Resource Management in Hardware Systems for Programmable Network Nodes

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde
eines Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

vorgelegt von

Carlos Macián

geb. in València

Hauptberichter:	Prof. Dr.-Ing. Dr. h. c. mult. Paul J. Kühn
Mitberichter:	Prof. Dr.-Ing. Georg Carle, Universität Tübingen
Tag der Einreichung:	19. 08. 2004
Tag der mündlichen Prüfung:	05. 02. 2007

Institut für Kommunikationsnetze und Rechnersysteme
der Universität Stuttgart

2007

„There is no such thing as the truth.
The truth is always for someone
and for something. „
Robert A. Cox (slightly reworded)

„Life, all life, is about asking questions,
not about knowing answers. „
Allie Keys, in the last episode of Taken

A mi iaio,
que hablaba poco
y decía mucho,
y que se hubiese sentido
orgulloso.

Since the advent of the World Wide Web, information exchange by means of computer networks has experienced a true worldwide revolution. The Internet has become ubiquitous and indispensable. Its original simplicity of design has been used to build around it a very rich, ever expanding set of services, with the help of a tremendously complex and diverse range of technologies. It is precisely this unheard-of success, which is severely putting to the test the ability of IP-based networks to scale with respect to the number and diversity of services that can be offered on top of them, as well as the aggregate volume of traffic that can be processed by and transported through the network and to the final user.

First of all, the *Multiservice Internet* is strained by the sheer diversity of services that should be supported. Although all of them can separately make use of the Internet to run acceptably, it is their interaction that represents a problem. Secondly, the complexity of network management is aggravated by the sheer volume of service instances that are active simultaneously worldwide. As a consequence, every network node has an ever decreasing amount of time to process an always increasing amount and diversity of information. This dual problem is aggravated by the increasing concern by the users regarding security and Quality of Service (QoS). Last but not least, innovation is, from the point of view of the network operator, a blessing but also a jinx. Innovation implies the introduction of new functionality in an existing infrastructure, for which it was not built. The consequences are difficult to predict and require tight supervision. Network reliability is the one asset, that a network operator would never risk endangering. The increasing relevance of the Multiservice Internet implies, nevertheless, that there is a strong demand and a strong incentive to bring constant innovation to it, thus introducing additional tension into the networking world.

To augment network flexibility and reduce the complexity of operation, several different approaches have been proposed. Of particular relevance in this context are Active and Programmable Networks (A&PN), for they moreover try to achieve a further goal in network evolution: Openness. From the A&PN perspective, complexity derives also from the fact that network operation, management and update falls into the hands of a centralized instance, the network operator. Their goal is to share those responsibilities among different parties, every one of which will be responsible only for their respective service. This sharing of the network resources implies not only a coordination problem, but mainly a security concern for the incumbent operator and its customers. Besides, these approaches have rarely achieved good performance results, another of the critical problems of modern networking.

The technological trend in communication nodes points toward an increase in the role of hardware and embedded systems for the support of communication tasks. But on top of that, rapidly evolving standards, new services and protocols and in general the necessity to rapidly react to market trends has pushed equipment providers into considering more *programmable* solutions that somewhat prolong the useful life of their (very costly) equipment.

Hence, the goal of this Thesis is to bring together these diverse technological developments to provide the sketch of a novel network model encompassing performance, flexibility, openness, security and QoS, while keeping the complexity at acceptable levels. To that end, a combination of overlay approaches based on programmable network ideas at the network level with programmable hardware at the system level is at the core of the proposal. The main emphasis will reside in a novel resource management architecture for complex hardware systems providing, besides high performance, security and QoS policies.

This work presents the author's vision of an overlay network model able to overcome the limitations and trade-offs mentioned above. It begins by acknowledging the necessity of strong security checks in the form of access control, AAA, encryption, sandboxes, etc. However, in order not to endanger performance and flexibility in exchange for security, it clearly separates service admission and introduction, which is subject to strong security checks, from service management and operation, which is not. Once a service has been deployed, only run-time checks are applied, which critically improves performance with respect to other proposals. Furthermore, performance is also underpinned by the introduction of not only programmable software, but also programmable hardware environments in the network nodes. Flexibility is sustained by the programmability of the platform, which extends to the control plane: Every service operator is allowed to implement its own management interfaces and protocols. The key to achieve flexibility without compromising security at the system level lies in a strong isolation among services and between every service and the node platform itself. In software, that isolation is reached through well-known methods: Sandboxes, Execution Environments and the Operating System. In hardware, those mechanisms are not adequate, for their view of, and hence their control over embedded applications is too coarse. In order to prevent threats coming from the hardware part of the applications, resource management has to happen at the hardware level, too.

As a consequence, the Embedded Hardware Manager is presented. The EHM provides isolation by acting as intermediate element between the applications and the off-chip resources, as well as among the applications themselves. It acts as scheduler, QoS manager and security agent for native hardware applications multiplexed on a common platform. The EHM can work with a wide range of applications and off-chip resources, as long as they have no critical delay requirements in the ns domain.

The EHM has been evaluated by means of a system emulation. Additionally, a prototype has been designed, in order to consider the physical constraints set on the architecture by a real platform. In this way, relative as well as absolute results could be obtained. Isolation, scalability, QoS allocation in terms of bandwidth and delay, resilience to security threats and overall efficiency were measured, as well as overall performance (processed packets per second). The results are very satisfactory in all areas, proving the feasibility and adequacy of the approach

presented in this Thesis. Nevertheless, a number of restrictions also apply. First, delay is the price to pay in exchange for isolation and QoS guarantees. Second, the platform presented a number of inadequacies, which also curtailed the performance achievable by the prototype. Still, the performance lies in the Gbps range, which is much better than previous software proposals. Resource usage is moderate and even small for the latest FPGAs.

Seit dem Aufkommen des World Wide Web hat der Informationsaustausch mittels Rechnernetze eine wahre weltweite Revolution erfahren. Das Internet ist überall vorhanden und unentbehrlich geworden. Seine ursprüngliche Einfachheit im Design ist deshalb verwendet worden, um über dieses Netz eine immer grössere Dienste-Palette mittels einer ganzen Reihe von unterschiedlichen, zum Teil sehr komplexen Technologien anzubieten. Aber gerade dieser Erfolg stellt eine Herausforderung hinsichtlich der Skalierbarkeit von IP-basierten Netze in Bezug auf die Anzahl und die Verschiedenartigkeit der angebotenen Dienste sowie die gesamte Verkehrsbelastung dar.

Zum Ersten wird das Multiservice Internet durch die blosse Verschiedenartigkeit der zu unterstützenden Dienste belastet; obwohl das Internet diese Dienste alle mit akzeptabler Qualität im Einzelnen unterstützen kann, stellt ihre Interaktion ein Problem dar. Zum Zweiten wird die Komplexität des Netzmanagements durch das blosse Volumen der weltweit gleichzeitig aktiven Dienstinstanzen vergrössert. Als Folge verfügt jeder Netzknoten über eine stetig abnehmende Zeit, um eine ständig zunehmende Menge und Verschiedenartigkeit von Daten zu verarbeiten. Dieses Doppelproblem wird durch das wachsende Bedürfnis an Sicherheit und Dienstgüte verstärkt. Schliesslich kann somit eine Innovation, vom Standpunkt des Netzbetreibers aus gesehen, sowohl Segen als auch Fluch bedeuten. Innovation drückt sich in der Erweiterung durch neue Funktionalitäten einer vorhandenen Infrastruktur aus, für die sie aber nicht eingerichtet wurde. Die Folgen sind schwer vorauszusagen und erfordern eine strenge überwachung. Netzzuverlässigkeit ist der Trumpf, den ein Netzbetreiber nie in Frage stellen würde. Die zunehmende Bedeutung des Multiservice Internets bringt mit sich, dass es eine starke Nachfrage und einen hohen Anreiz gibt, um Innovationen ständig weiter einzuführen. Dennoch bewirkt dies eine zusätzliche Spannung in der Netzwelt.

Um Netzflexibilität zu vergrössern und um die Netzmanagement-Belastung zu verringern, sind einige unterschiedliche Vorgehensweisen vorgeschlagen worden. Von besonderer Bedeutung in diesem Kontext sind Aktive und Programmierbare Netze (A&PN), die ein weiteres Ziel in der Netzentwicklung zu erreichen versuchen, nämlich Offenheit. Aus der A&PN Perspektive bewirkt die Netzmanagement-Komplexität, dass Netzbetrieb, Management und Update in den Händen eines zentralisierten Agenten, des Netzbetreibers, liegen. Ziel ist es, jene Verantwortlichkeiten unter unterschiedlichen Parteien aufzuteilen, so dass jede nur für den jeweiligen Dienst verantwortlich ist. Dieses Teilen der Netzressourcen weist nicht nur auf ein Koordinationsproblem, sondern hauptsächlich auf ein Sicherheitsproblem für den Netzbetreiber und seine Kunden hin. Ausserdem haben diese Vorgehensweisen selten gute Leistungsergebnisse erzielt.

Die technologische Tendenz in den Kommunikationsknotenarchitekturen weist auf eine Zunahme der Rolle der Hardware und der eingebetteten Systeme für die Unterstützung der Kommunikationsaufgaben hin. Aber sich rasch entwickelnde Standards, neue Dienste und Protokolle und im allgemeinen die Notwendigkeit,

auf Markttendenzen schnell zu reagieren, hat die Hersteller dazu geführt, mehr programmierbare Lösungen einzubeziehen, die die effektive Betriebsdauer ihrer (sehr teuren) Ausrüstung weiter ausdehnen.

