedingt Genomlängen zwischen $l = 140$ und $l = 160$.

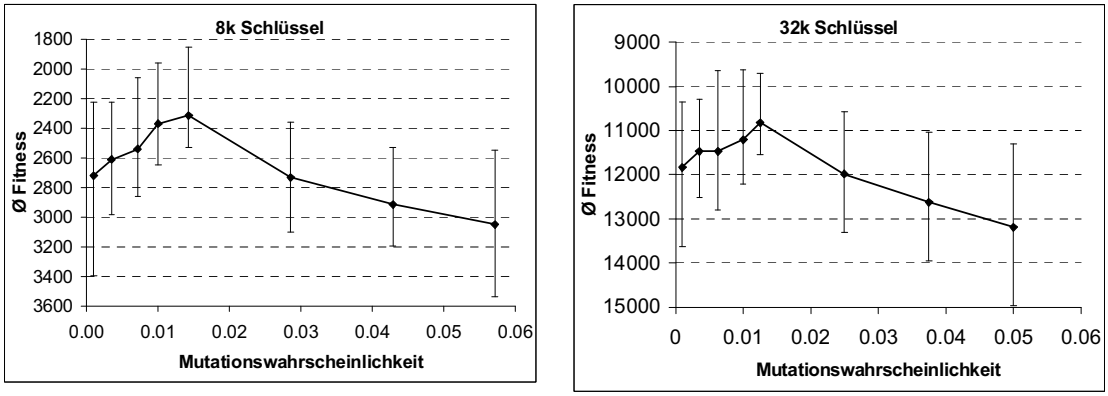


Abbildung 94 - Einfluss unterschiedlicher konstanter Mutationsraten auf die Performance der evolvierbaren Hashfunktion.

Wie Abbildung 94 entnommen werden kann, ist es wichtig, die richtige Mutationsrate zu wählen, um optimale Ergebnisse zu erzielen. Diese liegt für die aktuelle Problemstellung sowohl für 8k als auch für 32k Schlüssel bei $2/l$. Das entspricht einer Mutationsrate von 0,0134 ($|S| = 8.192$) bzw. 0,0125 ($|S| = 32.768$). Bereits sehr geringe Abweichungen in der absoluten Mutationsrate beeinflussen die Performance negativ. Bemerkenswert ist der starke Abfall der Performance mit zunehmender Mutationsrate. Wird eine Mutationsrate von 0.05, also $8/l$, gewählt, fällt die mittlere Fitness für 32k Schlüssel von 10.612 auf 12.956 zurück. Das entspricht einer Verschlechterung von 18%. Bei einer Schlüsselmenge von 8k beläuft sich der Fitnessunterschied zwischen $2/l$ und $8/l$ sogar auf 24%. Offensichtlich können optimale Konfigurationen nicht gefunden werden, wenn die Mutationsrate zu groß gewählt wird, wobei bereits eine Rate von 0,05 als groß zu betrachten ist.

6.2.3.2 Variable Mutationsraten

Es ist auch denkbar, eine variable Mutationsrate einzusetzen, um sich an die aktuelle Entwicklung der Fitness anzupassen, damit eine möglichst schnelle und im Endergebnis gute Evolution erreicht werden kann. In der Literatur gibt es verschiedene Vorschläge, die Mutationsrate variabel zu gestalten.

Schaffer (Schaf)

Wie bereits oben erwähnt, ermittelten Schaffer et. al. in [SCE+89], dass gute Mutationsraten im Bereich von 0.005 und 0.01 liegen. Um diese umzusetzen, wurde untersucht, inwieweit eine zufällige Auswahl einer Mutationsrate aus diesem Bereich zu guten Ergebnissen führen kann. Dazu wurde in Experimenten die Mutationsrate in den Grenzen von [0.005, 0.013] mittels eines LFSR zufällig ausgewählt. Damit liegt die Rate in dem Bereich, der für konstante Mutationsraten als optimal ermittelt wurde.

Bäck

Bäck schlägt in [Bae92] vor, die aktuelle Fitness *f* mit in die Berechnung der Mutationsrate einzubeziehen:

$$\frac{1}{2 \cdot (f+1) - l} \tag{36}$$

Dieser Ansatz erwies sich allerdings als wenig praktikabel für die aktuelle Problemstellung, da die Fitness invers verwendet wird. Das heißt, die Fitness der Population steigert sich, wenn der Fitnesswert sinkt. Gravierender ist jedoch die Tatsache, dass für die Fitness Startwerte von 100.000.000 nicht ungewöhnlich sind, während der finale Fitnesswert um die 11.000 liegt. Das würde zu unsinnigen Werten für die Mutationswahrscheinlichkeit führen und große Änderungen für die Umsetzung der Formel erfordern. Aus diesem Grund wurde von der Implementierung dieser Mutationsrate abgesehen.

Bäck/Schütz (BS)

Eine Berechnung der Mutationsrate, die von der aktuellen Generation abhängt, schlagen Bäck und Schütz in [BS96] vor. Hierbei berechnet sich die Mutationswahrscheinlichkeit aus:

$$\frac{1}{2 + \frac{l-2}{actual_generation - 1} \cdot generation} \tag{37}$$

Stagnation (Stag_1, Stag_2)

Eine weitere Möglichkeit, die Fitness mit einfachen Mitteln variabel zu gestalten, kann die Verwendung von zwei verschiedenen Mutationswahrscheinlichkeiten sein. Als sehr gut haben

sich $1/l$ und $2/l$ erwiesen. Stagniert dann der Fitnesswert bei seiner Entwicklung, ist im günstigsten Fall das globale Optimum erreicht. Wahrscheinlicher ist jedoch die Konvergenz der Population bei einem lokalen Optimum (siehe Kapitel 2.3.1.3), das mit sehr kleinen Mutationswahrscheinlichkeiten nicht verlassen werden kann. Um ein Verlassen des lokalen Optimums zu ermöglichen, wird die Mutationsrate in diesem Fall vergrößert. In den untersuchten Fällen wird von Konvergenz ausgegangen, wenn sich die Fitness in den letzten 100 Generationen nicht verbessern konnte. Dann erhöht sich die Mutationswahrscheinlichkeit auf $16/l$. Sobald jedoch wieder eine Verbesserung zu verzeichnen ist, senkt der Algorithmus die Mutationswahrscheinlichkeit wieder auf $1/l$ bzw. $2/l$. Eine Weiterentwicklung der zweistufigen Mutationswahrscheinlichkeit stellt der lineare Anstieg dar. Hierbei wird die Mutationswahrscheinlichkeit bei Konvergenz solange mit jeder Generation linear vergrößert, bis ein Maximalwert von $16/l$ erreicht wird. Sie wird wieder zurück gesetzt, wenn ein besserer Fitnesswert gefunden wird. Die Folge dieser Maßnahme ist, dass bei Konvergenz allmählich zu einer zufälligen Lösungssuche im Suchraum übergegangen wird.

Vergleich

Der Vergleich der publizierten und vorgeschlagenen Mutationsraten mit der besten konstanten Mutationsrate von 2/l ist in Abbildung 95 dargestellt. Es ist zu erkennen, dass der von Bäck/Schütz vorgeschlagene Algorithmus im Mittel die besten Ergebnisse erzielt. Diese übertreffen auch leicht die Ergebnisse der besten konstanten Mutationsrate von 2/l. Der mittlere Fitnesswert sinkt bei 8k Schlüsseln von 2313 auf 2286. Bei 32k Schlüsseln ist eine Verbesserung von 10822 auf 10708 zu verzeichnen.

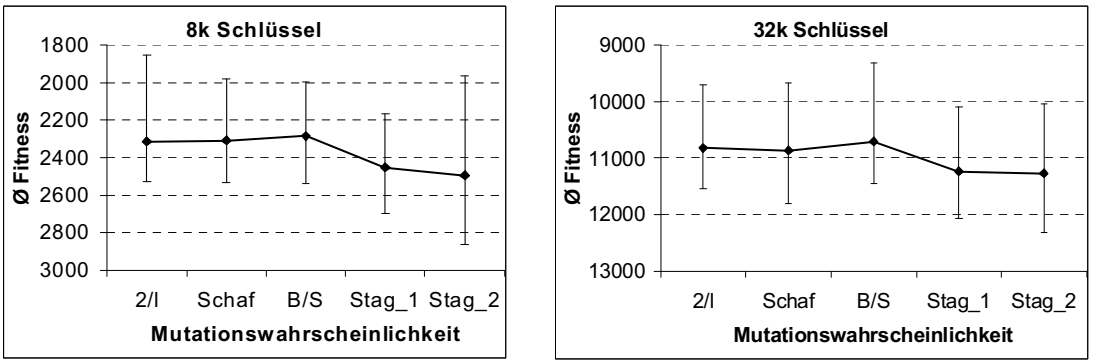


Abbildung 95 - Vergleich der variablen Mutationsraten mit der besten konstanten Rate von $2/l$.

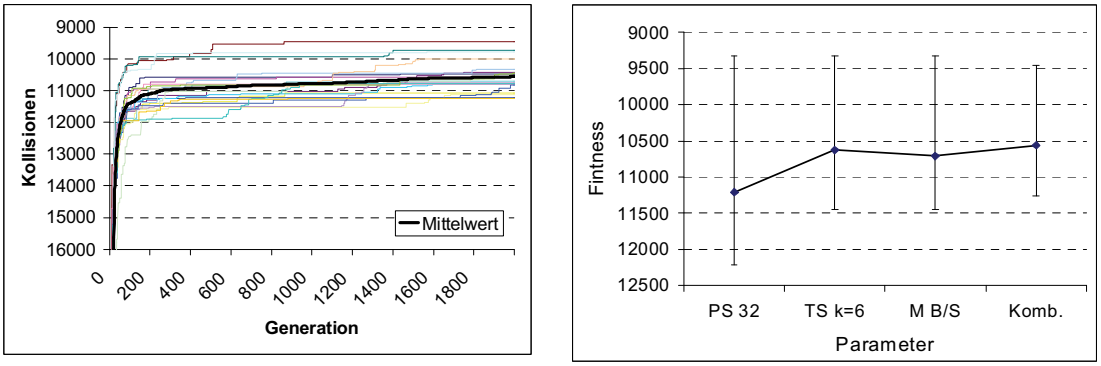
Dabei war bei 20 durchgeführten Testläufen in 13 bzw. 14 Fällen die Fitness des B/S Algorithmus leicht besser. Die anderen Algorithmen bleiben in ihren Ergebnissen hinter B/S

und $2/l$ zurück. Insbesondere die Algorithmen, die auf Stagnation in der Performance mit einer Erhöhung der Mutationsrate reagieren sollten, zeigten keine Verbesserungen mehr, nachdem die Mutationsrate vergrößert wurde. In keinem Fall konnte mit einer höheren Mutationsrate ein besserer Fitnesswert gefunden werden. Es ist demzufolge nicht sinnvoll, die Mutationsrate bei einer Stagnation zu steigern.

6.2.4 Kombination der Maßnahmen

Dem Autor ist bewusst, dass etwaige Wechselwirkungen zwischen den Parametern (Populationsgröße, Selektionsdruck, Mutationsrate) mit dieser Methode nicht untersucht und gefunden werden konnten. Demzufolge ist die Kombination der jeweils optimalen Parameter nicht zwangsläufig ein globales Optimum im Parameterraum. Allerdings ist die systematische Suche nach dem Optimum im in diesem Falle dreidimensionalen Parameterraum sehr aufwendig. Es soll hier angenommen werden, dass die Kombination der Einzelergebnisse eine hinreichend gute Annährung an die tatsächlich erreichbare Fitness mit diesem GA darstellt.

Abbildung 96 stellt die Performance des GA dar, wenn die jeweils beste Wahl der Parameter verwendet wird (Populationsgröße 32, Selektionsdruck $k = 6$, Mutationsrate nach Bäck/Schütz). Wie zu erkennen ist, wurden in Experimenten mit den kombinierten Parametern die besten Ergebnisse erzielt, obgleich zu konstatieren ist, dass das Optimum nur leicht besser ist als das auf $k = 6$ optimierte System. Die Kombination erreichte im Mittel eine Fitness von 10.560, während mit $k = 6$ 10.627 erreicht wurde.



a) Performanceentwicklung über die Generationen mit 20 unabhängigen Läufen

b) Vergleich der besten Parameter

Abbildung 96 - Eigenschaften des GA mit kombinierten Parametern und Vergleich mit den Einzelperformances.

Die Suche nach einer optimalen Parameterwahl könnte eine weitere interessante Anwendung für einen anderen evolutionären Algorithmus darstellen. Dieser hätte dann den mehrdimensionalen Parametersuchraum nach einer optimalen Parameterkombination zu

durchsuchen. Derartige Untersuchungen wurden allerdings im Rahmen dieser Arbeit nicht durchgeführt.

6.3 Smart Initialization

Eine Möglichkeit, die Berechnung der Fitness zu beschleunigen, ist der Start in die Entwicklung mit besonders vielversprechenden Genomen. Hierfür wurde eine Heuristik entwickelt, die aus der numerischen Verteilung der einzelnen Bitpositionen ein vielversprechendes Startgenom erzeugt [SWT06]. Die Smart Initialization (SI) basiert bei der Bildung des Startgenoms auf den Hashfunktionsarchitekturen, denen Multiplexer zu Grunde liegen (vorgestellt in den Kapiteln 5.2.1 -5.2.4). Bei der SI wird der Umstand ausgenutzt, dass Bitpositionen innerhalb der Schlüsselmenge, bei denen die Verteilung von '0' und '1' über alle Schlüssel möglichst ausgeglichen ist, potentiell nützlicher für eine große Diversität von Hashwerten sind, als solche Positionen, bei denen '0' und '1' sehr ungleich verteilt sind und somit potentiell weniger Information enthalten. Es wird folglich ein Individuum der initialen Population nicht zufällig, sondern mittels SI erzeugt.

Es gibt zwei Möglichkeiten der Realisierung der SI im EPC, Hardware- und Softwarerealisierung. Die Hardwarerealisierung ist wie folgt implementiert: Ein Initialisierungsmodul beinhaltet *N* Zähler, einen für jedes Bit des Schlüssels. Die Breite der Zähler beträgt $\log_2(|S|)+1$. Bei der Konfiguration der Schlüssel in den Speicher überprüft das Modul jeden Schlüssel Bit für Bit. Für alle Bitpositionen, die eine '1' enthalten, inkrementiert das Modul den entsprechenden Zähler. Für alle Bitpositionen die eine '0' enthalten, dekrementiert das Modul den entsprechenden Zähler. Nachdem alle Schlüssel verarbeitet sind, enthalten die Zähler jeweils die Differenz der Auftrethäufigkeiten von '1' und '0' an allen Bitpositionen. In einer zweiten Verarbeitungsstufe führt eine andere Zustandsmaschine des Moduls eine Sortierung (Bubble Sort) der Zähler aus. Während der Sortierung wird jeweils der Betrag der Zähler verwendet (Zweierkomplement, wenn der Wert negativ ist). Für das Genom wählt das Modul dann diejenigen Bitpositionen aus, bei denen die Verteilung von '0' und '1' am gleichmäßigsten ist – deren Zählerbeträge am kleinsten sind. Abhängig von der gewählten Hashfunktion sind unterschiedlich viele Positionen notwendig. Vorteil einer Hardwarelösung ist zweifellos die Einsetzbarkeit in jeder Umgebung, da SI in den EPC integriert ist. Ein entscheidender Nachteil sind jedoch die Hardwarekosten, die mit 7% der Gesamtkosten des EPC sehr hoch sind.

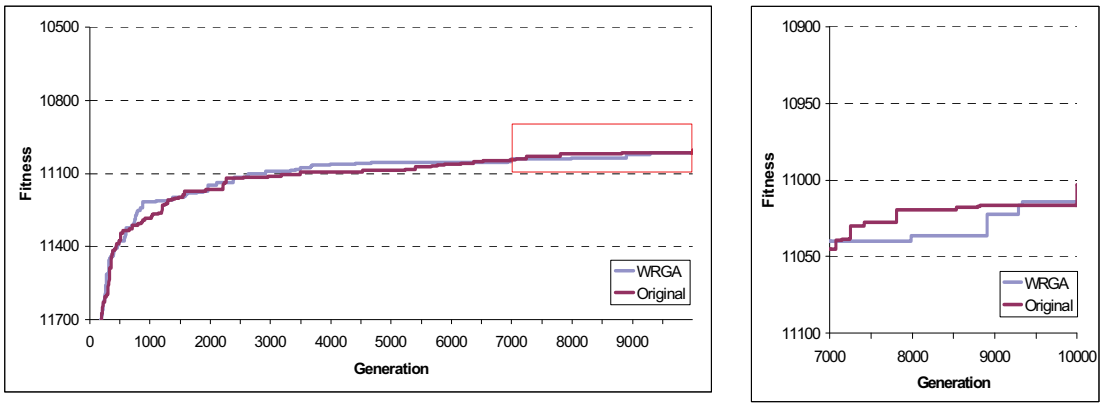
Realisiert man die SI in Software, so geschieht das extern. Die Schlüsselmenge ist zu analysieren und daraus das vielversprechendste Genom mittels des vorgestellten Algorithmus zu erzeugen. Die Hardware des EPC ist dann nur noch mit einer Schnittstelle auszustatten, die es erlaubt, das Startgenom von außen zu konfigurieren. Nachteil dieser Lösung ist, dass zur Erzeugung des Genoms ein externes System existieren muss, auf dem die SI implementiert ist.

SI sollte, wenn überhaupt, nur als Softwareversion eingesetzt werden, um die Fitness des EPC in der Startphase zu optimieren, da SI nur beim Systemstart eingesetzt wird. Untersuchungen haben gezeigt, dass SI die finale Performance nicht positiv beeinflusst [SWT06].

6.4 Wiederholt reinitialisierter GA

In der Zusammenfassung der Grundlagen genetischer Algorithmen wurde als eine Schwäche von GAs das verfrühte Konvergieren, also das Konvergieren in einem lokalen Optimum erläutert. Das Problem verschärft sich weiter, wenn der Selektionsdruck erhöht wird, wie in Abschnitt 6.2.2 untersucht wurde. Eine wichtige Eigenschaft des EPCs ist die, dass der GA im Gegensatz zu Softwarealgorithmen auf dem System als Hardwarekomponente stets zur Verfügung steht. Die Hardwaremodule würden nach Beendigung des GA bzw. nachdem dieser konvergiert ungenutzt beliben. Wie aus Abbildung 74 in Abschnitt 5.4 hervorgeht, erreicht der GA für ein und dieselbe Schlüsselmenge in einer großen Anzahl von Durchläufen unterschiedlich gute Ergebnisse. Die Fitnesswerte der 3-Mux-XOR Architektur weisen bei einer durchschnittlichen Fitness von 11.109 eine Standardabweichung von immerhin 498 auf. Es ist also nicht unwahrscheinlich, dass ein anderer Lauf des GA bessere Ergebnisse erzielen kann, als ein beliebiger aktueller Lauf. Anstatt einen GA unendlich weiter laufen zu lassen bzw. die Berechnung abzubrechen, nachdem keine Verbesserungen mehr erzielt werden, könnte ein wiederholt reinitialisierter GA (WRGA) bessere Ergebnisse erzielen, da nach vielen unabhängigen Durchläufen immer das beste Ergebnis für die Hashfunktion im Datenpfad verwendet werden kann. Auf der anderen Seite kann in der gleichen Zeit, in der 50 Läufe mit 2.000 Generationen durchgeführt wurden, der originale GA mindestens 100.000 Generationen optimieren.

Aus diesem Grund wurden mehrere Experimente durchgeführt, bei denen jeweils die gleichen Schlüsselmengen mittels WRGA und der klassischen Anwendung der GA evaluiert wurden. Die Ergebnisse einer Untersuchung über insgesamt 10.000 Generationen können Abbildung 97 entnommen werden. Wie zu erkennen ist, ist der originale GA über 10.000 Generationen im Mittel etwas besser als der WRGA, der alle 500 Generationen neu initialisiert wurde. Außerdem ist zu erkennen, dass sich die Fitness auch in hohen Generationen noch weiter verbessert.



a) Mittelwert über 20 Stichproben b) Vergrößerte Darstellung

Abbildung 97 - Verlauf der Fitness des originalen GA und von WRGA über 10.000 Generationen. Nach jeweils 500 Generationen erfolgte eine Neuinitialisierung des WRGA.

Dieser Umstand wird in Abbildung 98 noch deutlicher. Hier wurde die Entwicklung der Fitness über 100.000 Generationen betrachtet. Während die mittlere Fitness beim Originalalgorithmus nach 10.000 Generationen 10.840 betrug, steigerte sie sich bis zu Generation 100.000 noch auf 10.656. Auch bei dem langen Lauf ist festzustellen, dass WRGA keine Vorteile bringt. Die Ergebnisse, die mit dem originalen GA erreicht werden, sind besser als die durch WRGA erreichten. Offenbar ist es nicht sinnvoll, WRGA einzusetzen.

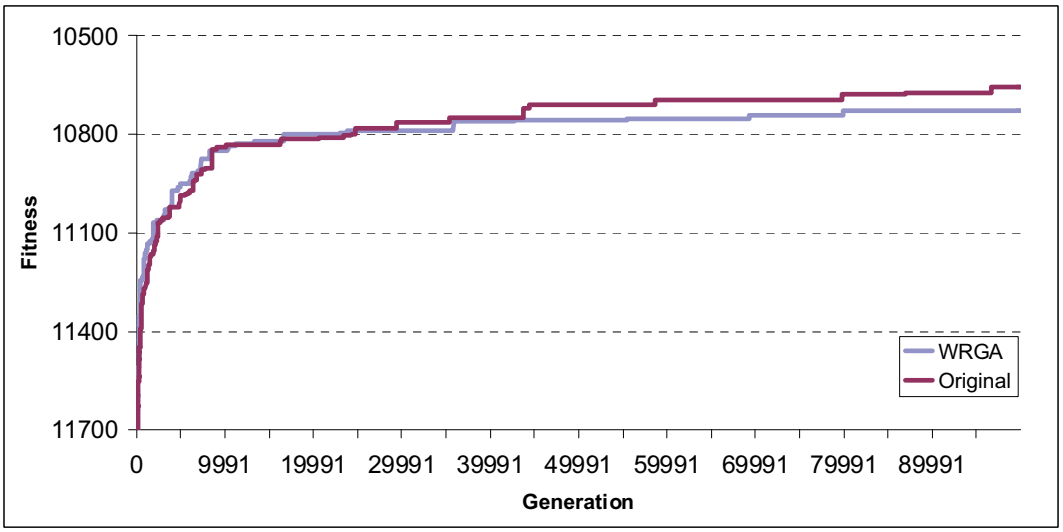


Abbildung 98 - Verlauf der Fitness des originalen GA und von WRGA über 100.000 Generationen. Nach jeweils 5.000 Generationen erfolgte eine Neuinitialisierung des WRGA.