Folglich ist das Ziel dieser Dissertation, die erwähnten verschiedenen technologischen Entwicklungen zusammenzubringen und ein Konzept einer Netzarchitektur vorzustellen, welche Leistung, Flexibilität, Offenheit, Sicherheit und QoS, bei gleichzeitiger Beschränkung der Komplexität auf ein annehmbares Niveau vereinigt. Zu diesem Zweck liegt im Kern der Arbeit eine Kombination von Overlay-Ansätzen, die auf Programmable Networks-Ideen auf der Netzebene und programmierbarer Hardware auf Systemebene basieren. Das Hauptgewicht liegt in einer innovativen Ressourcenmanagementarchitektur für komplexe Hardware-Systeme, die zusätzlich zu hoher Leistung Sicherheit und Dienstgüte (Quality of Service) bereitstellt.

Diese Arbeit stellt die Vision eines Overlay-Netzmodells dar, welches fähig ist, die oben erwähnten Beschränkungen und Kompromisse zu überwinden. Die Arbeit beginnt damit, dass die Notwendigkeit starker Sicherheitsmassnahmen in der Form von Zugriffskontrolle, AAA, Verschlüsselung, Sandboxes, etc. begründet wird. Um jedoch Leistung und Flexibilität zu erreichen ohne die Sicherheit zu gefährden, wird konsequent zwischen Diensteinrichtung und -nutzung (abhängig von den Ergebnissen starker Sicherheitsüberprüfungen) durch Service-Management und Betrieb unterschieden. Sobald ein Dienst eingeführt worden ist, werden nur Laufzeitüberprüfungen angewendet. Als Folge davon wird dadurch eine bedeutende Leistungserhöhung gegenüber anderen Lösungen erreicht. Ausserdem wird die Leistungsfähigkeit zusätzlich durch die Einführung programmierbarer Software und durch programmierbare Hardwareumgebungen in den Netzknoten untermauert. Flexibilität wird durch die Programmierbarkeit der Plattform unterstützt, die in die Steuerebene angreift: Jedem Dienstbetreiber wird erlaubt, seine eigenen Managementschnittstellen und -protokolle einzuführen. Der Schlüssel, um Flexibilität auf Systemebene zu erzielen, ohne die Sicherheit zu kompromittieren, liegt in einer starken Isolierung der Dienstinstanzen untereinander und zwischen jeder Dienstinstanz und der Knotenplattform selbst. In der Software wird diese Isolierung durch weithin bekannte Methoden erreicht: Sandboxes, Execution Environments und das Betriebssystem. In der Hardware sind jene Konstrukte nicht anwendbar, da ihre Sicht von der Plattform und folglich ihre Steuerungsmöglichkeiten über eingebettete Anwendungen zu grob ist. Um die Bedrohungen zu verhindern, die direkt von Hardware-Subsystemen der o.g. Anwendungen herrühren könnten, muss die Ressourcenverwaltung auch direkt auf der Hardware-Ebene geschehen.

Daraus folgend wird in dieser Arbeit der Embedded Hardware Manager begründet. Der EHM schafft die Trennung zwischen Diensten, als Zwischenelement zwischen den Anwendungen und den off-chip-Ressourcen, sowie unter den Anwendungen selbst. Er dient als Scheduler, QoS-Manager und Sicherheitsagent für eingebettete Hardware-Anwendungen, die auf einer allgemeinen Plattform

ablaufen. Der EHM kann mit einer breiten Palette von Anwendungen und off-chip-Ressourcen arbeiten, solange sie keine kritische Randbedingung in Bezug auf Verzögerungen im ns-Bereich einbringen.

Der EHM ist mittels einer System Emulation ausgewertet worden. Zusätzlich wurde ein Prototyp entworfen, um die physikalischen Begrenzungen, die jede reale Plattform hinsichtlich der Architektur besitzt, zu betrachten. Auf diese Art konnten relative sowie absolute Ergebnisse erreicht werden. Isolierung, Skalierbarkeit, QoS Allocation in Bezug auf Bandbreite und Verzögerung, Robustheit gegen Sicherheitsbedrohungen, Effizienz und Leistungsfähigkeit (d.h., verarbeitete Pakete pro Sekunde) konnten gemessen werden. Die Ergebnisse sind in allen Bereichen sehr zufriedenstellend und unterstreichen die Machbarkeit und Angemessenheit der Lösung, die in dieser Dissertation vorgeschlagen wird. Dennoch treffen eine Anzahl von Beschränkungen durchaus zu. Isolierung und QoS-Garantien werden einerseits durch zusätzliche Verzögerung geschaffen. Andererseits wies die Plattform eine Anzahl von Unzulänglichkeiten auf, die auch die Leistung, die durch den Prototyp zu erreichen ist, beschränken. Dennoch liegt die erreichte Leistung im Gbps-Bereich, eine deutliche Verbesserung zu bisherigen Software-Realisierungen. Der Ressourcenverbrauch ist mässig und für die neuesten FPGAs sogar gering.

Contents

List of Acronyms

AA	Active Application
AAA	Authentication, Authorization and Accounting
ACL	Access Control List
AES	Advanced Encryption Standard
AF	Assured Forwarding
AHP	Advanced Hardware Platform
AHPM	Advanced Hardware Platform Module
AN	Active Network
ANTS	Active Node Transfer System
A&PN	Active & Programmable Networks
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
ATM	Asynchronous Transfer Mode
b	Token Bucket Depth
BHP	Basic Hardware Platform
BIOS	Basic Input/Output System
CIDR	Classless Interdomain Routing
CDN	Content Delivery Network
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CR-LDP	Constraint Based Label Distribution Protocol
DAN	Distributed Code Caching for Active Networks
DER	Dynamically Extensible Router
DES	Data Encryption Standard
DD	Device Driver
DHP	Dynamic Hardware Plugin
DiffServ	Differentiated Services
DMA	Direct Memory Access
DNS	Domain Name Service
DRAM	Dynamic Random Access Memory
DSCP	Differentiated Services Code Point
DSL	Digital Subscriber Line

DSP	Digital Signal Processor
DVI	Digital Video Interface
EDIF	Electronic Design Interchange Format
EE	Execution Environment
EF	Expedited Forwarding
EHM	Embedded Hardware Manager
FEC	Forward Error Correction
FHiPPs	Flexible High Performance Platform
FIFO	First-In, First-Out
FilterSpec	Filter Specification
FlowSpec	Flow Specification
FPGA	Field Programmable Gate Array
FPX	Field-Programmable Port Extender
FTP	File Transfer Protocol
Gbps	Gigabit per second
GMPLS	Generalized Multiprotocol Label Switching
GPP	General Purpose Processor
HDL	Hardware Description Language
H-FSC	Hierarchical Fair Share Curve
IC	Integrated Circuit
ICMP	Internet Control Message Protocol
IKR	Institute of Communication Networks and Computer Engineering
IntServ	Integrated Services
I/O	Input / Output
IOS	Internetworking Operating System
IP	Intellectual Property
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ISDN	Integrated Services Digital Network
IS-IS-TE	Intermediate System-Intermediate System – Traffic Engineering
ISO	International Standards Organization
ISP	Internet Service Provider
JTAG	Joint Test Action Group
Kbps	Kilobit per second
Mbps	Megabit per second
MAN	Metropolitan Area Network
MBS	Maximum Burst Size
MIB	Management Information Base
MMU	Memory Management Unit
MPLS	Multiprotocol Label Switching
MSoC	Multiple Systems on a Chip
NAN	Network Admission Node

NAPI	Network Application Programming Interface
NAPT	Network and Port Address Translation
NAT	Network Address Translation
NID	Network Interface Device
NIST	National Institute of Standards and Technology
NP	Network Processor
OCP	Open Core Protocol
ONM	Octopus Network Model
OOG	Octopus Open Gateway
OS	Operating System
OSI	Open Systems Interconnection
OSPF-TE	Open Shortest Path First – Traffic Engineering
P4	Programmable Protocol Processing Pipeline
PC	Personal Computer
PCI	Peripheral Component Interconnect
PLAN	Programming Language for Active Networks
PLL	Phase-Locked Loop
PNNI	Private Node Network Interface
QoS	Quality of Service
r	Token Bucket Rate
RAD	Reprogrammable Application Device
RAM	Random Access Memory
RSpec	Request Specification
RSVP	Resource Reservation Protocol
RSVP-TE	Resource Reservation Protocol – Traffic Engineering
RTCP	Real Time Control Protocol
RUM	Resource Usage Manager
SANE	Secure Active Network Environment
SDH	Synchronous Digital Hierarchy
SDK	Software Development Kit
SDRAM	Synchronous Dynamic Random Access Memory
SLA	Service Level Agreement
SME	Small and Medium Enterprise
SNMP	Simple Network Management Protocol
SIIM	Service Introduction & Interaction Manager
SIP	Session Initiation Protocol
SoC	System on a Chip
SOHO	Small Office / Home Office
SPC	Smart Port Card
SRAM	Synchronous Random Access Memory
Svc Mgr	Service Manager
TCP	Transfer Control Protocol
ToS	Type of Service

TSpec	Traffic Specification
TTL	Time To Live
UDP	User Datagram Protocol
UHP	Universal Hardware Platform
VCI	Virtual Connection Identifier
VHDL	Very Large Scale Integration Hardware Description Language
VLSI	Very Large Scale Integration
VM	Virtual Machine
VOQ	Virtual Output Queuing
VPI	Virtual Path Identifier
VPN	Virtual Private Network
WDM	Wavelength Division Multiplex

List of Figures

1 Introduction

1.1 The Multiservice Internet

Since the advent of the World Wide Web, information exchange by means of computer networks has experienced a true worldwide revolution. The Internet has become ubiquitous and indispensable. Its original simplicity of design has been used to build around it a very rich, ever expanding set of services, with the help of a tremendously complex and diverse range of technologies. It is precisely this unheard-of success, which is severely putting to the test the ability of IP-based networks to scale with respect to the number and diversity of services that can be offered on top of them, as well as the aggregate volume of traffic that can be processed by and transported through the network and to the final user.

First of all, the *Multiservice Internet* [1] is strained by the sheer diversity of services that should be supported: Whereas in the beginning it was only exchange of raw data, it now encompasses voice and video communication, file searching and exchange, instant text communication, a myriad forms of e-business, sensor network control, and more. Although all of them can separately make use of the Internet to run acceptably, it is their interaction that represents a problem.

First, they are not isolated from each other but have to share a common infrastructure. Every service has different requirements in terms of the network characteristics, usually expressed in the form of their sensitivity to delay, jitter or lack of adequate bandwidth, but also in their security or quality of service requirements. The superposition of different service instances thus implies the necessity of accurate network management by the network operators. The more services are supported, and the more diverse they are, the more complex this task becomes, to the point that up to 50% of all operational costs of a network operator fall into this category.

But service diversity is not the only problem. Secondly, the complexity of network management is aggravated by the sheer volume of service instances that are active simultaneously worldwide. It is not uncommon for a core router in any backbone network to support between 100.000 and 500.000 flows simultaneously

[1] The term "Multiservice Internet" will be used throughout this work to refer to the compound of mainly IP-based networks, protocols, services and applications, that are currently and in the near future used for the mass exchange of information by means of computer networks.

at any given time. Every connection has its own requirements concerning the functioning of the network that, in principle, need to be honored separately. This augments the management complexity strongly. At the same time, the ever expanding use of the network implies an ever expanding amount of traffic to be processed and transported. As a consequence, every network node has an ever decreasing amount of time to process an always increasing amount and diversity of information. This dual problem is aggravated by the increasing concern by the users regarding security and Quality of Service (QoS). These concerns add tight boundary conditions to the way in which data can be handled, and at the same time increase the amount of processing needed for every bit of information.

Last but not least, innovation is, from the point of view of the network operator, a blessing but also a jinx. Innovation implies the introduction of new functionality in an existing infrastructure, for which it was not built. The consequences are difficult to predict and require tight supervision. Network reliability is the one asset, that a network operator would never risk endangering. The increasing relevance of the Multiservice Internet implies, nevertheless, that there is a strong demand and a strong incentive to bring constant innovation to it, thus introducing additional tension into the networking world.

To summarize, the Multiservice Internet presents hard requirements in terms of scalability (in terms of number and diversity of service instances), flexibility and performance while respecting the users' security and QoS concerns. A further downside is the increasing complexity of operation. As stated above, nevertheless, innovation also presents ways to soothe these tensions.

1.2 A Glimpse of the Future, a Glimpse of the Past

To augment network flexibility and reduce the complexity of operation, several different approaches have been proposed, as will be presented in chapter 3. Suffice it to say, that Overlay Networks, in one form or another, all pursue these goals. Of particular relevance in this context are Active and Programmable Networks (A&PN), for they moreover try to achieve a further goal in network evolution: Openness. From the A&PN perspective, complexity derives also from the fact that network operation, management and update falls into the hands of a centralized instance, the network operator. Their goal is to share those responsibilities among different parties, every one of which will be responsible only for their respective service. This sharing of the network resources implies not only a coordination problem, but mainly a security concern for the incumbent operator and its customers. Besides, these approaches have rarely achieved good performance results, another of the critical problems of modern networking.

The technological trend in communication nodes points toward an increase in the role of hardware and embedded systems for the support of communication tasks. Although pure software systems have increased their performance impressively in the last few years, application-specific hardware solutions for processing-

intensive operations still achieve orders of magnitude better results. In order to combine that impressive performance with flexibility, a number of more or less programmable hardware platforms have been developed. Their main elements are typically DSPs, Network Processors (NPs), FPGAs or combinations thereof. The increasingly short development and life cycles of communications systems, derived precisely from the innovation drive, has made the ASIC option increasingly less appealing. The development costs for ASICs have dramatically increased in recent times, to the point that hundreds of thousands of units have to be sold to reach the investment break-even point. But on top of that, rapidly evolving standards, new services and protocols and in general the necessity to rapidly react to market trends has pushed equipment providers into considering more *programmable* solutions that somewhat prolong the useful life of their (very costly) equipment.

An additional factor in this search for hardware support has been the technological evolution of ICs. In recent years, the evolution in chip size, on-chip memory and gate density opened the door to the integration of whole systems on a single chip (SoC), thus facilitating the shift of whole tasks to the hardware domain. At present, the sustained development in VLSI technology allows to go one step further: The advent of Multiple-Systems-on-a-Chip (MSoC). State-of-the-art chips can easily support several embedded micro-processors and a number of specialized hardware modules concurrently. And yet, new constraints have to be taken into account in this environment: The number of I/O pins grows at a slower pace than gate density, provoking a mismatch in I/O bandwidth and processing power. Furthermore, multiple systems (or subsystems) will have to share those pins. An efficient way of sharing access to external resources is needed.

In this context, the way in which interfaces and the access to off-chip resources (e.g. memory, I/O, external co-processors) are managed becomes critical for the successful introduction of MSoCs. The emphasis so far has been mainly on finding ways to extract the maximum possible performance (both in terms of bandwidth and delay) out of those resources. While performance is clearly a must, it is not sufficient. When several independent systems are sharing a chip, two other criteria become critical: Security and QoS, just as it happens in other aspects of networking. As with multitasking operating systems on a CPU, it is unacceptable that a malicious (or simply misbehaving) component could block the memory interface for the entire chip, or that it could flood the communication channels with data, that ignores the SoC arbitration mechanism. Or even that it would try to access reserved regions in off-chip memory that were granted to other systems. QoS is also paramount: Different systems will have different requirements in terms of delay sensitivity, performance, bandwidth, etc. Classical approaches tend to share those resources fairly among competing systems. That might not be enough in this new scenario, in which a more complex scheduling may be needed to honor the different requirements of the systems.

1.3 The Octopus Network Model

The goal of this Thesis is to bring together these diverse technological developments to provide the sketch of a novel network model encompassing performance, flexibility, openness, security and QoS, while keeping the complexity at acceptable levels. To that end, a combination of overlay approaches based on programmable network ideas at the network level with programmable hardware at the system level is at the core of the proposal. The main emphasis will reside in a novel resource management architecture for complex hardware systems providing, besides high performance, security and QoS policies.

1.4 Summary of Contributions

This work makes the following main contributions: First, it presents a new characterization and taxonomy of router tasks and the mechanisms used to implement them. This is followed by a thorough analysis and description of the main elements composing a modern router architecture. These are necessary prerequisites to identify how to integrate "active" functionality in future node architectures.

Second, this work reviews the main principles of Active & Programmable Networks and analyzes why they were not successfully deployed in real networks. The main weaknesses are identified in an apparently unsurmountable trade-off between openness, performance and security.

Third, the Octopus Network Model is introduced. It represents a novel Programmable Network architecture, which overcomes the trade-off mentioned above. Its main characteristics are the separation between service admission and service management and the introduction of a secure hardware programmable platform.

Fourth, a novel node architecture, denoted Octopus Open Gateway, is presented. It integrates "active" modules in the overall router architecture and provides openness, performance and security at the node level.

Lastly, as a fundamental new element of the node architecture, a resource management module for programmable hardware systems, called the Embedded Hardware Manager has been designed and evaluated by means of hardware-close functional and performance simulations. On top of the simulation studies, a prototype realization has been designed. The Embedded Hardware Manager is the cornerstone of the Octopus Open Gateway, ensuring openness and node sharing without endangering performance or security.

1.5 Outline of the Thesis

The remainder of this Thesis is structured as follows: Chapter 2 analyzes the area of router design and architectures and presents a novel characteriza-

tion thereof. Chapter 3 reviews the relevant previous research work most related to this Thesis. It analyzes the general aspects of Active and Programmable Networks. The main similarities and differences to this work will be presented. Additionally, an evaluation of the security threats to routers derived from system programmability, as well as an introduction to QoS in IP networks is included. Further details on the proposals closest to this project have been included in Annex A. Chapter 4 presents the Octopus Network Model and its main components. Chapter 5 concentrates on the design of the main architectural element, the Octopus Open Gateway (OOG). Chapter 6 then introduces the OOG's resource management architecture for hardware systems, while chapter 7 presents an evaluation of it. Chapter 8 finalizes the Thesis.

2 The Role of Routers in IP Networks

This Thesis explores new avenues in communication node design. Paramount among such nodes is the router, which has served as cornerstone of IP networking for over thirty years. In this chapter, a novel analysis and taxonomy of the role of routers in networking will be provided. Special attention will be paid to all aspects of router design: Its basic functionality, the history of its evolution, the main architectural characteristics and trade-offs and the technological trends.