Auch der Einsatz von WRGA, um auf veränderliche Schlüsselmengen zu reagieren, erscheint nicht sinnvoll. Um das Verhalten von WRGA unter diesen Umständen zu überprüfen, wurden

die Untersuchungen, die auch in Abschnitt 5.4.5 dargestellt sind, wiederholt. Der einzige Unterschied war die Reinitialisierung des GA, sobald sich die Schlüsselmenge verändert. Wie Abbildung 99 entnommen werden kann, wird die Qualität der Hashfunktion durch WRGA nicht positiv beeinflusst (vgl. mit Abbildung 79 aus Kapitel 5.4.5). Die Qualität der Hashfunktionen unterscheidet sich nicht maßgeblich voneinander.

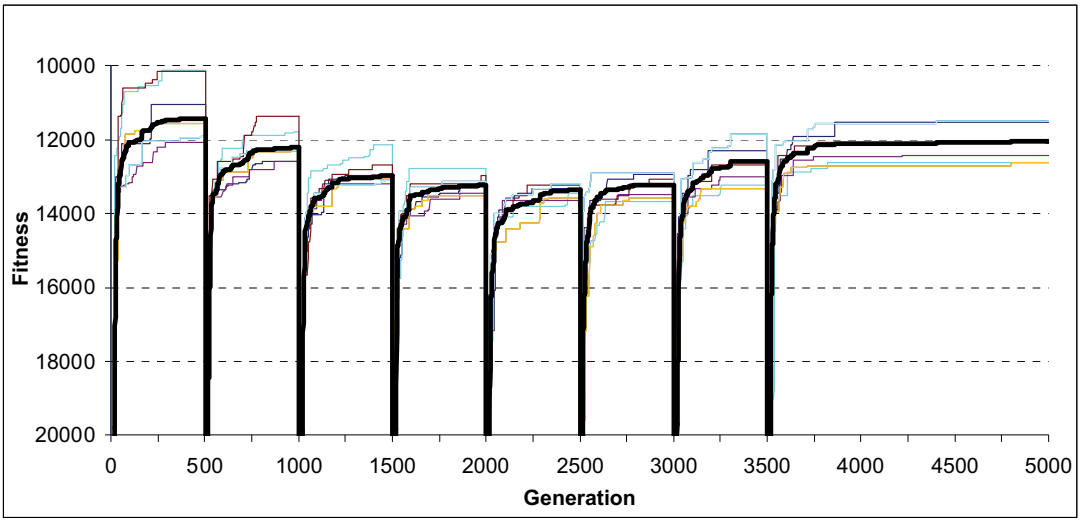
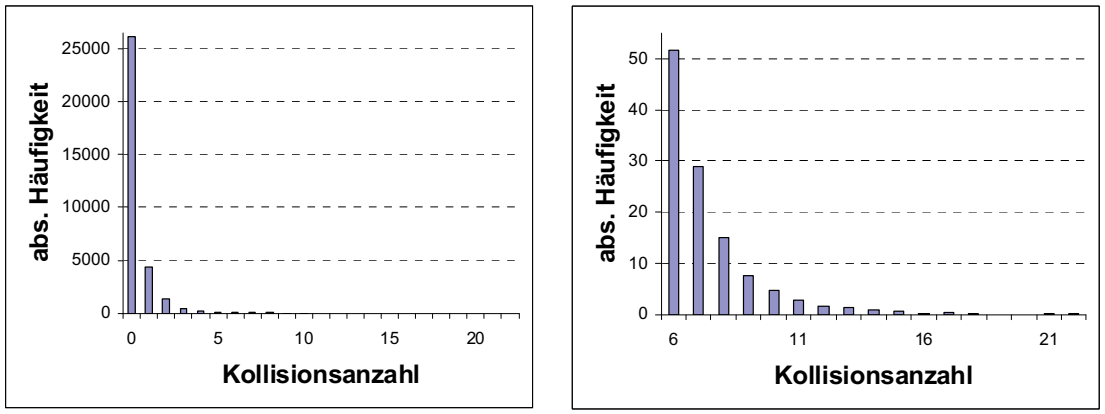


Abbildung 99 - Verhalten von WRGA bei veränderlichen Schlüsselmengen.

6.5 Caching im Datenpfad

Wie in Abschnitt 3.2.4 dargestellt wurde, liegt es im Bereich der Paketklassifizierung auf Grund der mangelnden Lokalität der Daten im Datenstrom nicht auf der Hand, einen Cache zur Verbesserung der Performance zu verwenden. Es besteht jedoch die Möglichkeit, durch Caching eine wichtige Schwäche von Hashfunktionen abzumildern.

Abbildung 100 zeigt die Verteilung aller auftretenden Kollisionen von 32k realistischen Schlüsseln und der Verwendung der leistungsfähigsten der im Rahmen dieser Arbeit vorgestellten Hashfunktionen (3-Mux-XOR). Wie der Grafik zu entnehmen ist, ist die Gesamtleistung sehr gut. Im Mittel treten nur 0,32 Kollisionen pro Schlüssel auf. Von allen Schlüsseln weisen die allermeisten entweder keine (80%) oder nur eine (13%) Kollision auf. Es gibt jedoch einige Schlüssel, die eine größere Zahl von Kollisionen verursachen. In den durchgeführten Testläufen traten Schlüssel auf, die bis zu 22 Kollisionen verursachten. An dieser Stelle liegt es nahe, über die Nutzung eines Caches nachzudenken, um die wenigen Schlüssel zu cachen, die sehr viele Kollisionen erzeugen, da sie ein Risiko für die Klassifizierungsperformance im Datenpfad darstellen. Kommt es zu Bursts mit ungünstigen Schlüsseln, bricht die Leistung ein, und es kommt zu Paketverwürfen.



a) Übersicht b) vergrößerter Ausschnitt

Abbildung 100 - Häufigkeit des Auftretens von Kollisionen bei einer Hashfunktion mit 3-Mux-XOR Architektur.

Der zu verwendende Cache arbeitet nicht, wie es von klassischen Cachearchitekturen bekannt ist, die in Mikroprozessoren genutzt werden. In Prozessoren wird versucht, eine möglichst große Anzahl von Hauptspeicherzugriffen durch den Cache zu bedienen, um die Anzahl der Speicherzugriffe zu minimieren. Das ist sinnvoll, da die Speicher weitaus langsamer arbeiten als ein moderner Prozessor. Somit ist die Maximierung der Cachehits das Optimierungsziel. Der Sachverhalt stellt sich beim EPC anders dar:

- Die Speicheranbindung erfolgt mit Systemgeschwindigkeit, das heißt, der Speicher wird mit dem gleichen Takt betrieben, wie die Hardware des EPCs, womit die Latenz sehr gering ist. Das ist möglich, da auf der Evaluierungsplattform ein schneller SRAM statt des langsameren dynamischen RAM verwendet wird. Cachehits sind also nicht wesentlich schneller als reguläre Speicherzugriffe.
- Es besteht nur geringe oder keine Lokalität der Daten, die zu klassifizieren sind. Eine geringe Hitrate wäre die Folge.

Aus diesen Gründen ist die Speicherstrategie beim verwendeten EPC Cache die Einträge zu speichern, die die meisten Kollisionen bei einer Suche im Speicher verursachen. Damit kann die mittlere Leistung des Paketklassifizierers erhöht werden. Wichtiger ist jedoch der Einfluss des Caches auf die Leistungsfähigkeit des EPCs in Extremfällen, die bei der Verwendung einer Hashfunktion auftreten können. Beide Fälle wurden mittels Simulationen auf dem Hardwareprototyp untersucht und sind im Folgenden dargestellt.

6.5.1 Architektur des Caches

Der eingesetzte Cache [WTT08] nutzt als Speicher BRAMs des FPGAs. In Abbildung 101 ist die Position des Caches innerhalb der Architektur des EPC dargestellt. Wie zu erkennen ist, ist der Cache parallel zu dem jeweils als Datenpfad konfigurierten Speicher eingebunden.

Regelanfragen erreichen sowohl den Cache als auch den eigentlichen Speicher zeitgleich mit dem gehashten Schlüssel.

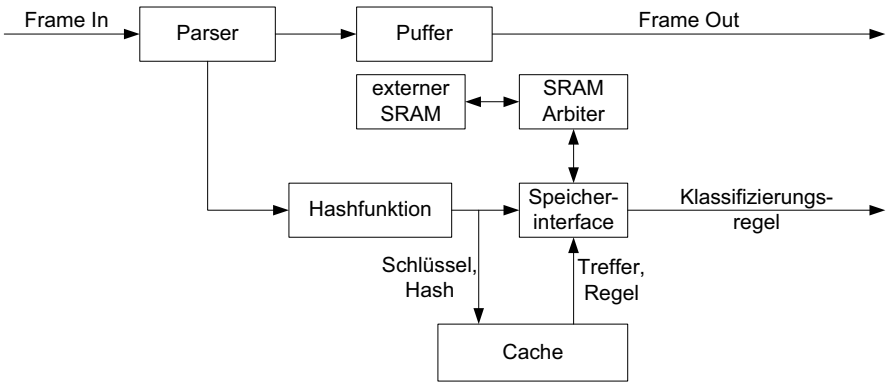


Abbildung 101 - Einbindung des Caches in den Datenpfad.

Der Cache selbst ist weitgehend frei konfigurierbar. Sowohl die Größe als auch der Grad der Assoziativität können angepasst werden. Im Prinzip kann der Cache als Tabelle betrachtet werden. Die Anzahl der Spalten ist durch die Assoziativität definiert. Für jede Assoziativitätsstufe sind ein paralleles Speicherelement und ein zugehöriger Komparator implementiert. Das ermöglicht eine parallele Suche in jedem Block. Die Anzahl der Blöcke ist demzufolge der Quotient aus Größe und Assoziativität. Damit jeder Block durch einen Bitvektor vollständig adressiert werden kann, muss dieser Quotient eine Zweierpotenz sein. Dies stellt eine Randbedingung für die Wahl der Parameter des Caches dar. Das Beispiel eines Caches mit 2-facher Assoziativität ist in Abbildung 102 dargestellt. Falls ein Schlüssel im Cache vorhanden ist, zeigt genau ein Komparator in der entsprechenden Cacheline dies an. Die Cacheline wird durch die Least Significant Bits des Hashwertes des zu suchenden Schlüssels bestimmt. Zur Auswahl der Cacheline kann auch der CRC-Wert des Schlüssels herangezogen werden. Schlüssel mit vielen Kollisionen können gleiche Hashwerte haben. Durch Nutzung einer anderen Hashfunktion im Cache als im Datenpfad wird ermöglicht, dass auch bei geringer Assoziativität die Schlüssel mit den meisten Kollisionen im Cache abgelegt werden können, ohne einander zu verdrängen. Eigene Experimente haben dies bestätigt. Die Nutzung des CRC-Wertes zur Adressierung des Caches ist vorteilhaft. Die entsprechende Regel wird an das angeschlossene Speicherinterface übertragen. Damit kann die Suchoperation nach einem Speichereintrag verkürzt werden, denn das Speicherinterface muss nun etwaig auftretende Kollisionen nicht mehr auflösen. Das muss nur im Falle eines Cachemisses geschehen. Bei einem Miss wird der entsprechende Speichereintrag, bestehend aus Schlüssel, Regel, Hashwert und Kollisionsanzahl, an den Cache übergeben. Dieser versucht dann den Wert in der entsprechenden Cacheline einzutragen. Diese Logik ist in der Abbildung nicht dargestellt. Sollte die gewählte Cacheline voll sein, wird ein existenter Eintrag gegebenenfalls ersetzt. Dazu vergleicht der Cache den Eintrag, der die wenigsten Kollisionen erzeugt, mit dem aktuellen

Eintrag. Sollte dieser mehr Kollisionen erzeugen, ersetzt er den alten Eintrag. Damit ist sichergestellt, dass sich die Schlüssel mit den meisten Kollisionen im Cache befinden und mit einer Anfrage gefunden werden.

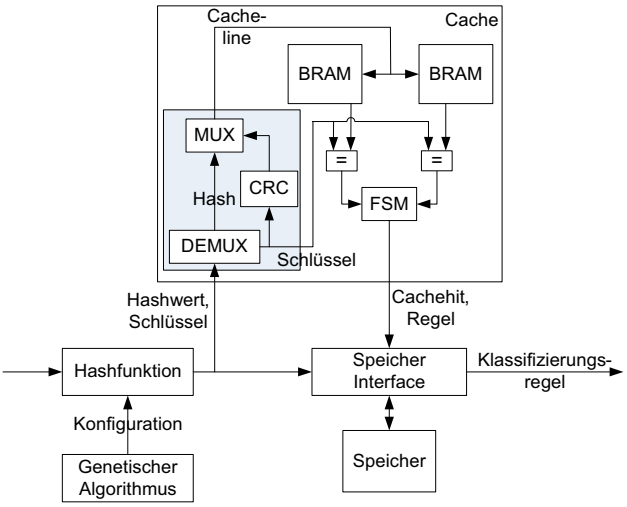


Abbildung 102 - Schematischer Aufbau des Caches mit 2-facher Assoziativität.

6.5.2 Experimentelle Ergebnisse

Um eine objektive Einschätzung der Performance eines Caches im Datenpfad des EPCs zu ermöglichen, wurden mit dem Hardwareprototyp verschiedenen Experimente durchgeführt, bei denen die unterschiedlichen Parameter des eingesetzten Caches variiert wurden. Allen Experimenten liegen dieselben Randbedingungen zu Grunde (Tabelle 14).

Tabelle 14 - Randbedingungen des EPCs

Hash Funktion		3-Mux-XOR
Schlüsselmenge		32k Route Views Daten
Genetischer Algorithmus		
	(μ,λ)	(32,32)
	Mutationsrate	0,01
	Rekombinationsrate	0,95
Cache		CRC-Cache

6.5.2.1 Einfluss der Cachegröße

Um den Einfluss der Cachegröße auf die Performance des Caches zum ermitteln, wurden Untersuchungen mit Cachegrößen zwischen 256 und 8.192 Einträgen getätigt. Das entspricht bei einer Schlüsselmenge von 32.768 zwischen 0,78% und 25% der Einträge im Klassifizierer. Es wurden jeweils Caches mit einer 4-fachen Assoziativität genutzt. Zum Testen der

Leistungsfähigkeit wurden insgesamt zwei Millionen Pakete mit den konfigurierten Schlüsseln gleichverteilt an den EPC versendet. Die wichtigsten Ergebnisse dieser Untersuchungen können Abbildung 103 entnommen werden.

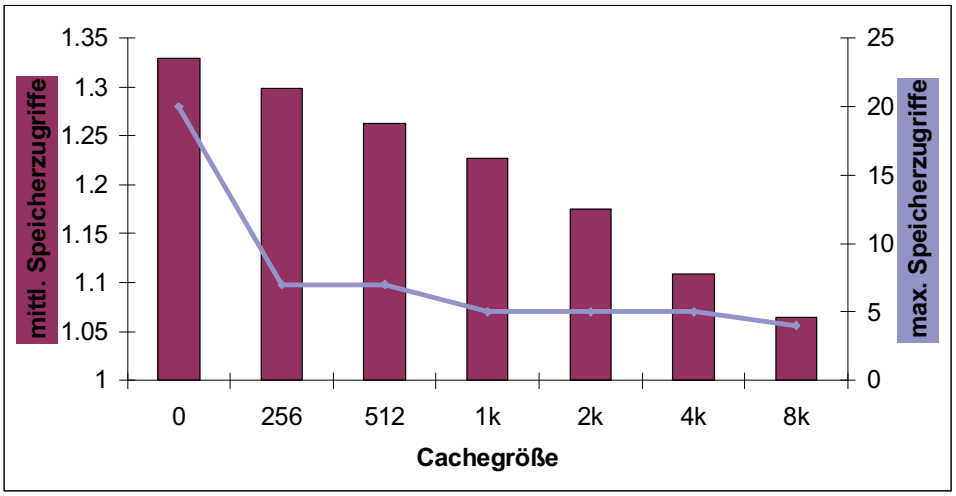


Abbildung 103 - Einfluss der Cachegröße auf die mittlere und die worst-case Leistungsfähigkeit der Paketklassifizierung von 32.768 Schlüsseln.

Wird kein Cache eingesetzt, sind im Mittel 1,32 Speicherzugriffe für jeden Eintrag notwendig. Beim Einsatz eines Caches sinkt diese auf bis zu 1,06 Speicherzugriffe. Dies ist jedoch nur der Fall, wenn der Cache sehr groß ist. 1,06 Zugriffe wurden mit einem Cache erreicht, dessen Größe mit 8.192 Einträgen genügt, um 25% aller existierenden Einträge abzudecken.

Mindestens ebenso wichtig wie der Aspekt der mittleren Verbesserung der Performance ist die Begrenzung der maximal notwendigen Speicherzugriffe (Abbildung 104).

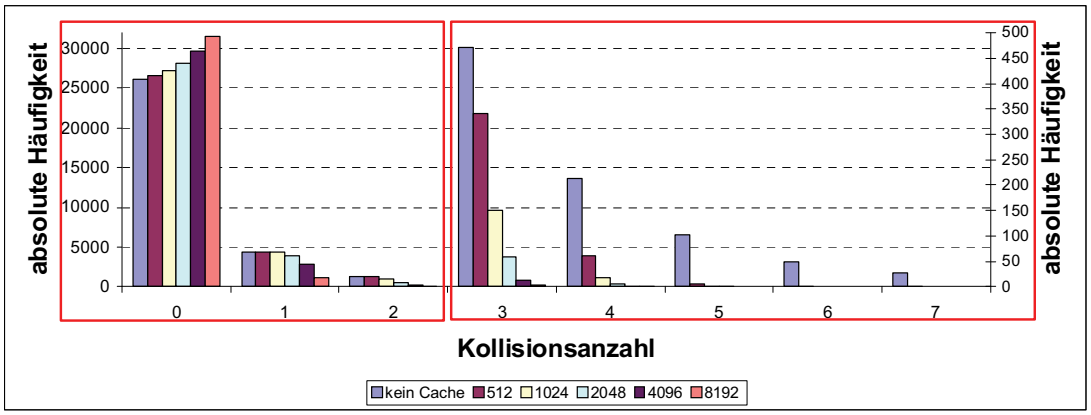


Abbildung 104 - Darstellung der absoluten Häufigkeit von Kollisionen für 32k Schlüssel.

Es wird deutlich, dass vor allem die Leistungsfähigkeit für den ungünstigsten Fall enorm zunimmt. Während ohne Caching in den Simulationen für einige Schlüssel bis zu 22 Kollisionen auftraten, waren mit Cache nur maximal 7 Kollisionen zu beobachten. Das ist bereits bei einem sehr kleinen Cache von 256 Einträgen zu erreichen, was etwa 0,8% der vorhandenen Schlüssel entspricht. Die Steigerung der worst-case-Performance bei Caches mit bis zu 25% Kapazität ist nur noch relativ gering im Verhältnis zu einem System mit sehr kleinem Cache.

6.5.2.2 Einfluss der Assoziativität

Um den Einfluss der Assoziativität des Caches auf dessen Performance zu untersuchen, wurden bei konstanter Schlüsselmenge (32k) und konstanter Cachegröße (1k) verschiedene Grade der Assoziativität untersucht. Im Gegensatz zur Cachegröße, die ausschließlich die notwendigen Speicherressourcen beeinflusst, nutzt der Cache umso mehr Logikressourcen, je stärker die Assoziativität ausgeprägt ist. Abbildung 105 stellt die Größe des Caches in Abhängigkeit von dessen Assoziativität dar. Da die auf der Hardwareplattform zur Verfügung stehenden Ressourcen begrenzt sind, ist maximal die Untersuchung einer 8-fachen Assoziativität möglich.

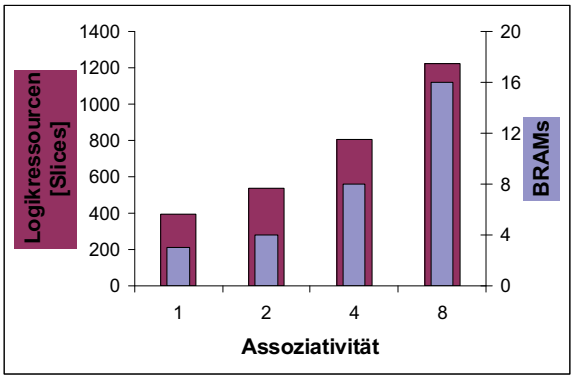
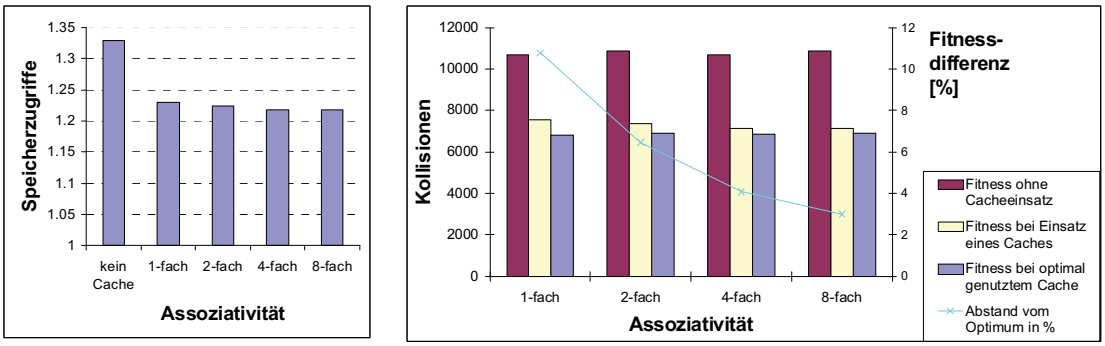


Abbildung 105 - Abhängigkeit der notwendigen Logikelemente von der gewählten Assoziativität.