2.1 Router Tasks

A router is an interworking unit, which main task is to forward incoming traffic, namely datagrams, in the best direction toward their final destination. This function is called *routing* [ITU94b]. Nowadays the Internet and its basic underlying technology, the so-called "TCP/IP protocol stack" [Pos81], dominate data communications. For this reason, this discussion will be centered on the role of routers in the Multiservice Internet, in order to become more concrete. These network elements work on layer 3, or the Network Layer, of the Internet reference model [Tan00]. The datagrams they process are accordingly IP datagrams, or *packets*.

With the evolution in Internet functionality, nevertheless, the role of the router has changed: Beyond providing reachability and efficient packet forwarding, a number of additional tasks have appeared. They are the consequence of the development from a unique, common, best-effort treatment of all traffic toward a differentiated processing of it. On the one hand, the different services supported by the Multiservice Internet have very different requirements regarding network behavior. This certainly pushes the need to differentiate the traffic corresponding to the different services and providing them with individual treatment. On the other hand, the commercialization, individualization and worldwide spreading of the Internet experience have brought the demand for user-tailored traffic handling, for which the user is also willing to pay a price. A supporting set of security, accounting & billing and quality-guaranteeing architectures have thus emerged. Routers being the elements in charge of accepting packets into and then forwarding them across the network, they become also responsible for the

realization and support of this differentiation. The tasks of a modern router can thus be classified as follows (see Fig. 2.1):

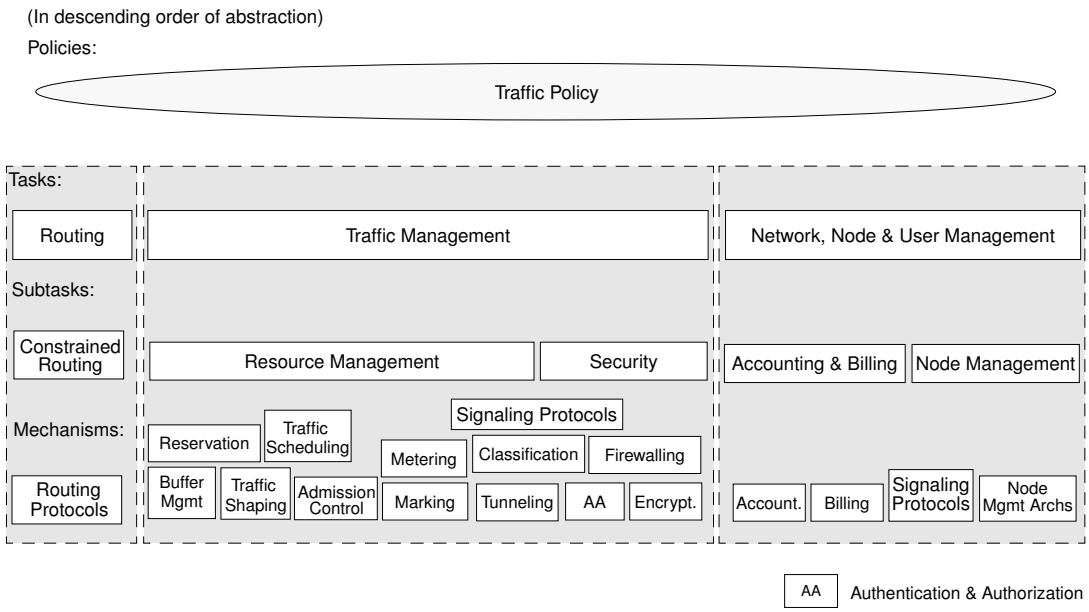


Figure 2.1: The Functions of a Modern Router

Traffic Policy. The starting point to define the router tasks and the methods to implement those tasks is the establishment of an overall traffic policy by the network operator. A *policy* defines in somewhat abstract terms the goals, acceptable methods of action and the set of rules employed when administering a network [WSS+01]. A network policy is not restricted to technical aspects, but can also involve business decisions. To a traffic policy belong elements like: the granularity of traffic differentiation, which kind of guarantee is going to be given to a certain traffic aggregate (absolute/relative), which security aspects shall be covered, which information is desired regarding billing (per call, per user, etc.), to name just a few. The mechanisms and corresponding configurations employed to implement this policy can be grouped in three blocks, that will be referred to as *router tasks*: Routing, Traffic Management and Network, Node & User Management.

Routing. From the overall traffic policy a certain routing policy is derived. As already stated, routing a packet consists on, according to the information carried therein and the routing policy, calculating the best path to reach its final destination and forward it along that path. The routing policy establishes a number of constraints on how this "best path" is to be calculated. The most common one is simply the minimization of the path length, measured as the number of routers traversed (so called "hops"). It is nevertheless possible to use several criteria simultaneously (e.g. maximum available bandwidth with minimum total distance) to establish the path. Furthermore, a router can support different

routing strategies for different traffic aggregates or even for different purposes. The routing task would then be divided in a number of *constraint-based routing subtasks*, as stated in Fig. 2.1. These subtasks are implemented by means of different *routing protocols*, which are the *mechanisms* used eventually to translate the routing policy into router configurations and actions.

Traffic Management. In the past, the previous task was the main occupation of every router. Nowadays, that role corresponds to Traffic Management. In the sense defined in [Cisa], traffic management represents the set of techniques used to make traffic compliant with the goals specified in the traffic policy. More specifically, the distribution of network resources among the different traffic aggregates is one of its main objectives. It serves two purposes: First, deliver to every traffic aggregate its corresponding service level (s. Section 3.3), according to the traffic policy. Second, try to achieve high network utilization, avoid network congestion (Traffic Control, [ITU02]) and, in case of necessity, redress that situation (Congestion Control, [ITU02]). Together, these two aspects will be summarized as the subtask *Resource Management*. Typically, the resources to be managed in a data network are the link bandwidth and the buffer space in the routers, tied to certain boundary conditions regarding maximum acceptable delay and jitter for every traffic aggregate. Another relatively new concern in traffic policies involves the definition of a *security* policy (s. Section 3.2).

Since the breadth of this task is enormous, the number of mechanisms employed for that purpose is also huge. The main ones will be shortly defined and enumerated here, beginning by those corresponding to the subtask **Resource Management**:

- Reservation: The most fundamental form of resource management is the explicit reservation of a certain amount of resources for a traffic aggregate. This may be static (performed by the network administrator) or dynamic (using signaling protocols). Due to the bursty nature of most data sources, there is an interest by the network operators to reserve less than the peak amount of needed resources, in order to achieve a multiplexing gain. In that case, a statistical instead of an absolute guarantee in access to those resources is given [Kel00], [BZB+97], [Wro97].

- Traffic Scheduling: Although inside a router a logical (and often even physical) differentiation among traffic aggregates takes place, packets finally have to be serialized prior to sending them on the link. The amount of packets sent from each aggregate per time unit sets the bandwidth allocated to it (generally according to a previous reservation), while the order in which they are sent influences the delay and jitter that an aggregate experiences. It is the role of an scheduling algorithm to control those decisions, so that reservations will be respected [Zha95], [ITU02].

- Traffic Shaping: While the role of traffic scheduling consists in servicing aggregates so that reservations will be honored *among* them, traffic shaping delays packets *within* a traffic aggregate, so that they better suit network requirements. Burstiness, e.g., provokes high traffic peaks that might exhaust buffer space and/or provoke congestion in the network during short periods of time. A controlled reduction of the source's send rate to assuage those bursts can go a long way in reducing that risk [BBC+98], [ITU02].

- Admission Control: Admission control implements the decision algorithm that a router uses to determine whether a new aggregate can be granted the requested service level without impacting earlier guarantees given to other aggregates. This algorithm must be consistent with the type of guarantee being supported (absolute/relative) and with the granularity of reservation (individual flows or larger aggregates). This function can be performed once, at the entrance point to the network, or on a local basis, depending on the reservation mechanisms supported [BCS94], [ITU02].

- Buffer Management: As will be explained in more detail in 2.2, packets usually have to be queued prior to processing at different points in the router. An obvious reason for that is the effect of a traffic scheduler, as explained above. In the absence of buffers for the interim storing of queued datagrams, packet loss would result. Buffer management thus defines the strategy, with which the space in those buffers is allocated to the different traffic aggregates [RLG98]. Buffering presents a trade-off between packet loss and packet delay and jitter: The larger the buffers, the smaller the packet loss probability, but the longer the delay until a packet is served. The convenience of a certain buffer management strategy depends in part on the service characteristics accorded to a traffic aggregate. In a sense, buffer management dictates how packets are written into the buffers, while traffic scheduling defines how and when they are read out of them.

- Metering: Admission control is performed to ensure that guarantees given to existing aggregates can be held by eventually preventing *new* aggregates to enter the network. However, an additional mechanism is needed to ensure, that *existing* aggregates are not making use of more resources than agreed upon. For that purpose, the temporal properties of a traffic stream are measured and averaged in short as well as long intervals [BBC+98]. Should a breach of contract be established, it is the function of the marker, scheduler and shaper to implement the traffic policy in this respect.

- Marking: In order to differentiate aggregates, an implicit (through classification) or explicit (through classification and the inclusion of some tag into the packet itself) marking has to be performed. This mark implicitly defines the type of service to be provided to that packet or even to the whole

aggregate [BBC+98], [ITU02]. As way of example, IPv6 [DH98] includes a flow descriptor in its header, while the Differentiated Services architecture [NBBB98] provides for a tag in the IPv4 header. Alternatively, RSVP [BZB+97] uses implicit labels stored in the routers, but not in the packets.

The subtask **Security** has strongly contributed to the increase in the number of mechanisms to be supported by routers, for it represents a completely new dimension in router functionality. The following mechanisms can be cited:

- Firewalling: Although not every router serves as a firewall, all of them perform at least part of the tasks associated thereto. Its mission is to act as an interface between two networks (typically a public, insecure network and a private network) and regulate traffic between those networks for the purpose of protecting the internal network from electronic attacks originating from the external network [ANS]. To that end, a number of subtasks are performed, which will not be further dealt with here [1].

- Authentication: Authentication is considered to be among the most relevant security services, for many others rely on it in some form. Authentication provides assurance of the identity of a claiming party as requested by a verifying entity [For94]. It can be distinguished between *entity authentication*, in which a party involved in a communication presents a claim to a certain identity, and *data origin authentication*, in which the originating instance of the data is validated.

- Authorization: Authorization can be defined as the granting of rights concerning access and manipulation of a certain resource by a foreign entity [For94]. Here resource is used in a very broad sense, encompassing, among others, communications capabilities, data or control information associated with the configuration of a device (especially the router itself).

- Encryption: Among the newest additions to router functionality is the support for data encryption. The aim of this mechanism is the transformation of data in order to hide its information, prevent its undetected modification or its unauthorized use [For94].

There is a third group of mechanisms, which are both used for Resource Management and Security, and are thus listed separately:

- Classification: The cornerstone of differentiated traffic handling is the ability to discriminate incoming packets by inspecting their content and subsequently grouping them in separate classes [BBC+98]. It is fundamental for the delivery of different quality levels to different aggregates as well as for

[1] A non-exhaustive list of those subtasks reads as follows: Proxying, encryption, tunneling, filtering, monitoring and blocking.

the implementation of security policies (separate dangerous from conventional traffic, say) and accounting (e.g., summarizing traffic from a certain kind or coming from a certain source). Because of its central role, it will be dealt with in more detail in Section 2.2.

- Tunneling: Since routers base the decision on how to process packets basically on the information contained in the header, a widespread method of insulating aggregates from the default treatment by a router consists in encapsulating the original packet in a new one, with a header chosen accordingly to the desired treatment [Com00]. The reason to do so can, once again, reflect quality, routing or security concerns, or even technological necessity. This is the case, when two networks using the same protocols are interconnected by a third one using a different stack.

- Signaling Protocols: The signaling process encompasses the interchange of control information concerning the management of traffic aggregates or of the network as a whole [ITU93]. As such, in order to coordinate the decisions taken on every router on a local basis, signaling protocols are the most common way used to support all other mechanisms described so far. On an logical plane, they represent a parallel network for the purpose of network and traffic control.

Network, Node & User Management. "Management" is a very broad and imprecise term. The sense given to it by the OSI terminology [ITU94b], embodies a concern for the problems of initiating, terminating and monitoring network activities, as well as assisting in their harmonious development. It also comprises the handling of abnormal operations. A previous definition of Network Management [KKR97] includes in the term five subtasks: Configuration Management, Performance Management (including QoS therein), Security Management (mainly resource access control), Fault Management and Accounting Management.

Traffic Management, defined above, deals with one aspect of this definition, namely the supervision of the smooth network operation in the *data plane*: Its activities are directly related to the handling of data traffic and the mechanisms associated thereto. In this sense, it comprises parts of the Configuration, Performance and Security Management, mainly.

Network, Node & User Management (Network Management, for short), deals with the smooth and correct operation of the network elements themselves as well as the compilation and distribution of accounting information. It then reflects mainly the Fault and Accounting Management named above, as well as aspects of the Configuration Management. The emphasis here is away from the data and concentrated on the network as an abstract machine, which, independently of its function, has to be kept in proper working order. In this sense, its subtasks can be subdivided in:

- **Accounting and Billing.** This subtask comprises the compilation of such information concerning the network operation as are deemed necessary for the accounting and billing processes. These aspects of network operation fall outside of the scope of this work and will therefore not be further addressed in this Thesis.

- **Node Management.** It comprises the supervision of the node itself as well as its configuration, by means of some *Node Management Architecture* as well as the use of *Signaling Protocols*, mainly to report status conditions to some centralized supervisory instance. These are also aspects that are not central to this work and will not be further deepened.

In this section, the main tasks realized by a router have been reviewed. The mapping of those abstract tasks to a system architecture is the topic of the next section.

2.2 Router Architectures

Modern routers share a common basic architecture [Awe01], [FKLS98], [KS98], [KLS98], [PCB+98], [Par98]. The main differences among them are found in the realization of the modules presented therein, although even at the module level similarities abound. This section will analyze the main characteristics of modern router architectures, pointing out the main trade-offs in router design. First, a brief overview of the architecture will be given, while details on the different modules will be presented in subsequent subsections.

2.2.1 Basic Router Architecture

Figs. 2.2 and 2.3 represent the progress of a packet through a router and the functions performed upon it. A number of basic modules can be identified, each module performing a set of tasks according to the description presented in the previous section. These basic modules correspond to a purely functional view of a router. The physical realization of this functional architecture will be described in section 2.3.

When a packet arrives at a router interface, the first module it encounters is the *Packet Classifier*. Its main functions consist in validating the formal correctness of the packet and then classifying it as belonging to a certain traffic class, so that it can be treated according to its corresponding service level. Next, the optimum outgoing interface to reach its final destination is chosen in the so-called route lookup process. Assuming that the packet was accepted for forwarding, it will be adapted to the internal format used in the router (i.e. fragmented in smaller chunks, appropriate for the internal bus width) and sent to the next module. At any point in the process, a packet can be discarded (as represented in the figure by the "ground" connections). It is nevertheless convenient to detect

invalid packets at an early processing stage, so that resources inside of the router can be spared for valid packets. To this end, as many invalid packets as possible should be discarded inside the Packet Classifier.

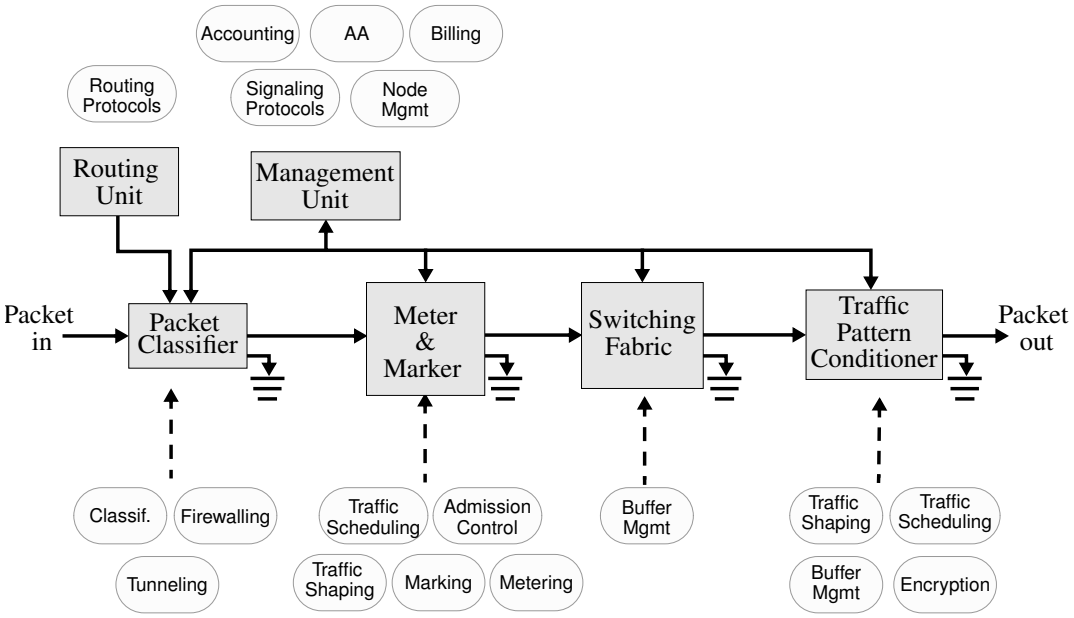


Figure 2.2: Block Diagram of a Router: Functional Description.

In order for the Packet Classifier to perform the route lookup, the *Routing Unit* runs a set of routing protocols, with which information it builds a routing table containing the correspondences between addresses and output ports in the router. This information is then passed on to the Packet Classifier.

When a data packet is considered to be apt for forwarding, it reaches the *Meter & Marker*. Here, the behavior of the traffic flows with respect to their accorded service levels is checked. Should a certain traffic stream, i.e., exceed its accorded capacity, it corresponds to the Meter & Marker to reclassify the packet as stated by the traffic policy, drop it, or schedule it as appropriate. The basic element of a Meter & Marker is a Statistics Unit, which keeps track of resource usage and also gathers the necessary information for the Accounting and Billing processes. Furthermore, the Meter & Marker buffers the packet (or a pointer thereto) in the corresponding queue for its class at the entrance of the *Switching Fabric*.

The Switching Fabric connects all input interfaces with all output ports, respecting the QoS policies implemented by the Meter & Marker, i.e., serving its input queues according to their priority, as defined by an allocation algorithm. The switch can be as simple as a bus or as complex as a multi-stage switching network. After traversing the switch, packets are queued inside the *Traffic Pattern Conditioner*. This unit receives packets destined to the same port (or set of ports) and is in charge of serializing their departure. For that purpose, it

implements a set of scheduling and shaping algorithms, which translate the QoS policy set by the router administrator.