Den Grad der Verbesserung der Leistungsfähigkeit des Caches im Kontext zu der gewählten Assoziativität stellt Abbildung 106 dar. Es ist zu erkennen, dass die Performance des Caches mit steigender Assoziativität zunimmt. Es ist allerdings ebenfalls festzustellen, dass die Steigerungen nicht sehr bedeutend sind. Abbildung 106a verdeutlicht das sehr gut. Der entscheidende Faktor für die Leistungssteigerung ist die Existenz eines Caches. Ein hoher Grad an Assoziativität hat keinen signifikanten Einfluss. Abbildung 106b verdeutlicht, dass der Cache selbst bei nur 1-facher Assoziativität nahe dem theoretischen Optimum liegt, das ein Cache der entsprechenden Größe liefern kann. Optimal ist der Cache, wenn in seinen n Einträgen die n Schlüssel abgelegt sind, die die meisten Kollisionen erzeugen. In einem voll assoziativen Cache wäre das der Fall. Der Abstand zum Optimum liegt beim Cache mit 1-facher Assoziativität bei

knapp 11%. Ist der Cache 8-fach ausgelegt verringert sich der Abstand bis auf 3%. Ins Verhältnis zu den Hardwarekosten gesetzt, ist zu konstatieren, dass ein 1-fach assoziativer Cache das beste Kosten-Nutzen Verhältnis aufweist.



a) Speicherzugriffe mit Caches verschiedener Assoziativitäten

b) Einfluss der Assoziativität auf die Fitness der eingesetzten Hashfunktion

Abbildung 106 - Leistungsfähigkeit eines Caches beim Einsatz unterschiedlicher Assoziativitäten und gleichbleibender Größe.

6.5.2.3 Verhalten des Caches für Schlüsselmengen verschiedener Größen

Da der EPC Prototyp nur die Untersuchung von bis zu 32786 Schlüsseln erlaubt, soll an dieser Stelle abgeschätzt werden, wie das Verhalten des Systems beim Einsatz größerer Schlüsselmengen wäre. Dazu wurde das Verhältnis von Schlüsselmenge zu Cachegröße beibehalten und das Verhalten für Schlüsselmengen von 2k bis 32k untersucht. Etwa 0,8% der Schlüssel können im Cache gespeichert werden. Das entspricht bei 32k Schlüsseln einem Cache mit 256 Einträgen und bei 1M Schlüsseln 8k Cacheeinträgen. Damit ist der Cache hinreichend klein, um auch bei größeren Schlüsselmengen eingesetzt zu werden.

Wie zu erwarten ist, liegt die Hitrate des Caches seiner Größe entsprechend bei 0,79%. Der Wert ist für alle untersuchten Schlüsselmengen konstant.

In Abbildung 107 ist im Säulendiagramm die mittlere Anzahl der notwendigen Speicherzugriffe dargestellt. Sie nehmen mit zunehmender Schlüsselmenge leicht zu. Das gilt sowohl mit als auch ohne Cache, wobei der Anstieg mit verwendetem Cache etwas weniger steil ausfällt. Das kann den Werten entnommen werden, die die Verbesserung des Durchsatzes im Datenpfad darstellen. Die Verbesserung liegt zwischen 2,6% und 3,2% und steigt mit zunehmender Schlüsselmenge stetig an. Angesichts der Ergebnisse kann vermutet werden, dass auch bei größeren Schlüsselmengen ein in etwa linearer Anstieg bei Verdoppelung der Schlüsselmenge vorliegt. Es ist anzunehmen, dass, wenn der Anstieg konstant bleibt, bei einer Größe der Schlüsselmenge von 1M 1,40 Zugriffe ohne und 1,35 Zugriffe mit Cache notwendig werden.

Der Einfluss eines solch kleinen Caches auf die worst-case-Performance ist jedoch wesentlich interessanter. Wie der Abbildung außerdem zu entnehmen ist, steigt ohne Cache auch die maximale Anzahl der Speicherzugriffe stark an. Wird ein Cache verwendet, bleibt sie jedoch weitgehend konstant bzw. steigt nur sehr moderat an.

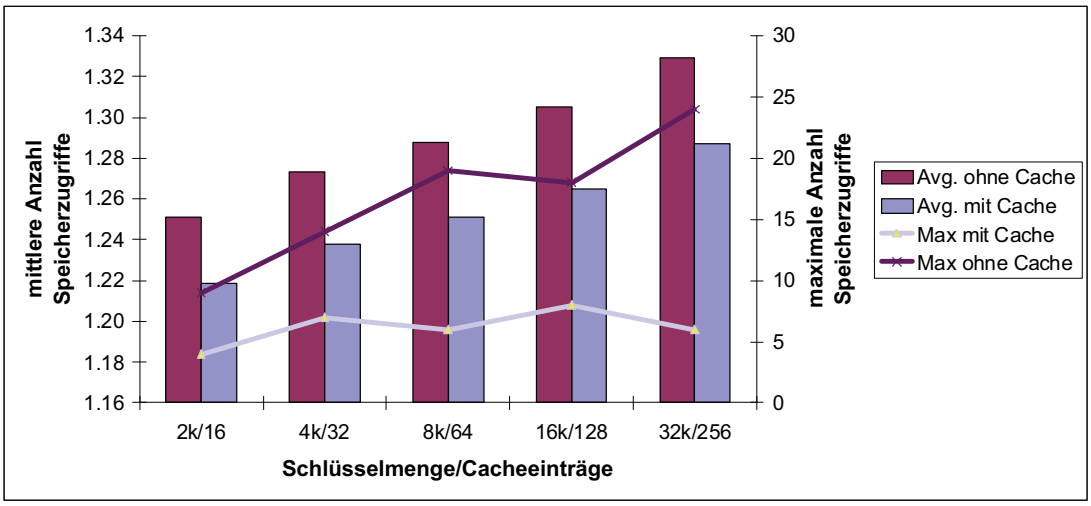


Abbildung 107 - Performance der Hashfunktionen mit und ohne den Einsatz eines kleinen Caches, der 0,8% der Speichereinträge cachen kann.

6.6 Zusammenfassung der sinnvollsten Optimierungen

Aufgrund der Tatsache, dass zu Beginn einer Evaluierung die Fitnesswerte sehr schlecht und damit die Leistungsfähigkeit der Hashfunktion mangelhaft ist, sind Maßnahmen zur Beschleunigung der Evaluierung empfehlen [WTS+07]. MI ist uneingeschränkt und mit einem möglichst hohen Grad an Parallelität zu empfehlen. ET ist nur dann einzusetzen, wenn eine Umweltselektion stattfindet. Der Grad, in dem PFE einzusetzen ist, hängt aufgrund der erheblichen notwendigen Hardwareressourcen davon ab, ob genügend dieser Ressourcen zur Verfügung stehen. Um die Ergebnisse zu optimieren, sind auf jeden Fall auch die ermittelten optimalen Werte für Populationsgröße (32), Selektionsdruck ($k = 6$) und Mutationsrate (Bäck/Schütz) zu verwenden. Ebenfalls nicht fehlen sollte ein Cache, dessen Größe durchaus sehr klein sein kann und der dennoch sehr effektiv ist [WTT08].

Die Smart Initialization und der Einsatz von WRGA haben sich als nicht effektiv erwiesen, um die Leistungsfähigkeit des EPC zu erhöhen. Von einem Einsatz dieser Optimierungen ist deshalb abzuraten.

7 Zusammenfassung der Ergebnisse

In der vorliegenden Arbeit wurden zwei Bereiche aus dem Gebiet der paketverarbeitenden Systeme untersucht und Lösungen für verschiedene Problematiken dieser Bereiche, der Paketverarbeitung und der Paketklassifizierung, entwickelt und vorgestellt.

7.1 Paketverarbeitung

Die primären Herausforderungen bei der Verarbeitung von Paketen, vor allem im Bereich des Teilnehmerzugangs, sind zum einen die Notwendigkeit, immer größere Datenmenge schritthaltend zu verarbeiten; zum anderen sind immer mehr und immer komplexere Funktionalitäten bereits auf Teilnehmerzugangssystemen gefordert. Einen Beitrag zur Lösung dieser Problematiken stellt die im Rahmen dieser Arbeit vorgestellte FPGA-basierte Architektur eines funktionalen Moduls dar, das es ermöglicht, maßgeschneiderte Funktionalitäten im Bereich der Paketverarbeitung in Hardware zu entwickeln. Diese können in aktuellen und zukünftigen Teilnehmerzugangssystemen zum Einsatz kommen. Das FM ist in der Lage, Datenpakete mit wirespeed zu verarbeiten. Der Nachweis der Anwendbarkeit der Architektur konnte geführt werden, indem drei unterschiedliche Funktionalitäten, die in aktuellen und zukünftigen Teilnehmerzugangssystemen notwendig sind, auf Basis der FM-Architektur entwickelt und implementiert wurden:

- Die MAC Address Translation – MAT dient dem Ersetzen von Customer-MACs durch Provider-MACs. Das entwickelte Hardwaremodul ist sowohl im Up- als auch im Downstream eines DSLAMs einzusetzen.
- Das Multiprotocol Label Switching User Network Interface – MPLS-UNI ermöglicht es, MPLS Label Stacks in jeden Ethernetframe eines Datenstroms einzufügen. Damit können Datenframes mit zusätzlicher Information angereichert werden. Das MPLS-UNI ist im Upstream eines DSLAMs einzusetzen.
- Der Traffic Manager – TM ermöglicht die teilnehmerabhängige Farbmarkierung von Datenframes mit den Farben rot, gelb und grün. Rote Frames können direkt durch den TM verworfen werden, womit Policing-Funktionalität unterstützt wird. Die Farbmarkierungen ermöglichen anderen Netzwerkgeräten ein effizientes Traffic Engineering. Der TM ist im Upstream eines DSLAMs einzusetzen.

Alle drei Funktionalitäten sollen im IP-DSLAM zum Einsatz kommen. Deshalb wurden im Rahmen dieser Arbeit auch Möglichkeiten der Kombination von FMs genauer untersucht: MATMUNI und blockorientierte Verarbeitung. Dabei zeigte sich, dass MATMUNI sehr hardwareeffizient ist, jedoch der Implementierungsaufwand bei mehr als drei oder vier unterschiedlichen Funktionalitäten den Nutzen übersteigt. Die blockorientierte Verarbeitung benötigt mehr Hardwareressourcen als MATMUNI, skaliert aber problemlos mit der Anzahl der

Funktionalitäten, was eine Anwendung bei vielen unterschiedlichen Funktionalitäten sinnvoll erscheinen lässt. Das gilt unter der Voraussetzung, dass genügend Hardwareressourcen zur Verfügung stehen.

Eine weitere etwas komplexere Funktionalität für die Paketverarbeitung, die ebenfalls für den Einsatz im Teilnehmerzugangsbereich konzipiert wurde, ist IPclip. Mit dem Einsatz von IPclip im IP-DSLAM ist es möglich, Trust-by-Wire auch im paketorientierten IP-basierten Internet zu ermöglichen. Der entwickelte Prototyp kann in verschiedenen Bereichen Anwendung finden. Das sind die Bereiche der VoIP-Notrufe, die Bekämpfung von Phishing und auch die Bekämpfung von E-Mail Spam. Alle drei Anwendungsszenarien wurden im Rahmen dieser Arbeit beschrieben.

7.2 Paketklassifizierung

Um Datenpakete korrekt verarbeiten zu können, ist es notwendig, jedes einzelne Datenpaket einer Verarbeitungsvorschrift zuzuordnen. Diese Klassifizierung von Paketen unterliegt folglich den gleichen zeitlichen Anforderungen wie die Paketverarbeitung selbst: Datenströme steigender Bandbreiten müssen schritthaltende verarbeitet werden. Die Anzahl unterschiedlicher Verarbeitungsvorschriften und damit die Größe der zur Klassifizierung zu durchsuchenden Datenbanken steigen ebenfalls rapide an. Beides erhöht die Komplexität der Paketklassifizierung.

Als Beitrag zur Lösung dieser Problematik wurde im Rahmen dieser Arbeit eine Hasharchitektur entwickelt, welche sich auf Basis eines genetischen Algorithmus an die Anforderungen einer Klassifizierungsaufgabe anzupassen vermag. Es wurde eine evolvierbare Hardware entwickelt, auf deren Basis ein FPGA-basierter evolvierbarer Paketklassifizierer implementiert und evaluiert wurde. Die evolvierbare Hashfunktion zeigt bessere Eigenschaften als die dem Autor aus der Literatur bekannten Ansätze und ist außerdem nach bestem Wissen des Autors die einzige evolvierbare Hashfunktion.

Zur Optimierung der Leistungsfähigkeit der evolvierbaren Hashfunktion wurden im Rahmen dieser Arbeit mehrere Ansätze entwickelt. Die erfolgreichen Ansätze sind hier noch einmal kurz aufgeführt:

- Um den Vorgang der Evolution der Hashfunktion zu beschleunigen, wurden die Early Termination (ET), die Parallel Fitness Evaluation (PFE) sowie das Memory Interleaving (MI) entwickelt, implementiert und evaluiert. Es zeigte sich, dass vor allem eine Kombination der Maßnahmen überaus effektiv ist. Damit war es möglich, die Berechnung der ersten 500 Generationen des GA von 301 auf nur noch 76 Sekunden und damit die Boot-Phase des EPC zu verkürzen.
- Um das Ergebnis des GA und somit die finalen Eigenschaften der Hashfunktion zu verbessern, wurden die Operatoren des GA optimiert, wozu die Mutationsrate, die Populationsgröße und die Wettkampfgröße bei der Elternselektion gehören. Damit

konnten die erhofften Verbesserungen erzielt werden. Gegenüber dem CRC32 beispielsweise sind nun für 32k Schlüssel im Mittel noch 1,32 gegenüber 1,46 Speicherzugriffen pro Schlüssel notwendig.

- Die ungünstige Eigenschaft von Hashfunktionen, im Mittel sehr gute, aber in Einzelfällen auch schlechte Ergebnisse (Anzahl der Kollisionen) zu erzeugen, wurde mittels des Einsatzes eines Caches abgeschwächt. Anders als herkömmliche Caches, speichert der entwickelte Cache Schlüssel, die sehr viele Kollisionen erzeugen. Die gemachten Untersuchungen ergaben, dass sogar ein extrem kleiner Cache die Anzahl der maximal auftretenden Kollisionen beim Hashing stark reduziert. In den durchgeführten Experimenten traten statt maximal 22 nur noch maximal 7 Speicherzugriffe für einen Schlüssel auf. Neben den worst-case Eigenschaften konnte mit dem Cache auch die mittlere Leistung der Hashfunktion weiter verbessert werden.

Es ist damit gelungen, eine sehr leistungsfähige, evolvierbare Hashfunktion zu entwickeln, die beim Einsatz in hash-basierten Paketklassifizierungsarchitekturen zu einer deutlichen Verbesserung der Klassifizierungseigenschaften führen wird.

7.3 Ausblick

Im Bereich der Paketverarbeitung ist es notwendig, weitere Anstrengungen zu unternehmen, um die Verbindung verschiedener Funktionalitäten effizienter, einfacher und verlustarm zu ermöglichen. Damit können komplexere und leistungsstarke paketverarbeitende Systeme auf FPGA-Basis in Zukunft noch mehr Funktionalität im Teilnehmerzugangsbereich bereit stellen. Dieser Themenbereich konnte im Rahmen dieser Arbeit mit MATMUNI und der blockorientierten Verarbeitung nur gestreift werden. Die Entwicklung neuer Funktionalitäten für den Teilnehmerzugangsbereich ist ebenfalls ein wichtiger Teil zukünftiger Entwicklungen. Vor allem im Bereich des GPONs als Teilnehmerzugangssystem sind noch viele Entwicklungen möglich.

Im Bereich der Paketklassifizierung mittels evolvierbarer Hashfunktionen sind in erster Linie bei der partiellen Rekonfigurierbarkeit von FPGAs noch große Forschungsanstrengungen notwendig. Positive Entwicklungen in diesem Bereich werden die Einsetzbarkeit und vor allem die Hardwareeffizienz der vorgestellten Architekturen immens verbessern, und somit den Einsatz von evolvierbaren Hashfunktionen begünstigen. Bei der Optimierung der 3-Mux-XOR Architektur stellte sich heraus, dass sehr viele Parameter Einfluss auf die Qualität der Hashfunktion haben, deren gegenseitige Beeinflussung nicht untersucht werden konnte. Möglicherweise könnte der Einsatz eines weiteren GA zur Parameteroptimierung des GA für die Hashfunktion zum Einsatz kommen. Die Untersuchung eines solchen „GA zweiter Ordnung“ ist sicherlich ein lohnendes Forschungsprojekt.

Abbildungen

Abbildung 1 - Gliederung der Arbeit. Die Abschnitte mit den wesentlichen eigenen Beiträgen sind fett dargestellt. 3
Abbildung 2 - Kommunikationsbeziehung zwischen zwei Teilnehmern über einen dazwischen liegenden Router aus Sicht des ISO-OSI Schichtenmodells. 6
Abbildung 3 - Abbildung des ISO-OSI Modells auf die Internet-Architektur mit Beispielprotokollen. 8
Abbildung 4 - Aufbau der verschiedenen Protokollschichten der Internet-Architektur. 8
Abbildung 5 - Aufbau eines Ethernetframes. 9
Abbildung 6 - Aufbau eines IPv4-Pakets. 10
Abbildung 7 - Aufbau eines TCP-Datagramms. 11
Abbildung 8 - Netzwerkstruktur des Internets. 12
Abbildung 9 - Aufbau eines Teilnehmerzugagnsnetzwerks. 13
Abbildung 10 - Verkehrsaufkommen am größten europäischen Internetknotenpunkt DE-CIX in Frankfurt/Main (Quelle: [Ger08]). 15
Abbildung 11 - Darstellung des Wachstums der Anzahl der Einträge in verschiedenen BGP-Routingtabellen (Quelle: [Hus08]). 16
Abbildung 12 - Schematisch Darstellung der Paketklassifizierung. 17
Abbildung 13 - Quersummenbildung als Hashfunktion. 19
Abbildung 14 - Kollisionsauflösung durch Verkettung. 20
Abbildung 15 - Lineare Kollisionsauflösung durch Addition einer Konstanten (hier $c = 1$). 21
Abbildung 16 - Geometrie der Aufhängung einer Satellitenantenne. Links das originale reguläre Design. Auf der rechten Seite ist das finale Design abgebildet, welches mit einem genetischen Algorithmus ermittelt wurde [KB96]. 24
Abbildung 17 - n-point Crossover. 28
Abbildung 18 - 1-point Crossover. 28
Abbildung 19 - Mutation eines Bits des Genoms. 28
Abbildung 20 - Verlauf eines GA. Das globale Optimum wird nicht erreicht. Der Algorithmus konvergiert zu einem lokalen Optimum [Toc07]. 30
Abbildung 21 - Typisches Laufzeitverhalten von evolutionären Algorithmen [Toc07]. Dabei kann es von einer Generation zur nächsten, abhängig von der gewählten Umweltselektionsstrategie, auch zu graduellen Verschlechterungen der Fitness kommen. 31

Abbildung 22 - Vereinfachter Ablauf eines genetischen Algorithmus mit den Operatoren Elternselektion (s), Rekombination (r) und Mutation (m). 31
Abbildung 23 - Eine einfache FPGA-Architektur [GT07]. 32
Abbildung 24 - Aufbau eines 8-Bit LFSR. Die Werte der Stufen vier, fünf, sechs und acht werden auf den Registereingang zurückgeführt. 34
Abbildung 25 - Verhältnis von Flexibilität und Leistungsfähigkeit verschiedener paketverarbeitender Systeme nach [Sha01]. 35
Abbildung 26 - Architektur des GigaNetIC nach [NPS+05]. 39
Abbildung 27 - Aufbau des FPX-Systems nach [LNT+01]. 42
Abbildung 28 - DynaCORE Architektur (nach [AFK+06]). 43
Abbildung 29 - Darstellung eines einfachen Tries und der Position von fünf Präfixen (P1 – P5). 45
Abbildung 30 - Darstellung eines PATRICIA Trees und der Position von fünf Präfixen (P1 – P5). 46
Abbildung 31 - Controlled Prefix Expansion. 47
Abbildung 32 - DIR-24-8 Architektur [GLM98]. Das Klassifizierungsergebnis ergibt sich entweder direkt aus TBL24 oder aus TBLlong. 48
Abbildung 33 - Indirect Lookup-Mechanismus mit variabler Offsetlänge (nach [HZ99]). 49
Abbildung 34 - Partitionierung des Binärbaums in Unterbäume mit 255 Knoten [PLW+03]. 50
Abbildung 35 - Beispielhafter Trie vierten Grades mit dem korrespondierenden Bitmuster [MF02]. 52
Abbildung 36 - Hashfunktionen der Klasse H_3. 56
Abbildung 37 - Aufbau der Architektur von Nie. 58
Abbildung 38 - Architektur eines Funktionalen Moduls bestehend aus FB, CA, Speicher und AE. 67
Abbildung 39 - Framebuffer mit Parser mit integrierten Teilmodulen zum Parsen von SRC-MAC, VLAN und DST-IP. Die grau schraffierten Elemente sind nicht für die Synthese konfiguriert. 68
Abbildung 40 - Funktionales Modul mit vier parallelen Datenpfaden. 69
Abbildung 41 - CA Architektur mit LLF Scheduling Modul und Demultiplexer zur Übertragung von Regeln an die parallelen EAs. 71
Abbildung 42 - Zustandsmaschine des Suchalgorithmus mit und ohne Prädiktion. 73
Abbildung 43 - Struktur von Ethernet Frames ohne und mit MPLS Label Stack. 76
Abbildung 44 - Verlauf der Zählerwerte und Farbmarkierungen der Daten eines Nutzers. 79
Abbildung 45 - Architektur des TMs (einzelner Datenpfad). 80