A number of circumstances can conduce a packet to the *Management Unit*. Packets with unknown characteristics and yet not deemed erroneous is one of them. The Management Unit runs a number of protocols, that deal with these special cases. On top of that, all signaling packets are sent to this Unit for processing. Lastly, the management of the node itself is also realized here, as is the communication with a centralized network management instance outside of the node. The relevant information for this task is gathered at the different router modules and then polled by the Management Unit as needed.

2.2.2 The Packet Classifier

A router is a costly investment, especially if the number of ports and/or the number of services that it offers are high. In order to optimize the investment that this equipment represents, routers usually must work under heavy load conditions. This implies that its resources (especially its processing power) are scarce and must therefore be sensibly used. For this reason, one of the most important tasks of a Packet Classifier is the detection of corrupted or otherwise invalid packets, so that they can be dropped as early as possible in the routing process. The first set of tests that a Classifier performs on a packet are the so-called sanity checks (s. Fig. 2.3). The formal validity of the packet is tested by asserting that:

- The IP version number corresponds to the one supported
- The IP header has the right length
- The total packet length is bigger than the header length
- The destination IP address contained corresponds to a standardized format (i.e., to one of the 3 standard classes, A, B and C [FLYV93]).
- The CRC is valid. For efficiency reasons, it is usually updated at the same time.
- The TTL value is valid. It is usually also decreased simultaneously.

These checks can be performed in parallel, which greatly contributes to minimize the processing delay inside of a router.

Should a packet be considered formally valid, it corresponds to the router, in the simplest case, to find an adequate output port to forward it. For that purpose, the Routing Unit (s. Section 2.2.6) downloads a summary of all reachable destinations and their corresponding output interfaces into the Packet Classifier. This table is known as the *forwarding table*. In principle, different forwarding tables can be downloaded and used for different aggregates, thus reflecting their separate requirements. However, this rarely is the case in commercial routers.

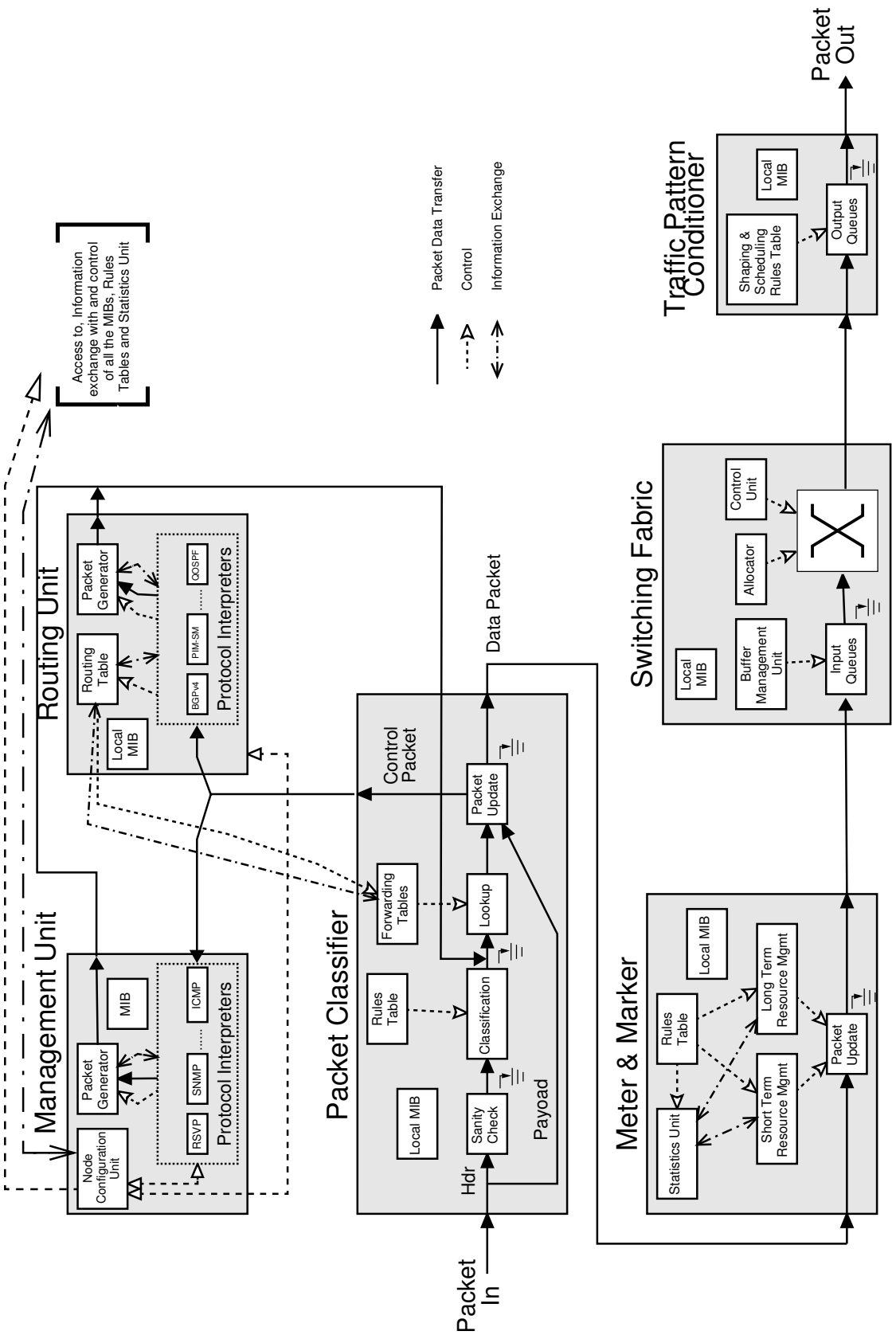


Figure 2.3: Block Diagram of a Router: Detailed Description.

The task of searching the forwarding table for the best route is called *route lookup*. Until around 1997 [DBCP97], [WVTP97], it was considered one of the most critical router functions, since it is a complex operation that required much precious time. The reason lies in the abolition of network classes and the introduction of masks [FLYV93]. Classless Interdomain Routing (CIDR) allowed networks of any size to be specified with the help of a network mask marking the length of the network and host parts on an IP address. This allows, that networks of different size share a common prefix (the first bits of the address), although belonging to different institutions and residing in different locations. Route lookup thus implies the search for the longest among all prefixes contained in the forwarding table, that matches the destination IP address of every incoming packet. It is this non-exact search procedure, which is very time-consuming. Nevertheless, increasingly sophisticated algorithms produced a breakthrough in the late 90s that seemed to solve the issue by increasing the performance of the lookup process several orders of magnitude [TP99], [LSV99]. Most algorithms represent variations of the classical binary tree search algorithm [Sed97].

An additional mechanism employed to accelerate the lookup process was the use of caches. The main rationale was, that traffic presents a high degree of locality, i.e., there is a high probability, that more than one packet destined to the same address will reach the router within a short time [Awe01]. Thus, keeping a cache of recently seen addresses and their corresponding output interfaces could highly increase the performance of the lookup process. Nevertheless, nowadays the advances in lookup algorithms and a loss of effectivity in caching make this mechanism less relevant. Caching is losing effectivity because of the increase in network capacity and reduction in locality: The more packets arrive per second destined to different locations, the bigger should be the cache to maintain the hit rate. The price/performance ratio of caches quickly diminishes with the cache size due to technological limitations.

If differentiation is to be supported in a router, *packet classification* is the fundamental mechanism. The first step toward delivering differentiated service treatment is the ability to categorize packets into different "groups" (classes, flows, and so on). Each one of these groups of packets subsequently receives different treatment in the router. Only after this classification mechanisms can be applied to ensure a certain preferential treatment in terms of bandwidth, delay, or any other criteria. Thus, if it is not achieved at wire speed, packet classification can become the processing bottleneck of the router. The delivery of differentiated treatment in the IP world is centered on the concept of flow[2] [CAB99], [DH95]. It represents a traffic stream that should be treated with the same service level. A flow is identified by the content of two or more fields in its header. For example, a

[2]By this it is not meant, that differentiation always happens at this level of granularity. A flow represents, nevertheless, the *highest* degree of granularity, that common differentiation strategies propose.

flow transporting a video communication between two people could be defined by their destination and source addresses, plus the port numbers and the protocol field. Even if traffic is aggregated in a small set of classes, there are points in the network (the edges) where the aggregation/disaggregation of the traffic has to take place. At those points, a potentially huge number of flows have to be individually classified. Because the classification process has to be repeated for every incoming packet and because the number of simultaneously active flows in a router can be enormous, packet classification has tremendous performance constraints.

Classification is done according to a set of filters (also called rules). These rules describe the characteristics that define every traffic class in terms of the header content. The usual definition of a filter F is an n-tuple of bit patterns, either prefixes, ranges or exact values. Each element of the tuple, F[i], i$\in$[1, n], is associated with a header field H_i in an IP packet. A packet matches a certain rule F, if for all i, the i-th field of the header matches F[i]. Since rules can overlap, more than one filter can match a certain packet. Thus, filters need to have a unique priority that allows such conflicts to be solved. Given a set of rules, packet classification consists in finding the rule with the highest priority among those matching an incoming packet.

In judging the quality of a classification algorithm, there are three main variables to consider: The speed at which a matching filter is found (in the average as well as in the worst case), the storage requirements of the chosen solution, and the time it takes to make an update in that structure (i.e., to add or remove a filter or rule). From these three criteria, most of the classification proposals to date have attempted to find a satisfactory compromise between the first two at the cost of very expensive updates, with rare exceptions. This seems like a dangerous approach because the basic unit to be classified is the flow, and a flow is intrinsically dynamic. For example, if a bandwidth reservation request for a certain flow triggers the insertion of a new rule in the classifier, a delay of several seconds would be unacceptable.