Abbildung 46 - Schematische Darstellung der Architektur des MAT Moduls (vgl. Abbildung 40).....82
Abbildung 47 - Verhalten des MPLS-UNI mit 4 parallelen Datenpfaden und einer Gesamtdatenrate von 4 GBit/s.....83
Abbildung 48 - Verhalten des TM mit 4 parallelen Datenpfaden und einer Gesamtdatenrate von 4 GBit/s.....84
Abbildung 49 - Verhalten von MAT mit jeweils vier parallelen Datenpfaden im Up- und Downstream und einer Gesamtdatenrate von 8 GBit/s. 84
Abbildung 50 - Verteilung von realem Internetdatenverkehr.....85
Abbildung 51 - Architektur von IPclip.....89
Abbildung 52 - Validierung von Ortsinformationen durch das Option Verification Module. 90
Abbildung 53 - Verarbeitung von IP-Paketen durch AIA. 91
Abbildung 54 - Performance des IPclip-Prototyps. Dargestellt sind der Durchsatz des Systems bei voller Last (1 GBit/s), die Verlustrate in Abhängigkeit vom Anteil der Datenframes, die bereits über OI verfügen und die durch IPclip in den Datenpfad eingefügte Verzögerung. 93
Abbildung 55 - Architektur von MATMUNI mit allen Funktionalitäten von MAT, TM und MPLS-UNI in Up- und Downstream. 99
Abbildung 56 - Performance von MATMUNI in der Konfiguration, wie sie in Abbildung 55 dargestellt ist. 101
Abbildung 57 - Architekturvorschlag eines blockorientierten Paketprozessorsystems mit dem gleichen funktionalen Umfang wie MATMUNI.....102
Abbildung 58 - Leistungsfähigkeit eines Systems, das in seiner Funktionalität MATMUNI entspricht, aber als einfache Kombination von FMs implementiert ist.....103
Abbildung 59 - Abbildung der Funktionalität von MATMUNI auf ein NoC (Abbildung aus [Kub08]). 103
Abbildung 60 - Datenpfadoperation. 106
Abbildung 61 - Struktur des Datenpfades des EPCs. 106
Abbildung 62 - Abbildung der Struktur eines Speichereintrags auf einen 16 Bit breiten Speicher. Der Schlüssel ist in dem Beispiel 32 Bit breit, die Klassifizierungsregel hat 10 Bit. Damit sind vier Speicheradressen (64 Bit) zu verwenden. 108
Abbildung 63 - Architektur des Kontrollpfades des EPC.....111
Abbildung 64 - Aufbau des Gesamtsystems des evolvierbaren Paketklassifizierers mit implementiertem Speicher im externen SRAM. Die parallelen Pfade im Kontrollpfad müssen sich den Speicherzugriff über einen Arbiter teilen. 112

Abbildung 65 - Architektur einer Hashfunktion, die aus miteinander verknüpften 4-Eingangs-LUTs besteht. Die Anzahl der Stufen ist variabel. 114
Abbildung 66 - Hashfunktion mit XOR-verknüpften Multiplexern (2Mux-XOR). 115
Abbildung 67 - Zweistufige Hashfunktion mit XOR-verknüpften Multiplexern (2-Mux-XOR-2-Stage). 116
Abbildung 68 - Hashfunktion mit drei XOR-verknüpften Multiplexern (3-Mux-XOR). 117
Abbildung 69 - Hashfunktion mit drei Multiplexern, welche mit einer beliebigen, variablen 3-Eingangslogikfunktion verknüpft sind 117
Abbildung 70 - Architektur der evolvierbaren Hashfunktion der Klasse H_3. 119
Abbildung 71 - Architektur des Fitnessevaluierungsmoduls. 120
Abbildung 72 - Präfixverteilung der verwendeten BGP-Routingtabelle (logarithmische Skalierung)..... 123
Abbildung 73 - Mittlere Fitness (Kollisionsanzahl) unterschiedlicher Hashfunktionen mit 32k Schlüsseln aus realistischen Schlüsseldaten. Es ist jeweils der Fitnesswert über der Architektur aufgetragen. Je kleiner der Fitnesswert ist, desto besser ist die Leistungsfähigkeit der jeweiligen Hashfunktion. 125
Abbildung 74 - Mittlere Kollisionsanzahl unterschiedlicher Hashfunktionen mit 64k und 128k Schlüsseln aus realistischen Schlüsseldaten. 126
Abbildung 75 - Speicherzugriffe pro Schlüssel für unterschiedlich große Schlüsselmengen der verschiedenen Hasharchitekturen. Je weniger Speicherzugriffe notwendig sind, desto besser ist die Hashfunktion. 1,0 Speicherzugriffe stellen das theoretische Minimum dar. Dieses wäre bei einer perfekten Hashfunktion erreicht. Für die Flechtercheckssumme liegen nur Werte für 32k Schlüssel vor, da der Hashwert maximal 16 Bit breit ist. 127
Abbildung 76 - Verhalten von 3-Mux-XOR und evolvable-H3 über 100.000 Generationen (32k Schlüssel). 128
Abbildung 77 - Fitness der Architektur mit 3-Mux-XOR und CRC32 bei zufällig generierten Schlüsseln und unterschiedlichen realen Schlüsseln. 129
Abbildung 78 - Fitnessverlauf des GA mit 32k Schlüsseln über 2000 Generationen. Es wurden alle 200 Generationen 4k Schlüssel ausgetauscht, bis ein kompletter Schlüsselaustausch sattfand. 130
Abbildung 79 - Fitnessverlauf des GA mit 32k Schlüsseln über 5000 Generationen. Es wurden alle 500 Generationen 4k Schlüssel ausgetauscht, bis ein kompletter Schlüsselwechsel stattfand. 130

Abbildung 80 - Einfluss von T_{reconf} auf die Gesamtberechnungszeit des genetischen Algorithmus. Im Mittel steigt die Berechnungszeit bei 500 Generationen um 4,7%. 134
Abbildung 81 - Zeitverhalten des genetischen Algorithmus. Mittelwert, obere und untere Grenze bei 10 Untersuchungen. 135
Abbildung 82 - Architektur der Fitnessevaluierung mit integrierter ET. Die Änderungen zur originalen Fitnessevaluierung sind farblich hervorgehoben. 136
Abbildung 83 - Einfluss des vorzeitigen Abbruchs auf die Rechenzeit des GA. 137
Abbildung 84 - Architektur der Fitnessevaluierung mit implementiertem Memory Interleaving. 138
Abbildung 85 - Zeitverlauf des GA. Einfluss der parallelen Fitnessevaluierung. 139
Abbildung 86 - Originale Fitnessevaluierung gegenüber PFE. Bevor ein neuer Schlüssel evaluiert werden kann, müssen alle Evaluierungsmodule die Berechnungen für den vorherigen Schlüssel abgeschlossen haben. 140
Abbildung 87 - Einfluss der parallelen Fitnessevaluierung auf den Zeitverlauf des GA. 141
Abbildung 88 - Notwendige Zeit, um 500 Generationen mit einem (32,64)-GA und 32k Schlüsseln zu evaluieren. Die eindeutig beste Leistung erzielt die Kombination aller Ansätze. 142
Abbildung 89 - Rechenzeit des 3-Mux-XOR Algorithmus zur Berechnung von 2000 Generationen des GA für unterschiedlich große Schlüsselmengen. Dabei ist bereits $8 \times$ MI implementiert. 143
Abbildung 90 - Entwicklung der Fitness des Genetischen Algorithmus über 2000 Generationen. Es fanden Populationsgrößen zwischen 8 und 128 Individuen Verwendung. 145
Abbildung 91 - Zeitlicher Verlauf der Entwicklung der Fitnesswerte der Hashfunktion bei Nutzung unterschiedlicher Populationsgrößen. 146
Abbildung 92 - Erreichte Fitness der Hashfunktion nach unterschiedlichen Zeiträumen bzw. dem Ablauf von 2000 Generationen. 147
Abbildung 93 - Mittlere Fitness der besten Individuen des GA nach Ablauf von 2000 Generationen. Es wurden für jede Wettkampfgröße 20 Untersuchungen vorgenommen. 148
Abbildung 94 - Einfluss unterschiedlicher konstanter Mutationsraten auf die Performance der evolvierbaren Hashfunktion. 150
Abbildung 95 - Vergleich der variablen Mutationsraten mit der besten konstanten Rate von $2/l$. 152

Abbildung 96 - Eigenschaften des GA mit kombinierten Parametern und Vergleich mit den Einzelperformances. 153
Abbildung 97 - Verlauf der Fitness des originalen GA und von WRGA über 10.000 Generationen. Nach jeweils 500 Generationen erfolgte eine Neuinitialisierung des WRGA. 156
Abbildung 98 - Verlauf der Fitness des originalen GA und von WRGA über 100.000 Generationen. Nach jeweils 5.000 Generationen erfolgte eine Neuinitialisierung des WRGA. 156
Abbildung 99 - Verhalten von WRGA bei veränderlichen Schlüsselmengen. 157
Abbildung 100 - Häufigkeit des Auftretens von Kollisionen bei einer Hashfunktion mit 3-Mux-XOR Architektur. 158
Abbildung 101 - Einbindung des Caches in den Datenpfad. 159
Abbildung 102 - Schematischer Aufbau des Caches mit 2-facher Assoziativität. 160
Abbildung 103 - Einfluss der Cachegröße auf die mittlere und die worst-case Leistungsfähigkeit der Paketklassifizierung von 32.768 Schlüsseln. 161
Abbildung 104 - Darstellung der absoluten Häufigkeit von Kollisionen für 32k Schlüssel. 161
Abbildung 105 - Abhängigkeit der notwendigen Logikelemente von der gewählten Assoziativität. 162
Abbildung 106 - Leistungsfähigkeit eines Caches beim Einsatz unterschiedlicher Assoziativitäten und gleichbleibender Größe. 163
Abbildung 107 - Performance der Hashfunktionen mit und ohne den Einsatz eines kleinen Caches, der 0,8% der Speichereinträge cachen kann. 164
Abbildung 108 - IP-Option, die eine IPclip-Option mit GPS Daten enthält. Insgesamt sind 11 Byte notwendig. 198
Abbildung 109 - Fitnessentwicklung der evolvierbaren Hashfunktion für 131072 Schlüssel. Auf der Y-Achse ist jeweils die Fitness über der Generation für 20 unabhängige Simulationsläufe aufgetragen. Die schwarze Linie stellt den Mittelwert dar. Die Untersuchungen wurden für einen genetischen Algorithmus und 2000 Generationen durchgeführt. f) verdeutlicht die Entwicklung und das Verhältnis der Fitnesswerte der einzelnen Architekturen im Verlauf von 2000 Generationen zueinander. 199
Abbildung 110 - Fitnessentwicklung der evolvierbaren Hashfunktion für 65536 Schlüssel. Auf der Y-Achse ist jeweils die Fitness über der Generation aufgetragen. Die Untersuchungen wurden für einen genetischen Algorithmus und 2000 Generationen durchgeführt. f) verdeutlicht die Entwicklung und das Verhältnis der Fitnesswerte der

einzelnen Architekturen im Verlauf von 2000 Generationen zueinander. 200
Abbildung 111 - Fitnessentwicklung der evolvierbaren Hashfunktion für 32768 Schlüssel. Auf der Y-Achse ist jeweils die Fitness über der Generation aufgetragen. Die Untersuchungen wurden für einen genetischen Algorithmus und 2000 Generationen durchgeführt. f) verdeutlicht die Entwicklung und das Verhältnis der Fitnesswerte der einzelnen Architekturen im Verlauf von 2000 Generationen zueinander 201
Abbildung 112 - Entwicklung der Fitness des EPC ohne und mit wiederholt reinitialisiertem GA. Es wurden 10000 Generationen evaluiert, wobei im Falle von WRGA alle 500 Generationen der GA reinitialisiert wurden. Es ist zu erkennen, dass sich WRGA mit dieser Konfiguration nicht besser verhält und auch der Originalalgorithmus nach mehreren Tausend Generationenen nicht vollständig terminiert. 202
Abbildung 113 - Verhalten des GA bei der Untersuchung unterschiedlicher Populationsgrößen. Es wurde für jeweils 20 unabhängige Simulationsläufe die Fitness über der Zeit in Sekunden aufgetragen.....203

Tabellen

Tabelle 1 - Komplexitäten verschiedener einfacher Suchverfahren 23
Tabelle 2- Vergleich der TCAM und SRAM-Technik [JP08]. 47
Tabelle 3 - Vergleich unterschiedlicher Hardwarelösungen. 53
Tabelle 4 - Paketklassifizierungsalgorithmen und die verwendeten Hashfunktionen. 62
Tabelle 5 - Hardwarebedarf des MPLS-UNI. 77
Tabelle 6 - Ressourcenbedarf des IPclip-Prototyps. 92
Tabelle 7 - Flags der IPclip-Option. 96
Tabelle 8 - Ressourcenbedarf von MATMUNI. 100
Tabelle 9 - Genomgrößen der verschiedenen Hashfunktionsarchitekturen bei unterschiedlich großen Schlüsselmengen [Bit]. 124
Tabelle 10 - Ressourcenverbrauch der verschiedenen Hashfunktionsarchitekturen bei unterschiedlich großen Schlüsselmengen [Slices]. 124
Tabelle 11 - Konfiguration des GA zur Überprüfung der Beschleunigung der Fitnessevaluierung durch Vorzeitigen Abbruch, Parallele Fitnessevaluierung und Speicherverschränkung. 135
Tabelle 12 - Überblick über das Verhältnis von Hardwarekosten zu Evaluierungsbeschleunigung der verschiedenen Maßnahmen. 143
Tabelle 13 - Parameter des GA für die durchgeführten Experimente 144
Tabelle 14 - Randbedingungen des EPCs 160
Tabelle 15 - Farbcodierung des EXP Feldes des MPLS Labels 194
Tabelle 16 - 8P0D: Es existieren acht Prioritäten und keine Farbmarkierungen. 195
Tabelle 17 - 7P1D: Es exitsitern sieben Prioritäten und eine Möglichkeit der Farbmarkierung. 195
Tabelle 18 - 6P2D: Es existieren sechs Prioritäten und zwei Möglichkeiten der Farbmarkierung. 195
Tabelle 19 - 5P3D: Es existieren fünf Prioritäten und drei Möglichkeiten der Farbmarkierung. 196
Tabelle 20 - Farbmarkierung bei Nutzung des DEI-Felds einer SVLAN. 196
Tabelle 21 - Flags der IPclip-Option. Darstellung des allgemeinen Falls (vgl. mit Tabelle 7). 197
Tabelle 22 - Format der GPS Ortsinformationen. 197

Literatur

[ABB+03] Allen J.R. Jr., Bass B.M., Basso C., Boivie R.H., Calvignac J.L., Davis G.T., Frelechoux L., Heddes M., Herkersdorf A., Kind A., Logan J.F., Peyravian M., Rinaldi M.A., Sabhikhi R.K., Siegel M.S., Waldvogel M.: "IBM PowerNP Network Processor: Hardware, Software, and Applications", IBM J. RES & DEV., vol. 47 no. 2/3, 2003

[ADL+07] Allman D., Delany J.M.C., Libbey M., Fenton J., Thomas M.: "DomainKeys Identified Mail (DKIM) Signatures", RFC 4871, http://www.rfc-archive.org/getrfc.php?rfc=4871, 2007

[AFK+06] Albrecht C., Foag J., Koch R., Maehle E.: "DynaCORE – A Dynamically Reconfigurable Coprocessor Architecture for Network Processors", Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-based Processing (PDP), S. 101-108, 2006

[Age05] Agere Systems: "Implementing Demanding Traffic Processing Requirements for Next-Generation DSLAM Network Architectures", White Paper, http://www.agere.com/telecom/TrueAdvantage/docs/DSLAM_White_Paper.pdf, 2005

[Agi01] Agilent Technologies: "JTC 003 Mixed Packet Size Throughputs", White Paper, http://advanced.comms.agilent.com/n2x/docs/insight/2001-08/Testing Tips/1MxdPktSzThroughput.pdf, 2001

[Ahn06] Ahn, C.W.: "Advances in Evolutionary Algorithms", Springer, Heidelberg, 2006

[Alt04] Altera: "Nios 3.0 CPU", Datasheet, http://www.altera.com/literature/ds/ds_nios_cpu.pdf, ver. 2.2, 2004

[And01] Anderson L.: "LCP Specification", RFC 3036, http://www.rfc-archive.org/getrfc.php?rfc=3036, 2001

[Ant07] Anti-Phishing Working Group: "Phishing Activity Trends", http://www.antiphishing.org, 2007

[Apa08] "The Apache Spam Assasin Project", http://apamassassin.apache.org, 2008

[Awe99] Aweya, J.: "IP Router Architectures: An Overview" Nortel Networks, http://www.cs.virginia.edu/~cs757/papers/awey99.pdf, 1999

[Bae92] Bäck T.: "The interaction of mutation rate, selection, and self-adaption within a genetic algorithm", Parallel Problem Solving from Nature II, Elsevier, Amsterdam, S. 85-94, 1992

[BK90] Broder A.Z., Karlin A.R.: "Multilevel Adaptive Hashing", Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms, 1990, S. 43-53

[Blo70] Bloom B.: “Space/Time Trade-offs in hash coding with allowable errors”, Communications of the ACM, vol. 13, S. 422-426, 1970

[BM01] Broeder A., Mitzenmacher M.: “Using Multiple Hash Functions to Improve IP Lookups”, Proceedings of the IEEE INFOCOM, S. 1454-1463, 2001

[Bon04] Boneh D.: “The Difficulties of Tracing Spam E-mail”, Technical Report Stanford University, http://spamassassin.apache.org, 2004

[BR03] Barreto P.S.L.M., Rijmen V.: “The Whirlpool Hashing Function”, 2003

[Bro04] Bronson J.: “Protecting Your Network from ARP Spoofing-Based Attacks”, Foundstone Inc., 2004

[BRS66] Bremerman H., Rogson M., Salaff S.: “Global properties of evolution processes”, Natural Automata and Useful Simulations, S. 3–41, 1966

[BS96] Bäck, T. Schütz M.: “Intelligent mutation rate control in canonical genetic algorithms”, Proceedings of the 9th International Symposium on Foundations of Intelligent Systems, S. 158–167, 1996

[Cal05] CallingID: “Why you need CallingID”, White Paper, http://callingid.com, 2005

[CC05] Chop, N.E., Calvert D.: “The Chopper Genetic Algorithm a Variable Population GA”, Artificial Neuronal Networks in Engineering (ANNIE), 2005

[CEH+06] Cranor L., Egelman S., Hong J., Zhang Y.: “Phinding Phish: An Evaluation of Anti-Phishing Toolbars”, Carnegie Mellon Universität, Technical Report CMU-CyLab-06-018, 2006

[CGE+04] Chiruvolu G., Ge A., Elie-Dit-Cosaque D., Ali M.: “Issues and Approaches on Extending Ethernet Beyond LANs”, IEEE Communications Magazine, S. 80-86, 2004

[CH06] Carpinter J., Hunt R.: “Tightening the Net: A Review of Current and Next Generation Spam Filtering Tools”, Computer & Security, vol. 25, S. 566-578, 2006

[Cis06] Cisco Inc.: “Resolve IP fragmentation, MTU, MSS, and PMTUD issues with GRE and IPSEC”, White Paper, http://www.cisco.com/application/pdf/paws/25885/pmtud_ipfrag.pdf, 2006

[Cis08] Cisco Inc.: “Cisco XR 12000 Series Routers”, Datasheet, http://www.hardware.com/product.asp?id=GSR12/60#, 2008

[CMX+07] Cho, S., Martin, J.R., Xu, R., Hammoud, M.H., Melhem, R.: “CA-RAM: A high-performance memory substrate for search-intensive applications”, Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2007), S. 230-241, 2007

[Con81] Connet G.: “Expected Length of the Longest Probe Sequence in Hash Code Searching” Journal of the ACM, vol. 28, 1981, S. 289-304

[CP00] Chieueh, T., Prashant P.: “Cache Memory Design for Network Processor”, Proceedings of the 6th International Symposium on High-Performance Computer Architecture, 2000

[CP99] Chieueh, T., Prashant P.: “High-Performance IP Routing Table Lookup Using CPU Caching”, Proceedings of the IEEE INFOCOM, S. 1421-1428, 1999

[CW81] Carter L., Wegman M.: “New Hash Functions and their Use in Authentication and Set Equality”, Journal of Computer and Systems Sience, vol. 22, S. 265-279, 1981

[Dan06] Danielis P.: “Realisierung und Implementierung eines Algorithmus zur Echtzeit-Mustererkennung in einem Ethernet-Datenstrom”, Diplomarbeit, Universität Rostock, 2006

[Dar59] Darwin, C.: “The Origin of Species”, John Murray, 1859

[DBB+05] Degioanni L., Baldi M., Buffa D., Risso F., Stirano F., Varanni G.: “Network Virtual Machine (NetVM): A New Architecture for Efficient and Portable Packet Processing Applications”, Proceedings of the 8th International Conference on Telecommunications (ConTEL), S. 15-17, 2005

[DeJ75] De Jong, K.A.: “Analysis of the Behavior of a Class of Genetic Adaptive Systems”, Ph.D. Thesis, University of Michigan, 1975

[Der74] Dertouzos M.: “Control Robotics: The Procedural Control of Physical Processes”, Proceedings of IFIP Congress, S.807-813, 1974

[DHV01] Deepakumara, J. and Heys, H.M., Venkatesan, R.: “FPGA implementation of MD5 hash algorithm”, Canadian Conference on Electrical and Computer Engineering, S. 919-924, 2001

[DIX82] Digtal, Intel, Xerox: “The Ethernet – A Local Area Network: Data Link Layer and Physical Layer Specifications”, Version 2.0, 1982

[DKW+08a] Danielis P., Kubisch S., Widiger H., Schulz J., Timmermann D., Bahls T., Duchow D.: “IPclip - An Innovative Mechanism to Reestablish Trust-by-Wire in Packet-switched IP Networks”, 3. Essener Workshop "Neue Herausforderungen in der Netzsicherheit", 2008

[DKW+08b] Danielis P., Kubisch S., Widiger H., Schulz J., Duchow D., Bahls T., Timmermann D., Lange C.: “Trust-by-Wire in Packet-switched IP Networks: Calling Line Identification Presentation for IP”, Design, Automation and Test in Europe Conference and Exhibition (DATE 2008) – University Booth, 2008

[DLT98] Damiani E., Liberali, V., Tettamanzi, A. G. B.: “Evolutionary Design of Hashing Function Circuits Using an FPGA”, Lecture Notes in Computer Scinence, vol. 1478, S. 36-46, 1998

[DLT99] Damiani E., Liberali, V., Tettamanzi, A. G. B.: “FPGA based hash circuit synthesis with evolutionary algorithms”, IEICE Transactions on Fundamentals, vol. E82-A, no. 9, S. 1888-1896, 1999

[Dro97] Droms R.: “Dynamic Host Configuration Protocol”, RFC 2131, http://www.rfc-archive.org/getrfc.php?rfc=2131, 1997

[DSL03a] DSL Forum: “Multi-Service Architecture & Framework Requirements”, Technical Report TR-058, 2003

[DSL03b] DSL Forum: “DSL Evolution - Architecture Requirements for the Support of QoS-Enabled IP Services”, Technical Report TR-059, 2003

[DT99] Damiani E., Tettamanzi, A. G. B.: “On-Line Evolution of FPGA-Based Circuits: A Case Study on Hash Functions”, Proceedings of the 1st NASA/DoD Workshop on Evolvable Hardware EH99, S. 33-36, 1999

[Egg07] Eggendorfer T.: “Reducing Spam to 20% of its Original Value with an SMTP Tat Pit Simulator”, Proceedings of the 2007 MIT Spam Conference, 2007

[ES03] Eiben, A.E., Smith, J.E.: “Introduction to Evolutionary Computing”, Springer, Berlin, 2003

[FC74] Frank, H., Chou W.: “Network Properties of the ARPA Computer Network”, Networks, vol. 4, S. 213-239, 1974

[FCA+00] Fan L., Cao P., Almeida J., Broder A.: “Summary Cache: a Scalable Wide-area Web Cache Sharing Protocol”, IEEE/ACM Transactions on Networking, vol. 8, S. 281-293, 2000

[Fel88] Feldmeier, D.C.: “Improving gateway performance with a routing-table cache”, Proceedings. of the 7th IEEE INFOCOM, S. 287-307, 1988

[FKS84] Fredman, M.L., Komlós J., Szemerédi E.: “Storing a sparse tabel with O(1) worst case access time”, Journal of the ACM, vol 31, no39, S. 538-544, 1984

[Fle82] Fletcher J.G.: “An Arithmetic Checksum for Serial Transmissions”, IEEE Transactions on Communications, vol. 30, S. 247-252, 1982

[FMM+84] Finlayson R., Mann T., Mogul J., Theimer M.: “A Reverse Address Resolution Protocol”, RFC 903, http://www.rfc-archive.org/ getrfc.php?rfc=903, 1984

[Fog94] Fogel D. B.: “Evolutionary Programming: An Introduction and Some Current Directions“, Statistics and Computing, vol. 4, S.113-129, 1994

[GCA+04] Gomes L.H., Cazita C., Almeida J.M., Almeida V., Wagner M. Jr.: “Characterizing Spam Traffic”, Proceedings fo the ACM SIGCOM Internet Measurement Conferrence (IMC) , S. 356-369, 2004

[Ger08] German Internet Exchange: “DE-CIX Yearly Graph (800d)”, http://www.de-cix.net/content/network.html, 2008

[GKL+04] Grünewald M., Kastens U., Le D.K., Niemann J., Porrmann M., Rückert U., Thies M., Slowik A.: “Network applications driven instruction set extensions for embedded processing clusters”, International Conference on Parallel Computing in Electrical Engineering (PARELEC), S. 209-214, 2004

[GLM98] Gupta P., Lin S., McKeown N.: “Routing Lookups in Hardware at Memory Access Speeds”, Proceedings of IEEE INFOCOM, S. 1240-1247, 1998

[Gol89] Goldberg D. E.: “Genetic Algorithms in Search, Optimization, and Machine Learning”, Addison-Wesley, Reading, 1989

[Gre86] Grefenstette, J.J.: “Optimization of Control Parameters for Genetic Algorithms”, IEEE Transaction on Systems, Man, and Cybernetics, vol 16, no. 1, S. 122-128, 1986

[GT07] Greenwood, G. W., Tyrrell, A.: “Introduction to Evolvable Hardware”, Whiley & Sons, Hobocken, 2007

[Gwe68] Gwehenberger G.: “Anwendung einer binären Verweiskettenmethode beim Aufbau von Listen“, Elektronische Rechenanlagen, S. 223-226, 1968.

[HAG06] Huntley, C., Antonova, G., Guinand P.: “Effect of Hash Collisions on the Performance of LAN Switching Devices and Networks”, Proceedings of the 31st Annual IEEE Conference on Local Computer Networks (LCN), S. 280-284, 2006

[Hal05] Halsal, F.: “Computer Networking and the Internet”, Addison Wesley, Harlow, 2005

[HBB+05] Hruby T., de Brujn W., Bos J., Crestea M.L., Xu L.: “Lessions Learned in Developing a Flexible Packet Processor for High Speed Links”, Technical Report IR-CS-016, Vrije Universiteit Amsterdam, 2005

[Hed88] Hedrick, C.: “Routing Information Protocol”, RFC 1058, http://www.rfc-archive.org/getrfc.php?rfc=1058, 1988

[Hei99a] Heinanen J.: “A Single Rate Three Color Marker”, RFC 2697, http://www.rfc-archive.org/getrfc.php?rfc=2697, 1999

[Hei99b] Heinanen J.: “A Two Rate Three Color Marker”, RFC 2698, http://www.rfc-archive.org/getrfc.php?rfc=2698, 1999

[HGT99] Hildebrandt J., Golatowski F, Timmermann D.: “Scheduling Coprocessor for Enhanced Least-Laxity-First Scheduling in Hard Real-Time Systems”, Proceedings of the 11th Euromicro Conference on Real Time Systems, S. 208-215, 1999

[Hil04] Hildebrandt J.: “Hardware basiertes Taskscheduling für Echtzeit-Systeme”, Dissertation, Universität Rostock, 2004

[Hir96] Hirst A. J.: “Notes on the Evolution of Adaptive Hardware”, Proceedings of the 2nd International Conference on Adaptive Computer Engineered Design (ACEDC'96), 1996

[HLP06] High Level Policy Task Force on VoIP: “Common Position on VoIP (Draft)”, Technical Report ERG (07) 56 Rev 1, European Regulators Group (ERG), 2007

[HLT+03] Horta E.L., Lockwood J.W., Taylor D.E., Parlour D.: “Dynamic hardware plugins in an FPGA with partial run-time reconfiguration”, Proceedings of the ACM IEEE 39th Design Automation Conference, S. 343-348, 2002

[HM91] Hesser J., Männer R.: “Towards an optimal mutation probability for genetic algorithms”, Parallel Problem Solving from Nature, Lecture Notes in Computer Science. vol 496, Springer, 1991

[Hol75] Holland, J.H.: “Adaption in Natural and Artificial Systems”, The University of Michigan Press, Ann Arbor, 1975

[HSS+99] Handley M., Schulzrinne H., Schooler E., Rosenberg J.: “SIP: Session Initialisation Protocol”, RFC 2543, http://www.rfc-archive.org/getrfc.php?rfc=2543, 1999

[Hus01] Huston G.: “Analyzing the Internet’s BGP Routing Table”, The Internet Protocol Journal, vol. 4, no 1, 2001

[Hus08] Huston G.: “Analysis of BGP Routing Table Dynamics”, Website, http://bgp.potaroo.net/, 2008

[HZ99] Huang N., Zhao S.: “A Novel IP-Routing Lookup Scheme and Hardware Architecture for Mulitgigabit Switching Routers”, IEEE Journal on selected Areas in Communication, vol. 17, no. 6, S. 1093-1104, 1999

[IAN07] Internet Assigned Numbers Authority (IANA): “IP Option Numbers”, 2007

[IEE04] IEEE: “802.3ah - IEEE Standard for Information technology-Telecommunications and information exchange between systems- Local and metropolitan area networks- Specific requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications Amendment: Media Access Control Parameters, Physical Layers, and Management Parameters for Subscriber Access Networks”, http://ieeexplore.ieee.org/xpls/abs_all.jsp?tp=&isnumber=29503&arnumber=1337489&punumber=9283

[IEE05] IEEE Computer Society: “Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specification”, IEEE Standard for Information technology, New York, 2005

[IEE06a] IEEE: “802.1q – Virtual LANs”, http://www.ieee802.org/1/pages/ 802.1Q.html, 2006

[IEE06b] IEEE: “802.1ad – Provider Bridge”, http://www.ieee802.org/1/pages/802.1ad.html, 2006

[Int01a] Intel: “Intel IXP1200 Network Processor – Hardware Reference Manual”, http://www.ece.udel.edu/~fchen/dloads/doc/ixp1200/IXP1200_HW_Ref_Manual.pdf, 2001

[Int01b] Intel: “Intel IXP1200 Network Processor – Microcode Software Reference Manual”, http://www.ece.udel.edu/~fchen/dloads/doc/ixp1200/IXP1200_SW_Ref_Manual.pdf, 2001

[Int03] Intel: “Intel IXP2400 Network Processor”, Product Brief, http://www.intel.com/design/network/prodbrf/27905302.pdf, 2003

[Int06a] Intel: “Dual-Core Intel Xeon Processor 7100 Series Data Sheet”, White Paper, Reference Number 314553, Rev. 002, http://download.intel.com/design/Xeon/datashts/30675402.pdf, 2006

[Int06b] Intel: “Intel IXP45X and Intel IXP46X Product Line of Network Processors”, Datasheet, http://download.intel.com/design/network/datashts/ 30626104.pdf, 2006

[ITU03] ITU Telecommunication Standardization Sector (ITU-T): “Gigabit-capable Passive Optical Networks (GPON): General characteristics”, Recommendation G.984.1, 2003

[ITU05] ITU Telecommunication Standardization Sector (ITU-T): “Broadband optical access systems based on passive optical networks (PON)”, Recommendation G.983.1, 2005

[ITU07] International Telecommunication Union High Level Group on Cybercrime: “ITU Global Cybercrime Agenda”, http://www.itu.int, 2007

[Jai92] Jain R.: “A Comparision of Hashing Schemes for Address Lookup in Computer Networks”, IEEE Transactions on Communications, vol. 40, no. 10, S. 1570-1573, 1992

[Joh09] Johannsen, W.L.: “Limitations of Natural Selection on Pure Lines”, 1909

[JP08] Jinang W., Prasanna V. K.: „Parallel IP Lookup using Mulitple SRAM-based Pipelines”, Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2008

[JW02] Jung V., Warnecke H.: “Handbuch für die Telekommunikation”, Springer, Heidelberg, 2002

[KB96] Keane, A.J., Browne, S.M.: “The Design of a Satellite Boom with Enhanced Vibration Performance using Genetic Algorithm Techniques”, Proceedings of the conference on Adaptive Computing in Engineering Design and Control, S. 107-113, 1996

[KD73] Krishnaiyer R., Donovan J.C.: „Shift Register Generator of Pseudorandom Binary Sequences”, Computer Design, vol. 12, S. 69-74, 1973

[Kle03] Kleinrock L.: “An Internet Vision: The Invisible Global Infrastructure”, Ad Hoc Networks, vol. 1, no. 1, S. 3-11, 2003

[KLS+04] Kastens U., Le D.K., Slowik A., Thies M.: “Feedback driven instruction-set extensions”, Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES), 2004

[Knu05] Knuth D.E.: “The Art of Computer Programming – Searching and Sorting”, Adison-Wesley, Boston, 2005

[Koz94] Koza J. R.: “Genetic Programming as a Means for Programming Computers by Natural Selection”, Statistics and Computing, vol. 4, S. 87-112, 1994

[KR06] Kuon I., Rose J.: “Measuring the Gap between FPGAs and ASICs”, Proceedings of the ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays, S. 21-30, 2006

[Krö06] Kröger P.: “Entwicklung eines generischen und variablen Paket Prozessor-Designs für Access Netzwerke aus existenten funktionalen Elementen”, Diplomarbeit Universität Rostock, 2006

[Kub08] Kubisch S.: “Architekturen für Ethernet-basierte Teilnehmerzugangsnetzwerke und deren Umsetzung in Hardware”, Dissertation, Universität Rostock, 2008

[KWD+06] Kubisch, S., Widiger, H., Duchow, D., Bahls, T., Timmermann, D.: “Wirespeed MAC Address Translation and Traffic Management in Access Networks”, Proceedings of the World Telecommunication Congress (WTC), 2006

[KWD+07] Kubisch S., Widiger H., Danielis P. Duchow D., Bahls T. Timmerann D., Lange C., Röwer O.: “Configuration Tool and FPGA-Prototype of a Hardware Packet Processing System”, Design, Automation and Test in Europe Conference and Exhibition (DATE 2007) – University Booth, 2007

[KWD+08a] Kubisch S., Widiger H., Danielis P., Schulz J., Timmermann D., Bahls T., Duchow D.: “Trust-by-Wire in Packet-switched IP Networks: Calling Line Identification Presentation for IP”, Proceedings of the 1st ITU-T Kaleidoscope Conference: Innovations in Next Generation Networks - Future Network and Services, S. 375-382, 2008

[KWD+08b] Kubisch S., Widiger H., Danielis P., Schulz J., Timmermann D., Bahls T., Duchow D.: “Countering Phishing Threats with Trust-by-Wire in Packet-switched IP Networks - A Conceptual Framework”, Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS) – 4th International Workshop on Security in Systems and Networks (SSN 2008), 2008

[KWD+08c] Kubisch S., Widiger H., Danielis P., Schulz J., Timmermann D., Bahls T., Duchow D.: “Complementing E-Mails with Distinct, Geographic Location Information in Packet-switched IP Networks”, MIT 2008 Spam Conference, 2008

[KWT+07] Kubisch S., Widiger H., Timmermann D., Duchow D., Bahls T.: “sMAT - A Simplified MAC Address Translation Scheme” Proceedings of the 15th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN 2007), 2007

[Lat00] Lattice Semiconductor Corp.: “ispPAC10 In-System Programmable Analog Circuit”, Data Sheet, http://www.fm.vslib.cz/~kes/pages/ez_prj/pac10o.pdf, 2000

[LBF+01] Langeheine J., Becker J., Folling F., Meier, K, Schemmel J.: “Initial Studies of a new VLSI Field Programmable Transistor Array”, 4th International Conference on Evolvable Systems: From Biology to Hardware, S. 62-73, 2001

[Lea07] Leavitt N.: “Vendors Fight Spam’s Sudden Rise”, IEEE Computer, vol. 40, no. 3, S. 16-19, 2007

[LHJ03] Lim H., Seo J., Jung Y.: “High Speed IP Address Lookup Architecture Using Hashing”, IEEE Communication Letters, vol. 7, no. 10, S. 502-504, 2001

[LL03] Lai W., Lea C.-T.: “A Programmable State Machine Architecture for Packet Processing”, IEEE Micro, vol. 23, no. 4, S. 32-42, 2003

[LL73] Liu C.L., Layland J.W.: “Scheduling Algorithms for Multiprogramming in Hard Real-Time Environments”, Journal of the Association for Computing Machinery (ACM) vol. 20, no. 1, S. 46-61, 1973

[LNP+02] Langen D., Niemann J., Porrmann M., Kalte H., Rückert U.: “Implementation of a RISC Processor Core for SoC Design – FPGA Prototype vs. ASIC Implementation”, Proceedings of the IEEE-Workshop: Heterogeneous reconfigurable Systems on Chip (SoC). 2002

[LNT+01] Lockwood J.W., Naufel N., Turner J.S., Taylor D.E.: “Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX)”, Proceedings of the 9th International Symposium on Field Programmable Gate Arrays, S. 87-93, 2001

[LTT00] Lockwood J.W., Turner J.S., Taylor D.E.: “Field Programmable Port Extender (FPX) for Distributed Routing and Queuing”, Proceedings of the 8th International Symposium on Field Programmable Gate Arrays, S. 137-144, 2000

[Loc85] Locke C. D.: “Best-Effort Decision Making for Real Time Scheduling”, Ph.D. Thesis, Carnegie-Mellon University, 1985

[LPS08] Lallet J., Pillement S., Sentieys O.: “Efficient dynamic reconfiguration for multi-context embedded FPGAs”, Proceedings of the 21st Annual Symposium on Intgrated Circuits and System Design, S. 210-215, 2008

[Man00] Mandeville R.: “Benchmarking Methodology for LAN Switching Devices”, RFC 2889, http://www.rfc-archive.org/getrfc.php?rfc=2889, 2000

[Mar05] Martini L.: “Encapsulation Method for Transport of Layer 2 Frames Over IP and MPLS Networks”, internet draft, 2005

[MB00] McKnight L. W., Boroumand J.: “Pricing Internet Services: Approaches and Challanges”, IEEE Computer, S. 128-129, 2000

[McD08] McDonald E.J.: “Runtime FPGA Partial Reconfiguration”, IEEE Aerospace and Electronic Systems Magazine, vol 23, no. 7, S. 10-15, 2008

[MF02] Mehrotra P., Frazon P.D.: “Novel Hardware Architecture for Fast Address Lookups”, IEEE Communications Magazine, S. 105-110, 2002

[Mor68] Morrison D.R.: “PATRICIA – Practical Algorithm to Retrieve Information coded in Alphanumeric”, Journal of the ACM, vol. 15, no. 14, S. 514-534, 1968

[Mot98] Motorola: “M-Core Reference Manual”, 1998

[MOW+07] Meitinger M., Ohlendorf R., Wild T., Herkersdorf A.: “A Programmable Stream Processing Engine for Packet Manipulation in Network Processors”, Proceedings of the IEEE Computer Society Annual Symposium on VLSI (IVLSI), S. 259-264, 2007

[Moy94] Moy, J.: “Open Shortest Path First Routing Protocol (Version 2)”, RFC 1583, http://www.rfc-archive.org/getrfc.php?rfc=1583, 1994

[MRS+05] Mintz-Habib M., Rawat A. Schulzrinne H, Wu X.: “A VoIP Emergency Services Architecture and Prototype”, Proceedings of the 14th International Conference on Computer Communications and Networks, 2005,

[MS98] McDysan D. E., Spohn D.L.: “ATM Theory and Applications”, McGraw-Hill, 1998

[Mue92] Mühlenbein, H.: “How genetic algorithms really work: I. mutation and hillclimbing”, Parallel Problem Solving from Nature II, S. 15–25, 1992

[Nat02] National Institute of Standards and Technology (NIST): “Secure Hash Standard (SHS) (FIPS PUB 180-2)”, Federal Information, 2002

[NBB+98] Nichols K., Blake S., Baker F., Black D.: “Definition of the Differentiated Services Field (DS Field) in IPv4 and IPv6 Headers”, RFC 2474, http://www.ietf.org/rfc/rfc2474.txt, 1998

[NEN05] National Emergency Number Association (NENA), VoIP-Packet Technical Committee: “Interim VoIP Architecture for Enhanced 9-1-1 Services”, Technical Report 08-001, Issue 1, 2005

[NK98] Nilsson S., Karlsson G.: “Fast address lookup for Internet routers”, Broadband Communications: The Future ofTelecommunications, S. 11-22, 1998

[NME02] National Marine Electronics Association (NMEA): “NMEA 0183 Standard”, 2002

[NML+97] Newman, P., Minshall, G., Lyon, T., Huston, L.: “IP Switching and Gigabit Routers”, IEEE Communication Magazine, 1997

[NNL06] Newport Networks Ltd.: “Emergency Call Handling in VoIP Networks”, White Paper, http://www.newport-networks.com/cust-docs/89-ECH.pdf, 2006

[NPP+07] Niemann, J., Puttmann C., Porrmann M., Rückert U.: “Resource efficiency of the GigaNetIC chip multiprocessor architecture”, Journal of System Architecture, vol. 53, no. 5-6, S. 285-299, 2007

[NPR05] Niemann J., Porrmann M., Rückert U.: “A Scalable Parallel SoC Architecture for Network Processors”, IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2005), 2005

[NPS+05] Niemann, J., Porrmann M., Sauer C., Rückert U.: “An Evaluation of the Scalable GigaNetIC Architecture for Access Networks”, Proceedings of the Advanced Networking and Communications Hardware Workshop (ANCHOR), 2005

[NWC+05] Nie X., Wilson D.J., Cornet J., Damm G., Zhao Y.: “IP Address Lookup Using A Dynamic Hash Function”, Proceedings of the Canadian Conference on Electrical and Computer Engineering, S. 1642-1647, 2005

[Och00] Ochoa G.: “Error Thresholds and Optimal Mutations Rates in Genetic Algorithms”, PhD Thesis, University of Sussex, 2000

[OHW05] Ohlendorf R., Herkersdorf A., Wild T.: “FlexPath NP – A Network Processor Concept with Application-Driven Flexible Processing Paths”, Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISS), S. 279-284, 2005