We divide the classification proposals found in the literature into four schools (see [MF01] and the references therein):

- Packet classification can be seen as an extension to multiple dimensions (multiple fields) of the IP route lookup problem. The most common approach to that problem, as stated above, has been to use search trees, based on the concept of binary trees. Since very successful algorithms have been developed in recent years for this, many efforts have been devoted to extending such techniques to the more general case of packet classification.

- Another school of thought sees the multidimensional matching problem as a special instantiation of the point location problem in a multidimensional space, or in general takes geometrical space decomposition approaches.

- Other solutions rely on a strong analysis of the content of a concrete dataset to precompute an optimal search structure.

- A reduced set of essentially different approaches also exists.

In principle, the mathematical problem of fast packet classification in the general case remains unsolved, especially for large sets of rules. Nevertheless, advances in technology coupled with algorithms tailored for explicit applications have proved to be sufficient for most users. The expected explosion in the size of rule datasets has also not occurred. One reason lies in the failure of QoS traffic delivery to the last mile, which would have provoked the need to perform fine-grained classification at every edge router. Even then, classification in large aggregates could keep the problem at bay. The other main application of this technique is firewalling, which has indeed seen an enormous increase in relevance due to a boom in networking security threats (s. Section 3.2). However, Firewalls usually contain no more than a few thousand rules, still small enough for most modern algorithms and systems to deal with.

The rules datasets reflect the traffic and security policies set by the system administrator. They are usually introduced offline and can be static for long periods of time. Firewalls and other traffic filtering applications, however, can introduce a high degree of dynamics in the rule dataset structure. It should be noted, that the classifier only differentiates traffic streams according to the *existing* set of classes. New classes and thus new rules might be created dynamically in different scenarios. The detection of denial-of-service attacks, e.g., may trigger the introduction of a new rule for the dropping of malicious packets. Traffic coming from a new customer should be accounted & billed for, etc. The definition of new traffic classes and the corresponding filters, however, is not the responsibility of the Packet Classifier. Either they are introduced offline, or they are dynamically introduced by the Meter & Marker (see next section).

2.2.3 The Meter & Marker

The main mission of this module consists in keeping track of the resource usage by the different traffic aggregates. This task is generally called *metering*. It serves different purposes: First, a certain quantity of resources has been allocated for every traffic aggregate, either a priori (manually by a system administrator or dynamically by some reservation protocol) or upon packet arrival (usually by some scheduling algorithm). It is mandatory, in order to preserve the integrity of the system as a whole and to isolate the different traffic classes, that such resources will not be exceeded. For this purpose, resource usage is kept track of, in the short as well as in the long term. In the short term, parameters like peak bandwidth utilization and burst duration are evaluated, while in the long term, average bandwidth utilization is the usual parameter under scrutiny.

A second objective of metering is the evaluation of the overall resource usage in order to perform admission control. The granting of resources to a new traffic aggregate is certainly a long time activity. It is of critical importance, since once admitted into the network, excess demand can only be fought through queuing delay and packet loss, which is usually undesirable.

A third aspect regarding metering is the collection of data relevant to the accounting and billing process, which happens in the *Statistics Unit*, s. Fig. 2.3. Although the collection of these data takes place in this module, its summarization and eventual interpretation is usually performed by the Management Unit and/or a centralized accounting infrastructure.

As stated above, metering implies the comparison of actual usage with expected usage. The *Rules Table* contains the information regarding expected usage by the different traffic aggregates as well as the directives indicating how excess resource usage should be treated. These rules table transpose the traffic policy and control the working of all other submodules within the Meter & Marker. Should a deviation from the expected behavior be detected, a number of actions are possible (in Fig. 2.3 summarized as *Packet Update*). The most obvious one is packet discarding, but packet loss is usually the last resort. More frequently, a *remarking* takes place. Although the Packet Classifier implicitly marks traffic as belonging to a certain class prior to delivering it to the Meter & Marker, that decision can be revoked here according to some traffic degradation policy in response to breaking the resource usage agreement. In principle, marking can be implicit, derived from the queue in which a packet is stored (as in the Integrated Services Architecture [BCS94]), or explicit. In this second case, marking can be network-wide, in which the packet itself receives a tag in its IP header (as is the case in the Differentiated Services Architecture [BBC+98], s. Section 3.3), or locally valid. Tags, that are only valid within the router are usually appended to the packet and use a proprietary format. However, they serve the same purpose as the network-wide marking: To define the service level that will be accorded to the packet. Once the packet leaves the Meter & Marker, its class is set for the rest of the processing inside the router.

Under certain circumstances, it would be advisable, that the Meter & Marker, upon detection of some undesirable condition, would update the rule dataset in the Packet Classifier. In this way, future packets could be either directly dropped or correctly classified according to some new policy derived from its previous behavior. Such mechanisms are not uncommon in certain applications, like dynamic firewall configuration.

Furthermore, it also corresponds to the Meter & Marker to detect and trigger the creation of new traffic classes as needed. Flow detection techniques are used for this purpose [CAB99], [LM97].

As is also the case for all other modules, a Management Information Base (MIB) is included. It stores the information concerning the status of the module

itself. It is regularly polled by the Management Unit, usually under request of a central network management instance.

2.2.4 The Switching Fabric

As opposed to the Packet Classifier, the Meter & Marker and the Traffic Pattern Conditioner, of which several instances can be found on every router, there is only one Switching Fabric (also called Router Backplane or simply Switch) per router. This responds to the physical placement of the different logical modules in the overall architecture, as will be evaluated in Section 2.3. The Switch temporarily interconnects every input and every output interface with each other and coordinates the forwarding of packets among them. For that purpose, packets may be fragmented in smaller chunks, adequate to the switch's interface width. An additional sought-after effect is the switching of constant-sized chunks, usually called cells. The main advantage vis à vis variable-length packets resides in the complexity of the allocator, as will be described shortly.

A number of switch architectures are possible and there is a very rich literature on this topic [CLO01], [CGMP99], [McK95], [MA98], [TY97], since switches have found application in many communication network architectures and at various layers in the OSI model. In this work only the main trade-offs in switch design will be sketched.

Independent of its architecture, any interconnection device has to resolve the following design problems:

- Internal Link Blocking: Depending on the architecture, it is possible that contention occurs inside of the switch, i.e., that two chunks should simultaneously traverse the same spot inside the fabric. A switch with this characteristic is called blocking, otherwise nonblocking. This can be solved either by placing buffering capacity inside the switch, or by rearranging the cells to be transferred prior to letting them enter the switch, so that no contention will arise. In that case, buffering occurs at the fabric's entrance.

- Output Port Contention: Switches are multiport devices. When several inputs try to send cells to the same output, contention occurs in the access to the outgoing link. This problem will always arise, independently of the switch architecture. The only solution is some form of buffering.

- Head of the Line (HOL) Blocking: Due to the internal situation in the switch, it can occur, that the first cell in the input queue can not be transmitted in the present switching cycle (assuming that an input port disposes of some FIFO buffering capability). In that case, subsequent cells in the queue, which are destined to other outputs and therefore might have been transmitted during the present switching cycle, are unnecessarily delayed. This situation can be prevented either by the use of non-FIFO queues,

which greatly increases the buffer management complexity, or by the implementation of Virtual Output Queuing (VOQ, to be explained shortly).

- Multicast support: There are several situations in networking, in which it would result advantageous to send the same information to several destinations at once[3]. Switches are best placed to perform that replication, since they interconnect every input with every output. Nevertheless, multicast support considerably increases the complexity of the switch control logic.

The long tradition in switch design has produced a variety of ways to cope with these design problems. Following [CLO01], switches can be classified according to their multiplexing strategy:

Time Division: Characterized by having one single path to interconnect all input and output ports. That one path is then multiplexed in time among them. They can be further subdivided in **shared memory** and **shared medium**.

The first type, characterized by a *common memory* in which all input ports write their packets and from which all output ports read, presents an excellent multiplexing gain in memory utilization. On the downside, memory access speed presents the limiting factor in overall switch performance. Due to this fact, although they were popular in the 80s and 90s, shared memory switches are not used nowadays in high-performance routers.

The second type, typically implemented as *buses or rings*, presents an evident solution for multicast support. Nevertheless, the bus (which is the most common form of shared medium switch) should present a capacity equal to the sum of all inputs, in order not to become a bottleneck. The scalability of bus systems is limited by the arbitration overhead in access to it and the bus capacitance, which limits the maximum frequency. Thus, buses are still the most common fabric in low-end routers due to their simplicity and low cost, but not in high-end products.

Space Division: These switches present several paths between inputs and outputs. This parallelism allows the transmission of several cells simultaneously, thus highly increasing the maximum throughput of the switch. They are, therefore, the architecture of choice for modern high-speed routers. Two variants exist: **Single-path and multiple-path** space division switches. Single-path fabrics present only one path between any input-output pair. The second class disposes of several paths between any two interfaces.