[ONI+06] Okuno M., Nishimura S., Ishida S., Nishi H.: “Cache-based Network Processor Architecture: Evaluation with Real Network Traffic”, IEICE Transactions on Electronics, vol. E89, no. 11, S. 1620-1628, 2006

[Par05] Park S.: “Implementation of Next Generation VDSL Networks in Metro Ethernet Backbone Environments”, Proceedings of the Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/E-Learning on Telecommunications Workshop (AICT/SAPIR/ ELETE'05) , S. 21-26, 2005

[Par96] Partidge C.: “Locality and Route Caches”, NFS Workshop on Internet Statistics, Measurement, and Analysis, 1996

[PD03] Peterson, L.L., Davie B.S.: “Computernetze – Eine systemorientierte Einführung”, Elsevier, Heidelberg, 2003

[Plu82] Plummer D.C.: “An Ethernet Address Resolution Protocol”, RFC 826, http://www.rfc-archive.org/getrfc.php?rfc=826, 1982

[PLW+03] Pao D., Liu C., Wu A., Yeung L., Chan K.S.: “Efficient Hardware Architecture for Fast IP Address Lookup”, IEEE Journal on Selected Areas in Communications, 2003

[Pos82] Postel J.B.: “Simple Mail Transfer Protocol”, RFC 821, http://www.rfc-archive.org/getrfc.php?rfc=821, 1982

[PSL04] Polk J., Schnizlein J., Lisner M.: “Dynamic Host Configuration Protocol Option for Coordinate-based Location Configuration Information”, RFC 3825, http://www.rfc-archive.org/getrfc.php?rfc=3825, 2004

[PVN+04] Papaefstathiou I., Vlachos K., Nikolaou N., Zervos N., Lawrence V.B.: “Packet Processing Acceleration with a 3-Stage Programmable Pipeline Engine”, IEEE Communication Letters, vol. 8, no. 3, S. 183-185, 2004

[PZW+07] Parameswaran M., Zhao X., Whinston A.B., Fang F.: “Reengineering the Internet for Better Security”, IEEE Computer, vol. 40, no. 1, S. 40-44, 2007

[RDS04] Rooney, J.J., Delgado-Frias J.G., Summerville D.H.: “Associative ternary cache for IP routing”, IEE Computers and Digital Tech. Journal, vol. 151, no 6, S. 409-416, 2004

[Rec94] Rechenberg, I.: “Evolutionsstrategie ‘94”, Frommann Holzboog, 1994

[Rek95] Rekhter, Y.: “CIDR and Classful Routing”, RFC 1817, http://www.rfc-archive.org/getrfc.php?rfc=1817, 1995

[Rek+06] Rekhter, Y. et al.: “A Border Gateway Protocol 4 (BGP-4)”, RFC 4271, http://www.rfc-archive.org/getrfc.php?rfc=4271, 2006

[RFB97] Ramakrishna M.V., Fu E., Bahcekapili E.: “Efficient Hardware Hashing Functions for High Performance Computers”, IEEE Transactions on Computers, vol. 46, S. 1378-1381, 1997

[Rip08] Ripe.net: “Routing Information Service (RIS) Raw Data”, BGP Routinginformation DE-CIX, http://www.ripe.net/projects/ris/rawdata.html, 2008

[Riv92] Rivest R.: “The MD5 Message-Digest Algorithm”, RFC 1321, http://www.rfc-archive.org/getrfc.php?rfc=1321, 1992

[Rou07] University of Oregon: “Route Views Project”, http://www.routeviews.org/, 2007

[RP06] Rosen B., Plok J.: “Best Current Practice for Communications Services in support of Emergency Calling”, Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-ecritphonebcp-01.txt, 2006

[RSP+08] Rosen B., Schulzrinne H., Polk J., Newton A.: “Framework for Emergency Calling using Internet Multimedia”, Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-ecrit-framework-05.txt, 2008

[RTF+01] Rosen D. et. al.: ”MPLS Label Stack Encoding”, RFC 3032, http://www.rfc-archive.org/getrfc.php?rfc=3032, 2001

[RVC01] Rosen D., Viswanathan A., Callon R.: ”Multiprotocol Label Switching Architecture”, RFC 3031, http://www.rfc-archive.org/getrfc.php?rfc=3031, 2001

[San03] Santitoro R.: “Metro Ethernet Services – A Technical Overview”, White Paper, 2003

[SCE+89] Schaffer J.D., Caruana R., Eshelman L.J., Das R.: “A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization”, Proceedings of the 3rd International Conference on Genetic Algorithms, S. 51-60, 1989

[Sch81] Schwefel H. P.: “Numerical Optimization of Computer Models”, Whiley & Sons, Hoboken, 1981

[SCH97] Schulzrinne H.: “Dynamic Host Configuration Protocol (DHCPv4 and DHCPv6) Option for Civic Addresses Configuration Information”, RFC 4776, http://www.rfc-archive.org/getrfc.php?rfc=4776, 2006

[SCK+06] Sill F., Cornelius C., Kubisch S., Timmermann D.: ”Mixed Gates: Leakage Reduction techniques applied to Switches for Networks-on-Chip”, Proceedings of the 2nd International Workshop on Reconfigurable Communication-centric SoCs (ReCoSoC), S. 76-82, 2006

[SDT+05] Soong H., Dharmapurikar S., Turner J. S., Lockwood J.: “Fast Hash Table Lookup Using Extended Bloom Filter: An Aid to Network Processing”, Proceedings of the annual ACM SIGCOMM, S. 181-192, 2005

[SGS05] Sauer C., Gries M., Sonntag S.: “Modular Reference Implementation of an IP-DSLAM”, Proceedings of the 10th IEEE Symposium on Computer an Communications (ISCC), S. 191-198, 2005

[Sil02] Silicore: “WHISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores”, 2002

[SKA+04] Stoica A., Keymeulen D., Arslan T., Duong V.: “Self-Recovery Experiments in Extreme Environments Using a Field Programmable Transistor Array”, Proceedings of the 2004 IEEE International Conference on Field-Programmable Technology, S. 9-15, 2004

[Skl93] Sklower K.: “A tree-based packet routing table for Berkeley Unix”, Technical report, University of California, 1993

[SMG03] Shi, W., MacGregor, M. H., Gburzynski, P.: “Traffic locality characteristics in a parallel forwarding system”, International Journal of Communication Systems, vol 16, no 9, pp 823-839, 2003

[SP01] Siduh R., Prasanna V.K.: “Fast Regular Expression Matching using FPGAs”, Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM01), S.227-238, 2001

[SR06] Salim R., Rao G.S.V.R.K.: “Software-based Packet Classification in Network Intrusion Detection System using Network Processor”, Prodeedings of IEEE Region 10 Conference (TENCON), S. 1-4, 2006

[Sri99] Srinivasan V.: “Fast and efficient Internet lookups,” PhD Thesis, Washington University, 1999.

[SV98] Srinivasan V., Varghese G.: “Faster IP Lookups using Controlled Prefix Expansion”, ACM SIGMETRICS Performance Evaluation, vol. 26, S. 1-10, 1998

[SWT06] Salomon R., Widiger H., Tockhorn A.: “Rapid Evolution of Time-efficient Packet Classifiers”, Proceedings of the IEEE World Congress on Computational Intelligence (WCCI 2006), 2006, S. 2793-2799

[Sym08] Symantec Inc.: “The State of Spam – A Monthly Report”, Technical Report, 2008

[SZK+06] Stoica A., Zebulum, R., Keymeulen D., Ramesham R., Neff J., Kartkoon S.: "Temperature-Adaptive Circuits on Reconfigurable Analog Arrays", 1st NASA/ESA Conference on Adaptive Hardware and Systems (AHS), S. 28-31, 2006

[Tan95] Tanenbaum A.S. "Moderne Betriebssysteme", Hanser, 1995

[TH99] Tufte G., Haddow P. C.: "Prototyping a GA pipeline for Complete Hardware Evolution", Proceedings of the 1st NASA/DoD Workshop on Evolvable Hardware EH99, S. 143-150, 1999

[Toc07] Tockhorn A.: "Leistungsoptimierung eines Paket-Klassifizierers auf Basis einer evolvierbaren Hardwareimplementierung", Diplomarbeit, Universtiät Rostock, 2007

[Usc81a] Information Science Institute, University of Southern California: "Transmission Control Protocol – DARPA Internet Program Protocol Specification", RFC 793, http://www.rfc-archive.org/getrfc.php?rfc=793, 1981

[Usc81b] Information Science Institute, University of Southern California: "Internet Protocol – DARPA Internet Program Protocol Specification", RFC 791, http://www.rfc-archive.org/getrfc.php?rfc=791, 1981

[Var05] Varghese, G.: "Network Algorithmics", Morgan Kaufman, San Francisco, 2005

[Wal00] Waldvogel M.: "Fast Longest Prefix Matching: Algorihms, Analysis, and Applications", Ph.D. Thesis, ETH Zürich, http://e-collection.ethbib.ethz.ch/show?type=diss&nr=13266, 2000

[Wei02] Weicker, K.: "Evolutionäre Algorithmen", Teubner, Stuttgart, 2002

[Whi94] Whitley D.: "A Genetic Algorithm Tutorial", Statistics and Computing, vol. 4, S. 65-85, 1994

[WHT05] Widiger H., Handy M., Timmermann D.: "Packet Classification with Evolvable Hardware Hash Functions", Proceedings of the 8th EUROMICRO Conference on Digital System Design (DSD 2005), 2005

[Wil01] Williams, J.: "Architectures for Network Processing", International Symposium on VLSI Technology, Systems, and Applications, 2001

[WKD+06] Widiger, H., Kubisch S., Duchow, D., Bahls, T., Timmermann, D.: "A Simplified Cost-Effective MPLS Labeling Architecture for Access Networks", Proceedings of the World Telecommunication Congress (WTC), 2006

[WKD+08] Widiger H., Kubisch S., Danielis P., Schulz J., Timmermann D., Bahls T., Duchow D.: "IPclip: An Architecture to restore Trust-by-Wire in Packet-switched Networks", Proceedings of the 33rd Annual IEEE Conference on Local Computer Networks (LCN), S. 312-319 2008

[WKT+06] Widiger, H., Kubisch S., Timmermann D., Bahls, T.: "A Hardware Solution for MAT, MPLS-UNI, and TM in Access Networks", Proceedings of the 31st Annual IEEE Conference on Local Computer Networks (LCN), S. 272-279, 2006

[WKT07] Widiger H., Kubisch S., Timmermann D.: "A Structural Architecture for HW Packet Processing", Proceedings of the 2007 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, 2007, S. 363-366

[Won05] Wong M.W.: "Sender Authentication – What to do", White Paper, NGS Next Generation Security Ltd., 2005

[WST06] Widiger H., Salomon R., Timmermann D.: "Packet Classification with Evolvable Hardware Hash Functions", Proceedings of the 2nd International Workshop on Biologically Inspired Approaches to Advanced Information Technology (BioADIT), 2006, S. 64-79

[WTS+07] Widiger H., Tockhorn A., Salomon R., Timmermann D.: "Acceleration of the Evolution of Evolvable Hardware-based Packet Classifiers", Proceedings of the IEEE SSCI 2007 - Workshop on Evolvable and Adaptive Hardware (WEAH 2007), 2007, S. 27-34

[WTT08] Widiger H., Tockhorn A., Timmermann D.: "On the Impact of Caching for high Performance Packet Classifiers", IEEE Global Communications Conference (GlobeCOM 2008), 2008

[WVT+01] Waldvogel M., Varghese G., Turner J., Plattner B.: "Scalable High-Speed Prefix Matching", ACM Transcations on Computer Systems, vol. 19, 2001, S. 440-482

[WVT+97] Waldvogel M., Varghese G., Turner J., Plattner B.: "Scalable High Speed IP Routing Lookups", Proceedings of the ACM SIGCOM'97, 1997, S.25-36

[Xil04a] Xilinx Inc.: "Virtex-4 User Guide", http://www.xilinx.com/support/documentation/user_guides/ug070.pdf, ver. 1.1, 2004

[Xil04b] Xilinx Inc.: "Virtex-4 Product Selection", http://www.xilinx.com/devices, 2004

[Xil06] Xilinx Inc.: "MicroBlaze Processor Reference Guide", http://www.xilinx.com/support/documentation/sw_manuals/edk92i_mb_ref_guide.pdf, ver.8.0, 2006

[Xil07a] Xilinx Inc.: "Virtex-4 Family Overwiev", Product Specification, http://xilinx.com. ver. 3.0, 2007

[Xil07b] Xilinx Inc.: "Virtex-5 Product Table", http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex5/v5product_table.pdf, 2007

[Xil08] Xilinx Inc.: "ML405 Evaluation Platform User Guide", http://www.xilinx.com/support/documentation/boards_and_kits/ug210.pdf, ver. 1.5.1, 2008

[Xil96] Xilinx Inc.: “Efficient Shift Registers, LFSR Counters, and Long Pseudorandom Sequence Generators”, Application Note (XAPP 052), http://www.xilinx.com/support/documentation/application_notes/xapp052.pdf, 1996

[YAF+03] Young S., Alfke P., Fewer C., McMillan S., Blodger B., Levi D.: “A High I/O Reconfigurable Crossbar Switch”, Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, S. 3-10, 2003

[YH99] Yao X., Higuchi T.: “Promises and Challenges of Evolvable Hardware”, IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews, vol. 29, no. 1, S. 87-97, 1999

[ZHL04] Zheng, K., Hu, C., Lu, H., Liu, B.: “An Ultra High Throughput and Power Efficient TCAM-based IP Lookup Engine”, Proceedings of the IEEE INFOCOM, S. 1984-1994, 2004

[Zim80] Zimmermann, H.: “OSI Reference Model – The ISO Model of Architecture for Open System Interconnection”, IEEE Transactions on Communications, vol 28, no. 4, S. 452-432, 1980

[ZN03] Zibin D. Ning Z.: “FPGA Implementation of SHA-1 Algorithm”, Proceedings of the 5th Interational Conference on ACIC, 2003, S. 1321-1324

[ZNB03] Zane, F., Narlikar, G., Basu, A.: “CoolCAMs: Power-Efficient TCAMs for Forwarding Engines”, Proceedings of the IEEE INFOCOM, S. 42-52, 2003

Anhang

Die Abbildungen und Tabellen in diesem Anhang stellen Einzelheiten der Arbeit und Untersuchungsergebnisse vor, auf die in den vorangegangenen Abschnitten nicht näher eingegangen werden konnte. Das betrifft vor allem die Darstellung der Simulationsergebnisse für die einzelnen Architekturen der evolvierbaren Hashfunktion.

Anhang A ASIC vs. FPGA Größenverhältnisse

Die hier gemachten Angaben für die Hardwarekosten desselben Designs beziehen sich auf die in [SCK+06] vorgestellte Routerarchitektur. Diese wurde unter Verwendung unterschiedlicher Parameter sowohl auf einem ASIC (ST65[21]) als auch auf einem Virtex-4 FPGA (XC4VLX100) synthetisiert. Für die Verhältnisse ergeben sich die folgenden Werte:

		ASIC		FPGA			
	FIFO Tiefe	# Gate	Total Cell Area	# Slices	# Flipflops	# LUTs	Gate/Slice
Router	1	1091	5038.80	332	201	591	3.286
	2	3331	13465.92	633	381	1139	5.262
	4	3818	18128.24	1573	729	2974	2.427
	8	6430	31260.32	1627	1434	2958	3.952
	16	11850	61176.44	4320	2829	7806	2.743
	32	25986	119437.24	8146	5599	14667	3.190
Netz-werk	1	14015	70321.68	3493	2611	6435	4.012
	2	35146	160951.44	7436	4884	13878	4.726
	4	54098	276226.06	12660	9425	24102	4.273
	8	105531	528807.25	18604	18328	34701	5.672

Anhang B TM Farbmarkierungen

MPLS Label Stack (Methode 1)

Ist der Traffic Manager entsprechend konfiguriert, erfolgt die Farbmarkierung der Daten im Experimental-Feld (EXP-Feld) eines jeden MPLS Labels im MPLS Label Stack des Frames. Das EXP-Feld umfasst drei Bit und ist laut Standard noch nicht in Verwendung, sondern kann für experimentelle Anwendungen verwendet werden. Die Farbmarkierung erfolgt entsprechend der folgenden Tabelle. Hierbei handelt es sich um eine Kodierung von 8 Verkehrsklassen auf 5 Traffic-Klassen von denen drei mit einer Farbmarkierung versehen werden können.

[21] ST65: 65 nm-Prozess der Firma STMicroelectronics

Tabelle 15 - Farbcodierung des EXP Feldes des MPLS Labels

Altes Exp-Feld	Farbe	Verkehrsklasse, DP	Neues EXP Feld
111	-	(Expedited Forwarding) EF	111
110, 101	Grün	Assured Forwarding AF11	110
110, 101	Gelb	Assured Forwarding AF12	101
100, 011	Grün	Assured Forwarding AF21	100
100, 011	Gelb	Assured Forwarding AF22	011
010, 001	Grün	Assured Forwarding AF31	010
010, 001	Gelb	Assured Forwarding AF32	001
000	-	Best Effort BE	000

VLAN Tag (Methode 2)

Die Farbmarkierung der Datenframes kann auch im 802.1p-Feld eines VLAN Tags vorgenommen werden. In diesem ist die Verkehrsklasse der Frames kodiert, welche zum Zweck der Farbmarkierung umkodiert wird. Die Markierung erfolgt analog zur Farbkodierung des EXP-Feldes des MPLS Labels also entsprechend der Angaben in Tabelle 15.

DSCP Feld des IP-Headers (Methode 3)

Die dritte Möglichkeit der Farbkodierung ist das Heranziehen des DSCP-Feldes des IP-Headers eines Frames. Um eine maximale Variabilität zu gewährleisten, findet ein BRAM Verwendung, bei dem einem aktuellen DSCP-Wert ein beliebiger neuer DSCP-Wert für den Fall einer grünen und den Fall einer gelben Markierung zuzuweisen ist. Frames, welche kein IP-Protokoll enthalten, werden, ohne markiert zu werden, weitergeleitet. Jeder Speichereintrag ist zur Laufzeit konfigurierbar.

Provider Bridge (Methode 4)

Es besteht weiterhin die Möglichkeit, die Farbmarkierung nach dem IEEE Standard 802.1ad vorzunehmen. Es ist hierbei vorgesehen, von verschiedenen Anzahlen von Verkehrsklassen und dementsprechend von verschiedenen Möglichkeiten der Farbmarkierung auszugehen. Die Markierung der Frames erfolgt im VLAN eines jeden Frames. Es bestehen vier Möglichkeiten der Markierung (Tabelle 16 - Tabelle 19):

Tabelle 16 - 8P0D: Es existieren acht Prioritäten und keine Farbmarkierungen.

Alte Priorität	Neue Priorität	Farbe	Codierung
7 (111)	7	-	111
6 (110)	6	-	110
5 (101)	5	-	101
4 (100)	4	-	100
3 (011)	3	-	011
2 (010)	2	-	010
1 (001)	1	-	001
0 (000)	0	-	000

Tabelle 17 - 7P1D: Es exitsitern sieben Prioritäten und eine Möglichkeit der Farbmarkierung.

Alte Priorität	Neue Priorität	Farbe	Codierung
7 (111)	7	-	111
6 (110)	6	-	110
5 (101)	5	grün	101
4 (100)	5	gelb	100
3 (011)	3	-	011
2 (010)	2	-	010
1 (001)	1	-	001
0 (000)	0	-	000

Tabelle 18 - 6P2D: Es existieren sechs Prioritäten und zwei Möglichkeiten der Farbmarkierung.

Alte Priorität	Neue Priorität	Farbe	Codierung
7 (111)	7	-	111
6 (110)	6	-	110
5 (101)	5	grün	101
4 (100)	5	gelb	100
3 (011)	3	grün	011
2 (010)	3	gelb	010
1 (001)	1	-	001
0 (000)	0	-	000

Tabelle 19 - 5P3D: Es existieren fünf Prioritäten und drei Möglichkeiten der Farbmarkierung.

Alte Priorität	Neue Priorität	Farbe	Codierung
7 (111)	7	-	111
6 (110)	6	-	110
5 (101)	5	grün	101
4 (100)	5	gelb	100
3 (011)	3	grün	011
2 (010)	3	gelb	010
1 (001)	1	grün	001
0 (000)	1	gelb	000

Handelt es sich bei den empfangenen Frames um ein Service-VLAN (SVLAN definiert in [IEE06b]), ist eine weitere Kodierung möglich. Das DEI[22]-Feld des SVLAN-Tags kann genutzt werden, um die Farbmarkierung vorzunehmen. Daraus folgt, dass acht Prioritäten und acht Möglichkeiten der Farbmarkierung gegeben sind:

Tabelle 20 - Farbmarkierung bei Nutzung des DEI-Felds einer SVLAN.

Alte Priorität	Neue Priorität	Farbe	Codierung	DEI
7 (111)	7	grün	111	0
6 (110)	6	gelb	110	1
5 (101)	5	grün	101	0
4 (100)	4	gelb	100	1
3 (011)	3	grün	011	0
2 (010)	2	gelb	010	1
1 (001)	1	grün	001	0
0 (000)	0	gelb	000	1

Anhang C IPclip-Option

Als Typinformation in der zusätzlichen IP-Option wurde für eine prototypische Implementierung die Zahl 26 gewählt, um IPclip-Optionen zu identifizieren. Diese wurde von der International Asigned Numbers Authority (IANA) [IAN07] noch nicht vergeben. Die IPclip-Option ist in das Value-Feld der IP-Option einzutragen. Die IPclip-Option besteht aus einem Typ, der ein Byte lang ist. Dazu kommen vier Bit für Flags und die OI, die eine variable

[22] DEI: Drop Eligable Indicator

Länge haben kann. Das Auffüllen (Padding) mit Bits kann notwendig sein, da die Summe aller IP-Optionen in einem IP-Paket ein Vielfaches von 32 Bit sein muss.

Insgesamt vier Bit sind als Status Flags vorgesehen. Von diesen geben zwei Bit über die Herkunft und die Vertrauenswürdigkeit einer IPclip-Option Auskunft. Die entsprechenden Informationen sind Tabelle 21 zu entnehmen. Die restlichen zwei Bit sind noch nicht zugewiesen und zukünftigen Entwicklungen vorbehalten.

Tabelle 21 - Flags der IPclip-Option. Darstellung des allgemeinen Falls (vgl. mit Tabelle 7).

Flag	Quelle/ Vertrauenswürdigkeit	Beschreibung
00	`user provided / untrusted`	Die IPclip-Option wurde von einem Endgerät beim Nutzer eingetragen und durch IPclip **nicht** als vertrauenswürdig validiert.
01	`user provided/ trusted`	Die IPclip-Option wurde von einem Endgerät beim Nutzer eingetragen und durch IPclip als vertrauenswürdig validiert.
10	`network provided/ untrusted`	Die IPclip-Option wurde von einem Endgerät beim Nutzer eingetragen, durch IPclip **nicht** als vertrauenswürdig validiert und deshalb durch eine korrekte IPclip-Option ersetzt.
11	`network provided/ trusted`	Im Teilnehmerzugangsknoten wurde eine neue IPclip-Option in das IP-Paket eingefügt

Das Format, das genutzt wird, um GPS-Informationen zu transportieren, ist das NMEA-0182 Datenformat [NME02]. Die relevanten Informationen umfassen die Längen- und Breiteninformationen (siehe Tabelle 22).

Tabelle 22 - Format der GPS Ortsinformationen.

Information			Weite	# der Bits
Optionsinformation				57
	Breite			28
		Grad	0..90	7
		Minuten (Ganzzahl – Int.)	0…59	6
		Minuten (Nachkommastelle – Frag.)	$0…(1-2^{-15})$	14
		Hemisphäre	N,S	1
	Länge			29
		Grad	0..180	8
		Minuten (Ganzzahl – Int.)	0…59	6

		Minuten (Nachkommastelle – Frag.)	$0…(1-2^{-15})$	14
		Hemisphäre	O,W	1

Die Kodierung der Informationen in eine IP-Option ist in Abbildung 108 dargestellt. Insgesamt sind 57 Bit für die Kodierung der OI notwendig. Mit 12 Bit für IPclip-Type und Flags ergeben sich 9 Byte für eine IPclip-Option. Die gesamte IP-Option benötigt 11 Byte. Werden auch Knoten ID und Portnummer übertragen sind insgesamt 15 Byte für die IP-Option notwendig.

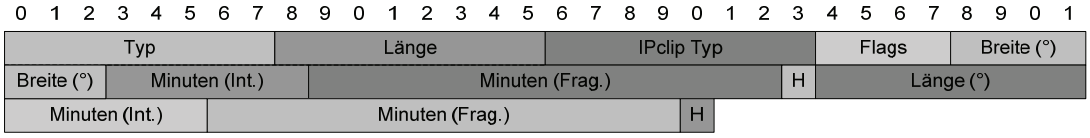


Abbildung 108 - IP-Option, die eine IPclip-Option mit GPS Daten enthält. Insgesamt sind 11 Byte notwendig.

Anhang D Abbildungen zu Kapitel 5

Dargestellt werden die Verläufe der statistischen Erhebung der Fitnessentwicklungen verschiedener evolvierbarer Hashfunktionen. Es wurden jeweils 20 Untersuchungen mit unterschiedlich großen und unterschiedlich zusammengesetzten Schlüsselmengen durchgeführt. Der Mittelwert ist jeweils hervorgehoben. Es ist zu beachten, dass der Wertebereich in den Graphen nicht identisch ist. Die folgenden Abbildungen stellen die Fitness für drei verschieden große Schlüsselmengen dar (131071 – 128k, 65536 – 64k, 32768 – 32k)

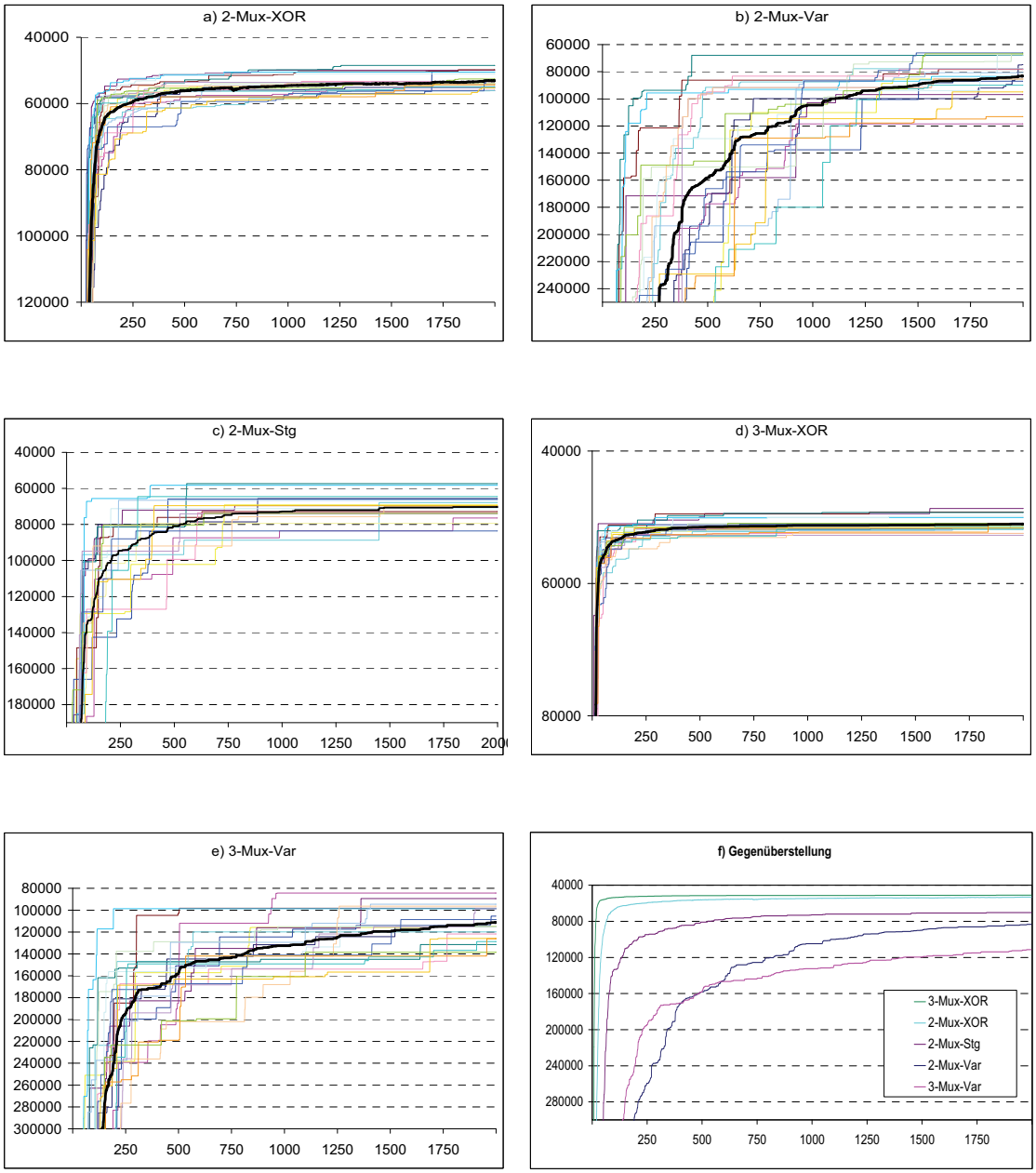


Abbildung 109 - Fitnessentwicklung der evolvierbaren Hashfunktion für 131072 Schlüssel. Auf der Y-Achse ist jeweils die Fitness über der Generation für 20 unabhängige Simulationsläufe aufgetragen. Die schwarze Linie stellt den Mittelwert dar. Die Untersuchungen wurden für einen genetischen Algorithmus und 2000 Generationen durchgeführt. f) verdeutlicht die Entwicklung und das Verhältnis der Fitnesswerte der einzelnen Architekturen im Verlauf von 2000 Generationen zueinander.

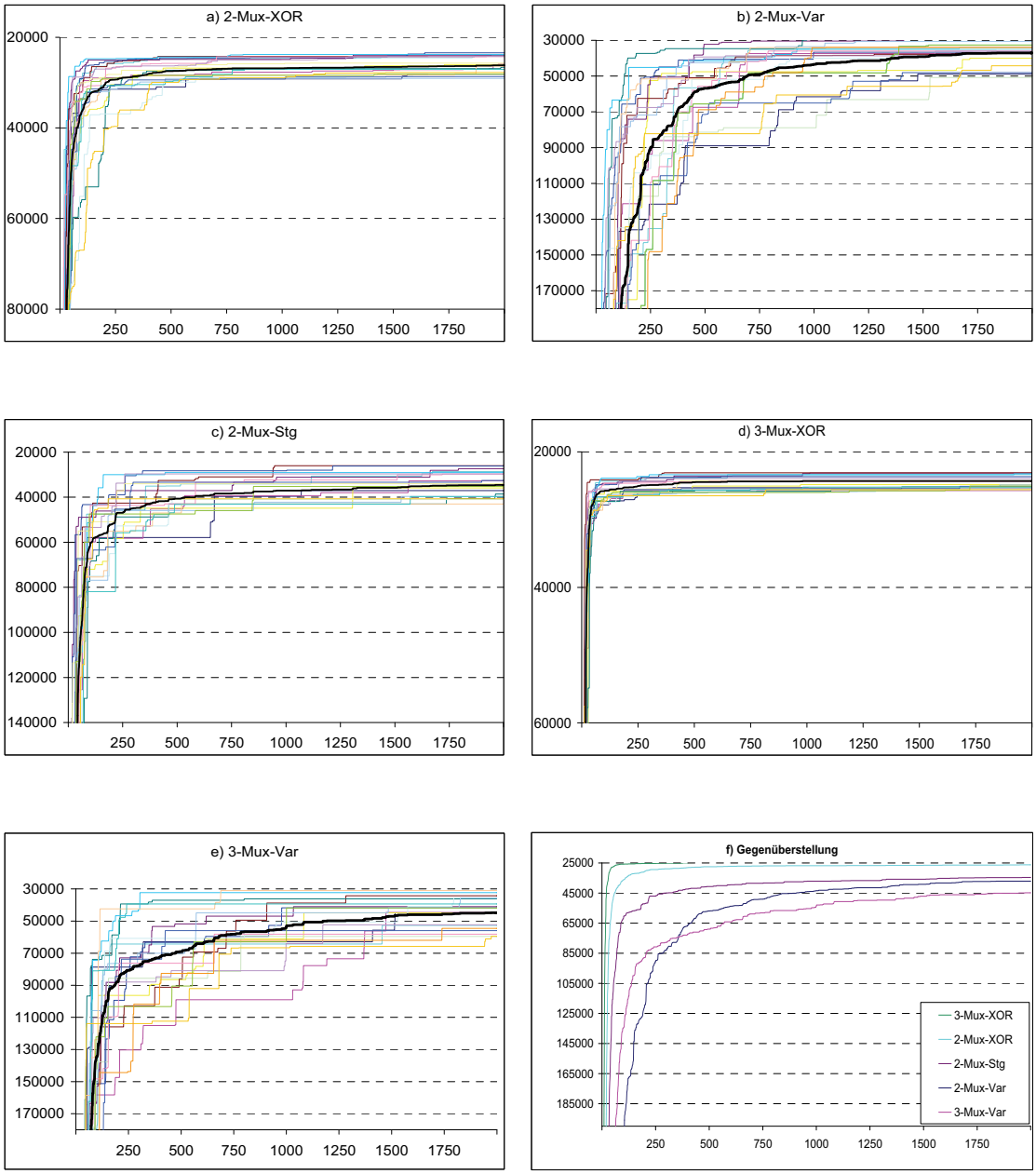


Abbildung 110 - Fitnessentwicklung der evolvierbaren Hashfunktion für 65536 Schlüssel. Auf der Y-Achse ist jeweils die Fitness über der Generation aufgetragen. Die Untersuchungen wurden für einen genetischen Algorithmus und 2000 Generationen durchgeführt. f) verdeutlicht die Entwicklung und das Verhältnis der Fitnesswerte der einzelnen Architekturen im Verlauf von 2000 Generationen zueinander.

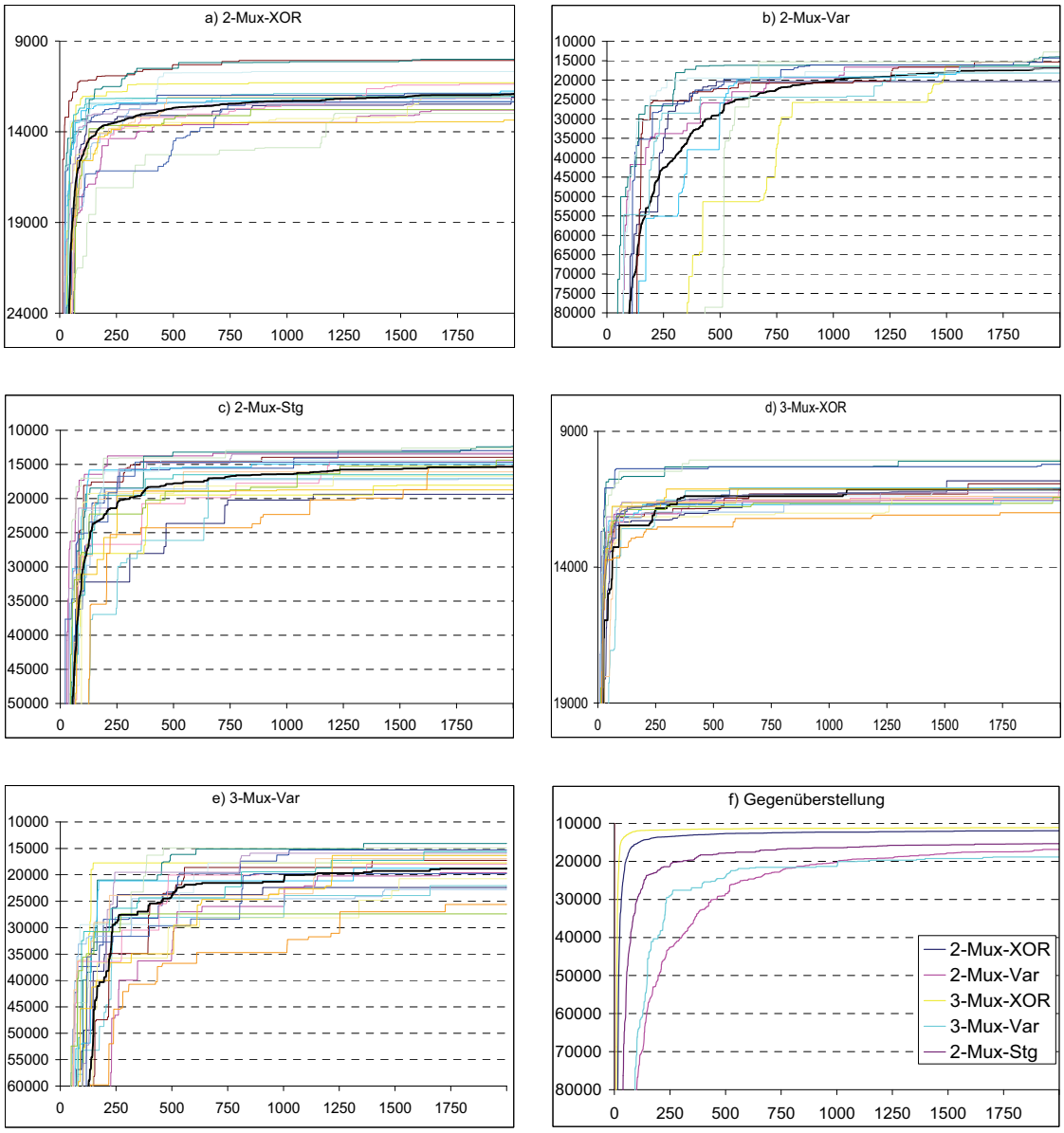
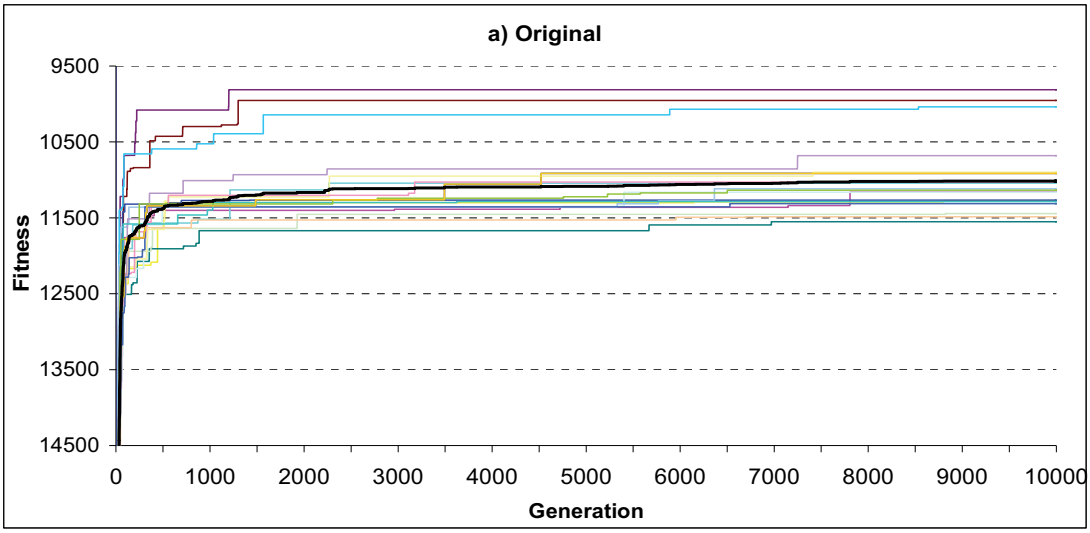


Abbildung 111 - Fitnessentwicklung der evolvierbaren Hashfunktion für 32768 Schlüssel. Auf der Y-Achse ist jeweils die Fitness über der Generation aufgetragen. Die Untersuchungen wurden für einen genetischen Algorithmus und 2000 Generationen durchgeführt. f) verdeutlicht die Entwicklung und das Verhältnis der Fitnesswerte der einzelnen Architekturen im Verlauf von 2000 Generationen zueinander



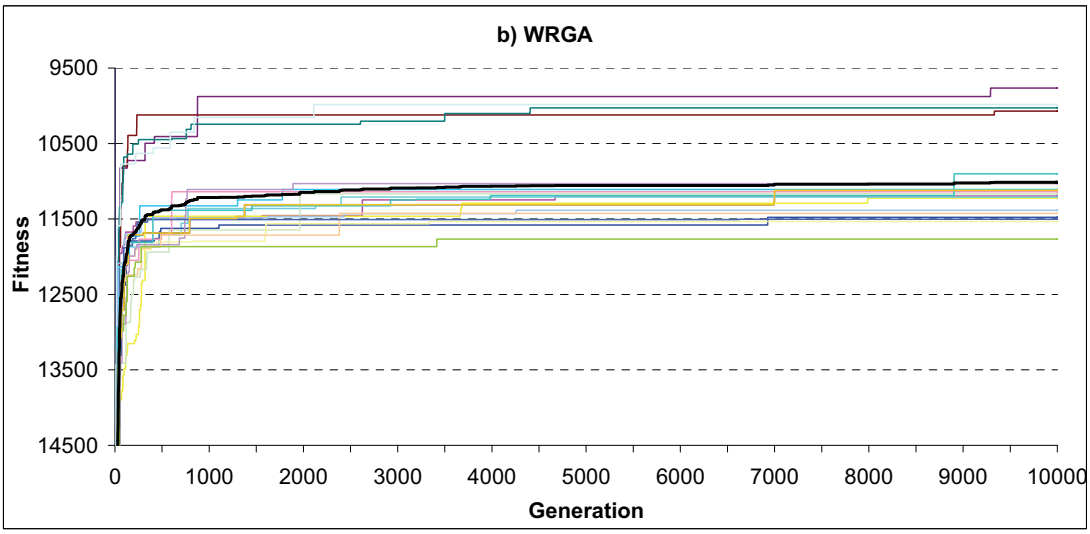


Abbildung 112 - Entwicklung der Fitness des EPC ohne und mit wiederholt reinitialisiertem GA. Es wurden 10000 Generationen evaluiert, wobei im Falle von WRGA alle 500 Generationen der GA reinitialisiert wurden. Es ist zu erkennen, dass sich WRGA mit dieser Konfiguration nicht besser verhält und auch der Originalalgorithmus nach mehreren Tausend Generationenen nicht vollständig terminiert.

Anhang E Abbildungen zu Kapitel 6

Dargestellt sind die Entwicklungen der Fitnessverläufe der evolvierbaren Hashfunktion mit unterschiedlich großen Populationen über die Zeit in Sekunden. Wie zu erkennen ist, werden befriedigende Erbnisse nach unterschiedlich langen Zeiträumer erreicht.

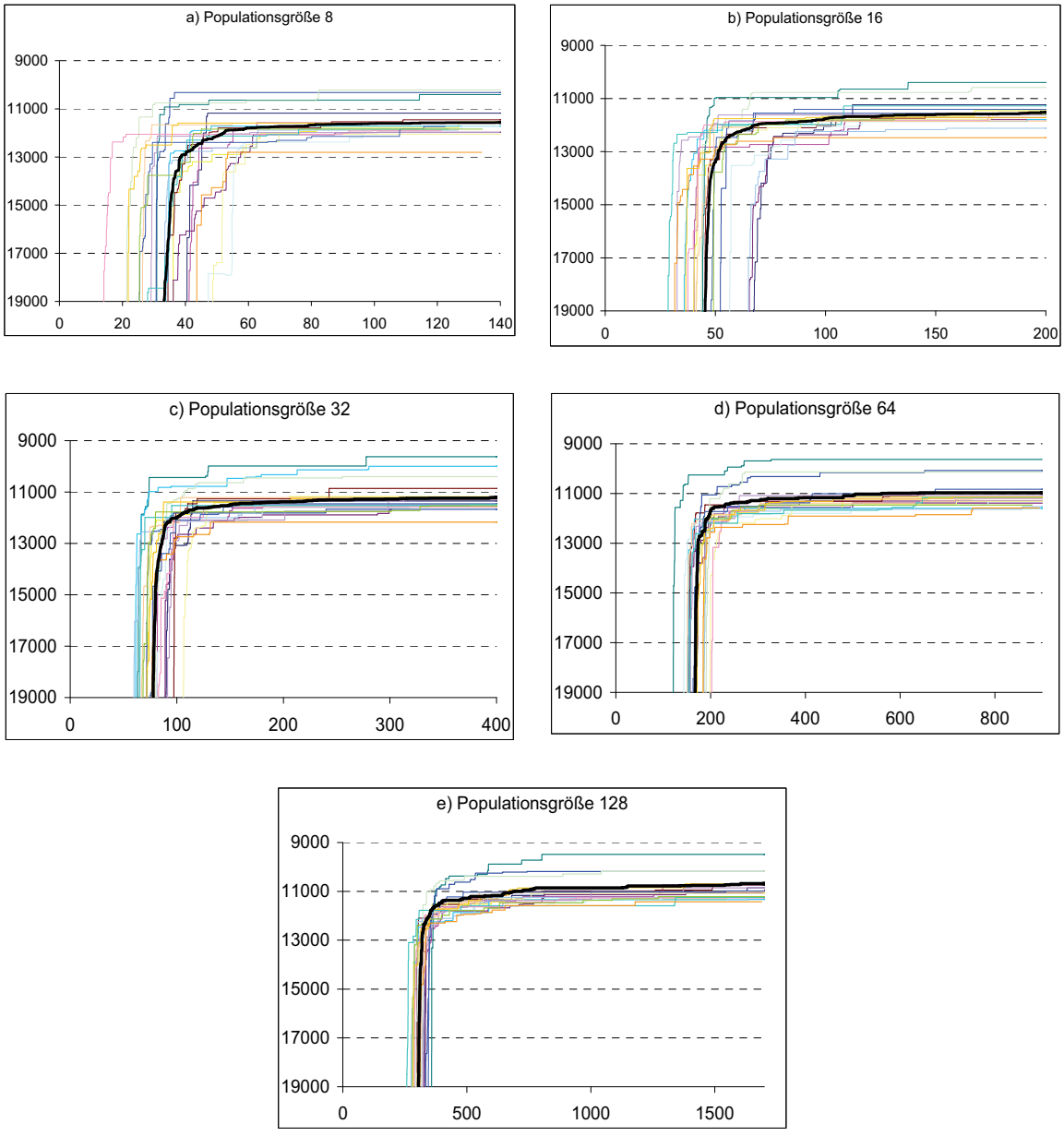


Abbildung 113 - Verhalten des GA bei der Untersuchung unterschiedlicher Populationsgrößen. Es wurde für jeweils 20 unabhängige Simulationsläufe die Fitness über der Zeit in Sekunden aufgetragen.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder veröffentlicht noch einer anderen Prüfungsbehörde vorgelegt.

Rostock, den 23.11.2008 Harald Widiger

Lebenslauf

Persönliche Daten

Harald Widiger
Budapester Str. 82
18057 Rostock
0176-24412649
harald.widiger@uni-rostock.de
Geb. am 23.08.1978 in Halle(Saale)
Ledig
Deutsch

Schulausbildung

09/1985 - 10/1988	Hermann-Matern-Oberschule Halle-Neustadt
11/1988 - 07/1991	19. Oberschule Rostock-Stadt
08/1991 - 07/1994	Gymnasium Toitenwinkel Rostock
08/1994 - 06/1997	Käthe-Kollwitz-Gymnasium Rostock

Wehrdienst

07/1997 - 04/1998	5./St/FmBtl 1 in Rotenburg(Wümme)

Studium

10/1998 - 03/2004	Studium der Informationstechnik/Technischen Informatik an der Universität Rostock, Schwerpunkt Informationstechnik Diplomingenieur Informationstechnik
10/2001 - 02/2002	Studienbegleitendes Praktikum (Toshiba in Düsseldorf)

Beruflicher Werdegang

seit 03/2004	Wissenschaftlicher Mitarbeiter am Institut für Angewandte Mikroelektronik und Datentechnik der Universität Rostock

Thesen

1. Die zunehmende Bandbreite und steigende funktionale Diversifikation im Teilnehmerzugangsbereich erfordert eine Paketverarbeitung in Hardware, damit eine Verarbeitung in „wirespeed“ und mit einer geringen Latenz möglich bleibt.
2. Die Paketverarbeitung umfasst neben der eigentlichen Verarbeitung (Manipulation, Weiterleitung, Verwurf, Sortierung) auch die Klassifizierung der Pakete. Beide Aufgaben müssen durch eine Lösung erfüllt werden, die bestehende Leistungsanforderungen erfüllt.
3. Die vorgestellte Architektur funktionaler Module (FM-Architektur) ist eine solche leistungsfähige Lösung, um die unterschiedlichen Aufgaben, die die Paketverarbeitung betreffen, zu realisieren. Mittels der FM-Architektur kann ein Durchsatz von über 4 GByte/s erreicht werden.
4. Die funktionalen Module zur MAC Address Translation (MAT), dem Traffic Management (TM) und dem Multiprotocol Label Switching - User Network Interface (MPLS-UNI) realisieren ihre Funktionalitäten durch die Anwendung der leistungsfähigen FM-Architektur.
5. Mittels MAT ist es möglich, sowohl die Skalierbarkeit als auch die Sicherheit von Teilnehmerzugangsnetzwerken zu erhöhen. Eine 1:1 Übersetzung von MAC-Adressen erhöht die Sicherheit im Netz, während durch eine n:1 Übersetzung die Skalierbarkeit erhöht und die „MAC Address Table Explosion“ verhindert wird.
6. Der Traffic Manager ist in der Lage, Datenframes verschiedener Nutzer farblich zu markieren (grün, gelb, rot). Damit kann im Falle der Notwendigkeit eine gerechte Verwurfsstrategie genutzt werden, wodurch die QoE einzelner Nutzer weniger stark leidet.
7. Das MPLS-UNI ermöglicht die Anreicherung von Datenframes mit zusätzlichen Informationen aus dem Teilnehmerzugangssystem. Diese Informationen können dann im Kernnetz Verwendung finden, um die verschiedensten Funktionalitäten zu realisieren.
8. Als „Trust-by-Wire“ bezeichnet man das Wissen um die Identität/Position eines Kommunikationspartners aus einem physikalischen Medium heraus, woraus sich eine Vertrauensbeziehung ableiten lässt (z. B. Telefonnummer → Telefonleitung → Gesprächspartner). Trust-by-Wire ist in heutigen IP-basierten Netzwerken nicht vorhanden. Es ist jedoch für viele unterschiedliche Anwendungen von essentieller Bedeutung.
9. IPclip (abgeleitet von der CLIP Fuktionalität in ISDN-Netzwerken) führt Trust-by-Wire in IP-basierte Netzwerke ein. Mit IPclip können Funktionen wie z. B. VoIP-Notrufe realisiert werden. Außerdem erweitert IPclip die Möglichkeiten zur Bekämpfung von E-Mail Spam und Phishing.

10. MATMUNI stellt eine ressourcensparende Möglichkeit dar, unterschiedliche Funktionalitäten effizient zu kombinieren. Der Implementierungsaufwand und steigende Hardwarekosten mit zunehmender Anzahl von FMs rechtfertigen allerdings maximal jeweils vier FMs in Up- und Downstream.
11. Die blockorientierte Verarbeitung benötigt mehr Hardwareressourcen als MATMUNI. Sie ist allerdings sehr einfach zu implementieren. Die blockorientierte Verarbeitung kann auch für mehr als vier FMs zum Einsatz kommen.
12. Hashfunktionen sind mit einer theoretisch möglichen Komplexität von O(1) ausgezeichnet geeignet, um zur Paketklassifizierung eingesetzt zu werden. Sie können sowohl für das heutige Internetprotokoll in der Version 4 als auch das zukünftige Internetprotokoll in der Version 6 verwendet werden. Nachteilig zu bewerten ist allerdings, dass „Prefix Match“ mit Hashfunktionen nicht ohne weiteres möglich ist.
13. Mittels eines genetischen Algorithmus (GA) ist es möglich, die Architektur von Hashfunktionen zu optimieren, um eine Anpassung der Hashfunktion an eine gegebene Schlüsselmenge vorzunehmen. Eine solche evolvierbare Hashfunktion kann sich auch an veränderliche Schlüsselmengen anpassen.
14. Die Abdeckung eines größeren Parameterraums einer evolvierbaren Hashfunktion (Länge des Genoms) muss nicht zu einer besseren Hashfunktion führen. Die eigentliche Architektur der Hashfunktion ist der entscheidende Faktor.
15. Die beste entwickelte evolvierbare Hashfunktionsarchitektur ist die 3-Mux-XOR Architektur. Sie zeigt eine bessere Leistung als eine Hashfunktionsarchitektur auf Basis der CRC32-Checksumme, welche die beste dem Autor bekannte hardware-basierte Hashfunktion ist.
16. Der vorzeitige Abbruch, die parallele Fitnessevaluierung, die Speicherverschränkung sowie eine Kombination der Maßnahmen sind sehr gut geeignet, um den Vorgang der Fitnessevaluierung des entwickelten GA zu beschleunigen.
17. Die Wahl der Parameter eines GA hat großen Einfluss auf die Leistungsfähigkeit einer evolvierbaren Hashfunktion und ist deshalb sorgfältig zu treffen. Die Suche nach sehr guten Parametern könnte mittels eines GA „zweiter Ordnung“ optimiert werden.
18. Smart Initialization und die wiederholte Reinitialisierung des verwendeten GA sind zur Optimierung der Evolution der evolvierbaren Hashfunktion nicht geeignet.
19. Das Caching von Schlüsseln im Datenpfad, die viele Hashkollisionen erzeugen, ist im Gegensatz zu herkömmlichen Cachestrategien sehr sinnvoll. Auch mit sehr kleinen Cachegrößen kann eine substantielle Verbesserung der Leistungsfähigkeit eines Paketklassifizierers erzielt werden. Des Weiteren kann die maximale Anzahl von Kollisionen, die aufgelöst werden muss, reduziert werden, was das „worst-case“-Verhalten eines evolvierbaren Paketklassifizierers verbessert.

Kurzreferat

Moderne Teilnehmerzugangsnetzwerke stehen vor zwei Problemen: Zum einen nehmen die den Nutzern zur Verfügung stehenden Bandbreiten zu. Zum anderen wird eine steigende Zahl von Diensten bereits in den Systemen des Teilnehmerzugangs erbracht. Deshalb sind paketverarbeitende Systeme mit großen Durchsatzraten und gleichzeitig großer funktionaler Flexibilität und Erweiterbarkeit notwendig. Ziel dieser Arbeit ist es, die Grundlage zur Entwicklung derartiger Systeme zu schaffen.

Um leistungsfähige und gleichzeitig flexible paketverarbeitende Systeme zu ermöglichen, wurden Entwicklungen in zwei wichtigen Bereichen vorgenommen: der eigentlichen Paketverarbeitung (Manipulation, Weiterleitung, Verwurf, Sortierung von Paketen) und der Paketklassifizierung.

Zur Optimierung der Paketverarbeitung wurde eine FPGA-basierte Architektur für funktionale Module entwickelt, die die schritthaltende Verarbeitung von Paketen mit Datenraten von 4 Gbit/s und mehr ermöglichen. Derartig aufgebaute funktionale Module können leicht miteinander kombiniert werden, ohne Leistungseinbußen hinnehmen zu müssen. Für den Einsatz im Teilnehmerzugangsnetzwerk wurden verschiedene funktionale Module entwickelt. Dazu gehören Module zur MAC Address Translation und dem Traffic Management sowie ein Multiprotocol Label Switching User Network Interface.

Jedes Paket muss vor einer Verarbeitung klassifiziert werden muss Eine schnelle Paketklassifizierung ist deshalb die notwendige Vorraussetzung einer leistungsfähigen Paketverarbeitung. Hash-basierte Klassifizierungsalgorithmen und –architekturen sind dafür grundsätzlich geeignet. Zur Optimierung dieser Klasse von Algorithmen wurde im Rahmen dieser Arbeit eine evolvierbare Hashfunktionsarchitektur entwickelt, die sich auf Basis eines in Hardware implementierten genetischen Algorithmus an die jeweils existierenden Klassifizierungsanforderungen (die Zusammensetzung der Schlüsselmenge) anzupassen vermag. Die evolvierbare Hashfunktion wurde als Teil eines einfachen evolvierbaren Paketklassifizierers implementiert und in verschiedenen Bereichen optimiert.

Somit konnten Beiträge zu zwei relevanten Aspekten der Paketverarbeitung geleistet werden.

Abstract

Modern Access Networks face two serious problems: On the one hand the bandwidth for each connected customer is rising permanently. On the other hand an increasing number of services is already provided in the Access Network. Thus, packet processing systems are required, which have both high performance and functional flexibility.

In order to allow for powerful and flexible packet processing systems, developments in two areas have been conducted: ultimate packet processing (manipulation, forwarding, discarding, sorting of packet), and packet classification.

For optimization of packet processing an FPGA-based architecture for functional modules has been developed, which provides computation of packets in wire speed for data rates of 4 Gbit/s and more. Functional modules of the developed architecture can be combined easily without suffering from performance loss. For application in Access Network, different functional modules have been developed, including a module for MAC Address Translation and a module for Traffic Management as well as a Multiprotocol Label Switching User Network Interface.

Fast packet classification is a necessary precondition for high speed packet processing, as each packet has to be classified prior to computation. In principle, hash-based classification algorithms and architectures are appropriate for fast packet classification. For optimization of this class of algorithms, in this work, an evolvable architecture for hash functions has been developed. Based on a genetic algorithm, the hash function is capable of adapting to different classification demands (different configurations of the key set). The evolvable hash function has been implemented as part of a simple evolvable packet classifier. It has been optimized in different areas.

With these developments, contributions to two relevant aspects of packet processing have been made.

Liste der Veröffentlichungen auf Fachvorträgen und Tagungen

- Widiger H., Kubisch S., Duchow D., Hecht R., Ossipowa N., Timmermann D., Tavangarian D.: „MWN – Ethernet-basierte & mobile drahtlose Zugangs-architekturen
- Widiger H., Handy M., Timmermann D.: „Packet Classification with Evolvable Hardware Hash Functions" Proceedings of the 8th EUROMICRO Conference on Digital System Design (DSD 2005), Porto, Portugal, September 2005
- Widiger H., Salomon R., Timmermann D.: "Packet Classification with Evolvable Hardware Hash Functions" Proceedings of the 2nd International Workshop on Biologically Inspired Approaches to Advanced Information Technology (BioADIT), Osaka, Japan, Februar 2006, S. 64-79
- Widiger H., Kubisch S., Duchow D., Bahls T., Timmermann D.: „A Simplified, Cost-Effective MPLS Labeling Architecture for Access Networks" Proceedings of the World Telecommunications Congress (WTC 2006), Budapest, Ungarn, Mai 2006
- Kubisch S., Widiger H., Duchow D., Bahls T., Timmermann D.: „Wirespeed MAC Address Translation and Traffic Management in Access Networks" Proceedings of the World Telecommunications Congress (WTC 2006), Budapest, Ungarn, Mai 2006
- Duchow D., Bahls, T., Timmermann D., Kubisch S., Widiger H.: „Efficient Port-based Network Access Control for IP DSLAMs in Ethernet-based Fixed Access Networks" Proceedings of the World Telecommunications Congress (WTC 2006), Budapest, Ungarn, Mai 2006
- Salomon R., Widiger H., Tockhorn A.: „Rapid Evolution of Time-efficient Packet Classifiers" Proceedings of the IEEE World Congress on Computational Intelligence (WCCI 2006), Vancouver, Kanada, Juli 2006, S. 2793-2799
- Duchow D., Bahls T., Timmermann D., Widiger H., Kubisch S.: "ACIP: An Access Control and Information Protocol for Ethernet-based Broadband Access Networks" Proceedings of the 12th International Telecommunications Network Strategy and Planning Symposium (Networks 2006), New Delhi, Indien, November 2006
- Widiger H., Kubisch S., Timmermann D., Bahls T.: „An Integrated Hardware Solution for Mac Address Translation, MPLS, and Traffic Management in Access Networks" Proceedings of the 31st Annual IEEE Conference on Local Computer Networks (LCN), Tampa, FL., USA, November 2006, S. 272-279
- Kubisch S., Widiger H., Hecht R., Timmermann D.: „Network-on-Chip Communication Grids for High Performance Packet Processing" Proceedings of the 15th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2007), Monterey, USA, Februar 2007, S. 228

- Kubisch S., Widiger H., Hecht R., Timmermann D., Siemroth M.: „Architektur einer Flexiblen, Wiederverwendbaren Testbench zur Verifikation Paketverarbeitender Hardware in SystemC“ 10. GI/ITG/GMM Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen", Erlangen, Deutschland, März 2007, S. 9-18
- Kubisch S., Widiger H., Danielis P. Duchow D., Bahls T. Timmerann D., Lange C., Röwer O.: „Configuration Tool and FPGA-Prototype of a Hardware Packet Processing System” Design, Automation and Test in Europe Conference and Exhibition (DATE 2007) – University Booth, Nizza, Frankreich, April 2007
- Kubisch S., Widiger H., Cornelius C., Timmermann D., Strzeletz A.: „E-Core - A Configurable IP Core for Application-specific NoC Performance Evaluation” Design, Automation and Test in Europe Conference and Exhibition (DATE 2007) – Workshop on Diagnostic Services in Network-on-Chips, Nizza, Frankreich, April 2007, S. 149-151
- Widiger H., Tockhorn A., Salomon R., Timmermann D.: „Acceleration of the Evolution of Evolvable Hardware-based Packet Classifiers” Proceedings of the IEEE SSCI 2007 - Workshop on Evolvable and Adaptive Hardware (WEAH 2007), Honolulu, USA, April 2007, S. 27-34
- Kubisch S., Widiger H., Timmermann D., Duchow D., Bahls T.: „sMAT - A Simplified MAC Address Translation Scheme” Proceedings of the 15th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN 2007), Princeton, USA, Juni 2007
- Widiger H., Kubisch S., Timmermann D.: „ A Structural Architecture for HW Packet Processing” Proceedings of the 2007 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, Victoria, Kanada, August 2007, S. 363-366
- Danielis P., Kubisch S., Widiger H., Schulz J., Timmermann D., Bahls T., Duchow D.: „IPclip - An Innovative Mechanism to Reestablish Trust-by-Wire in Packet-switched IP Networks” 3. Essener Workshop "Neue Herausforderungen in der Netzsicherheit", Essen, Deutschland, März 2008
- Kubisch S., Widiger H., Danielis P., Schulz J., Timmermann D., Bahls T., Duchow D.: „Complementing E-Mails with Distinct, Geographic Location Information in Packet-switched IP Networks“ MIT 2008 Spam Conference, Cambridge, USA, März 2008
- Danielis P., Kubisch S., Widiger H., Schulz J., Duchow D., Bahls T., Timmermann D., Lange C.: „Trust-by-Wire in Packet-switched IP Networks: Calling Line Identification Presentation for IP” Design, Automation and Test in Europe

Conference and Exhibition (DATE 2008) – University Booth, München, Deutschland, März 2008

- Danielis P., Kubisch S., Widiger H., Schulz J., Duchow D., Bahls T., Timmermann D.: „A Conceptual Framework for Increasing Physical Proximity in Unstructured Peer-To-Peer Networks" Proceedings of the IEEE Sarnoff Symposium 2008, Princeton, USA, April 2008
- Kubisch S., Widiger H., Danielis P., Schulz J., Timmermann D., Bahls T., Duchow D.: „Countering Phishing Threats with Trust-by-Wire in Packet-switched IP Networks - A Conceptual Framework" Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS) – 4th International Workshop on Security in Systems and Networks (SSN 2008), Miami, USA, April 2008
- Kubisch S., Widiger H., Danielis P., Schulz J., Timmermann D., Bahls T., Duchow D.: „Trust-by-Wire in Packet-switched IP Networks: Calling Line Identification Presentation for IP" Proceedings of the 1st ITU-T Kaleidoscope Conference: Innovations in Next Generation Networks - Future Network and Services, Genf, Schweiz, Mai 2008, S. 375-382
- Widiger H., Strzeletz A., Timmermann D.: „Evaluation of Dynamic Bandwidth Allocation Algorithms for G-PON Systems using a reconfigurable Hardware Testbed" The 16th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN 2008), Cluj-Napoca, Rumänien, September 2007
- Widiger H., Kubisch S., Danielis P., Schulz J., Timmermann D., Bahls T., Duchow D.: „IPclip: An Architecture to restore Trust-by-Wire in Packet-switched Networks" The 33rd Annual IEEE Conference on Local Computer Networks (LCN), Montreal, Kanada, Oktober 2008, S. 312-319
- Widiger H., Tockhorn A., Timmermann D.: "On the Impact of Caching for high Performance Packet Classifiers" IEEE Global Communications Conference (GlobeCOM 2008), New Orleans, USA, November 2008
- Danielis P., Kubisch S., Widiger H., Rohrbeck J., Altman V., Skodzik J., Timmermann D., Bahls T., Duchow D.: „Trust-by-Wire in Packet-Switched IPv6 Networks: Tools and FPGA Prototype fort he IPclip System" IEEE Consumer Communications and Networking Conference, Las Vegas, USA, Januar 2009