The most common representative of the single-path family is the *crossbar*. It is basically an array of crosspoints, that are connected or left open according to the desired connectivity. It is a very common choice, for its simplicity, good

[3]Here, multicast refers to the necessity of replicating packets inside of a switch, no matter for what purpose. Layer 2 switches, for example, routinely perform this task when forwarding frames to an unknown destination, independently of the higher layer packet being unicast or multicast. ARP is another example of layer 2 multicasting. In this case, ARP requests are explicitly broadcast to every possible recipient, and are also replicated inside of the switch.

scalability, modularity and the fact of being intrinsically nonblocking. It presents two main drawbacks, however: First, it scales with the square of the number of ports, thus setting an upper limit on them, albeit high. Second, an allocation algorithm is needed, which decides on the most efficient crossbar configuration, according to the cells presented by the inputs and the state of the outputs. This is a complex task, especially at high speed. Nevertheless, centralized as well as distributed allocators exist for multigigabit routers.

Banyan networks represent a second subcategory of single-path switches. Although self-routing and scalable in terms of complexity (even better than crossbars), they present the clear drawback of being blocking switches, their performance decreasing quickly with size.

The multiple-path subcategory presents a variation of the Banyan network, the *augmented Banyan* fabric. With the help of more stages in the fabric, multiple paths between any two interfaces are introduced. They share nevertheless the blocking drawback of all Banyan networks and add additional delay and routing complexity.

3-stage Clos networks, on the other hand, are strictly nonblocking and have a moderate complexity, which implies a good scaling property with the number of ports. However, the nonblocking characteristic is achieved at the price of rearranging the cells prior to transferring them through the fabric, which is a complex, time-consuming task.

The two last exponents of this family can also be seen as add-ons to most other designs. The *multiplane switch* simply replicates any previously cited structure and sets them in parallel, so as to increase the overall throughput of the system.

The *recirculation switch* reinserts cells blocked by the output contention problem into the fabric input by using dedicated interfaces for this purpose. This certainly reduces the cell loss ratio in the fabric. But besides reducing the number of true I/O ports, it presents the risk of out-of-order delivery of cells belonging to the same packet stream or even to the same packet, if fragmentation was used inside the router. This substantially increases the complexity of the control logic.

Summarizing, the two most commonly used switch structures nowadays are the bus, for low-end products, and the crossbar for high-throughput switches, often in a multiplane structure. The first approach is simple and cheap, while the second is simple, sufficiently scalable with the number of ports, nonblocking and presents a very high aggregate throughput. However, a new kind of switch is emerging: The optical switch [KCY+03]. Conceptually, it can be designed according to any of the schemes described above. Its main difference is the use of optical instead of electrical paths inside of the switch. Its main advantages are reduced power dissipation and consumption, enhanced scalability and the possibility to remove the need for an allocator.

Still, even nonblocking switches have to introduce some sort of buffering to deal with the output port contention problem or use recirculation, which, as has been seen, presents important drawbacks. The first approach consists in placing

the buffers at the output interfaces. In order to solve the contention problem, the switching fabric has to be able to forward up to N cells to the same output within one clock cycle, and the output buffer has to be able to store them in the same amount of time. This is referred to as a *speedup* of N, meaning that fabric and buffers have to run N times faster than the input ports. For high-speed routers, achieving a speedup of N is technologically unfeasible, memory access times for the buffers being the final limiting factor.

The alternative consists in placing buffer capacity at the inputs and adding an allocator instance, that chooses the optimal constellation of input cells to be transferred in a switching cycle, in order to maximize the throughput [McK95], [CGMP99]. As stated above, high-speed, high-density[4] routers need very fast and very complex allocation strategies. In order to achieve sufficient convergence speeds, heuristics are used, that try to come up with a suboptimal and yet acceptable cell constellation within the time bound. As line speeds increase, allocators become a bottleneck in switch throughput. Besides, variable length packets present an additional challenge: The time necessary to switch them is not constant, thus obligating the allocator to keep track of paths occupied by unfinished packets in every new computation cycle. Since this highly increases the allocator complexity, packet fragmentation in fix-sized cells is usually employed.

Input-buffered switches present an additional drawback: The HOL blocking problem. In order to solve it, VOQ was introduced [McK95]. Basically, the input buffers are divided in as many FIFO queues as output ports. Cells destined to a certain outgoing interface are stored in the corresponding queue. In every arbitration cycle, every queue from every input port can present a request to the allocator. Although this increases the number of requests to be computed, the switch utilization vastly increases. It has been proved [CGMP99], that input-queued switches with VOQ and a moderate speedup of two can reach 100% utilization, thus emulating output-queued switches with a speedup of N. Nevertheless, in order to perform traffic scheduling, as will be analyzed in the next section, buffering is also needed at the output interfaces. Consequently, most modern routers present a mixed form of input-output buffers in their switch design.

Finally, multicast support is guaranteed in most switches by introducing separate queues at the inputs for multicast traffic, so that it will not interfere with unicast packets. Basically, a multicast packet can be forwarded across a switch either by sending it to all its destinations at the same time, or by splitting the transfer among a number of cycles. This last technique is known as *fanout splitting*. It has been shown to vastly reduce the mean delay time for multicast cells [TY97].

[4]Routers with a large number of ports are usually referred as "high-density" or "highly populated" routers.

As has been seen, buffering takes place in different places in the router. Independently of its situation, different strategies can be employed to set its size and manage space usage within it. This is referred to as *buffer management* [CF98], [FJ93], [GIGK95], [RLG98], [SLSC98]. Its global goal is the improvement of network performance, which for best-effort routers, translates into minimizing buffer overflow and packet loss. Additionally, buffer management policies also have an impact in the delay and jitter suffered by packets. For routers supporting traffic differentiation and QoS, therefore, buffer management has a triple impact on experienced end-to-end quality through its effect on delay, jitter and loss. Besides, traffic differentiation implies, that the goal of minimizing packet loss and maximize network performance no longer holds: Buffer management must presently ensure, that network utilization will be maximized *while respecting the traffic contracts of the individual packet aggregates.*

2.2.5 The Traffic Pattern Conditioner

When packets arrive at the output interface, they have to be serialized in order to be sent on the network link. Since modern routers perform traffic differentiation, packets belonging to different aggregates will be sent with different priority and frequency on the network. That is the task of the network scheduler. Additionally, the router also adapts the traffic characteristics of each aggregate, so that network utilization will be maximized and congestion avoided. This process is known as shaping. Both tasks imply, that some packets, although arrived at the output interface, will be held back while others with a greater priority[5] are sent [Zha95]. For this purpose, they are buffered in queues prior to being serviced. Since the granularity of differentiation can be very fine, the number of output queues to be supported in a router can also be very large. That does not present a technological problem nowadays. Besides, in order to keep the network management tasks relatively simple, a tendency can be observed toward policing large aggregates instead of flows, which means that in most cases few queues are needed.

As was explained in the Switch section, entrance to the switching fabric (especially in input-queued or input-output-queued backplanes) already responds to the traffic differentiation policy set by the administrator. A first scheduling and shaping process is thus performed per input. Nevertheless, a second such process is needed at the output links: The *total* amount of traffic belonging to a certain aggregate and going out at a certain link is the sum of the packets supplied by all inputs. That is only known at the output interface. Hence, from an overall system perspective, link sharing according to the traffic policy can only be realized per output in the *Traffic Pattern Conditioner.*

[5]By priority it is not meant, that a static preference is accorded to a packet or traffic aggregate. It can well be, that a packet increases its priority during its queuing time, according to other parameters, like e.g. the accumulated queuing delay.

A number of criteria can be used to judge the goodness of a scheduling and shaping process (also known as service discipline) [Zha95]:

- Efficiency: An important goal of any service discipline from the network administrator's point of view is the achievement of a high network utilization.
- Protection: It is essential that the performance requirements of well-behaved sources be kept, even in the presence of misbehaving users, network load fluctuation and/or unconstrained best-effort traffic. Isolation among aggregates must be thus guaranteed.
- Flexibility: Diverse traffic characteristics and performance requirements must be equally well supported by the service discipline.
- Simplicity: In principle, complex algorithms might be in a better position to reach the above mentioned objectives. However, the scheduling process must be performed for every packet at line-speed, and thus must be amendable to efficient implementation. Hardware-support is also desirable. Simple although suboptimal algorithms are thus preferred.

A rich and sophisticated set of algorithms has been developed, that try to approximate the mathematically optimum result [PG93], [PG94], while reuniting the characteristics mentioned above [Zha95]. Two main sorts of algorithms can be distinguished: work-conserving and non work-conserving. The first sort always sends a packet to the network, if at least one queue is occupied and the link is idle. The second class, non work-conserving, might leave a queue unserved in the presence of an idle link, if the packets therein are deemed ineligible to be sent. This certainly has an impact on average packet delay and jitter, as well as the average system throughput. Nevertheless, non work-conserving strategies have shown to be advantageous in setting lower bounds on overall network delay. Which quality is preferred depends in part on the traffic characteristics to be supported.

In this context, a much discussed strategy foresees the *hierarchical* sharing of link bandwidth among parties [FJ95]. In this kind of strategy, every party receives a guaranteed lower-bound on link capacity. That bandwidth is further subdivided according to the traffic classes defined for that party. If necessary, further recursion is possible. One of the advantages of this approach is, that *different* service disciplines can be used for different traffic classes and/or users. Thus, the bandwidth-delay requirements of every user should be better served. Nevertheless, the interaction among different algorithms and the management complexity of such a hierarchical structure are not negligible.

Summarizing, no universal solution has been devised for the traffic conditioning problem. Several algorithms are used, depending on the kind of traffic to be served, the system characteristics and the expected user preferences.

2.2.6 The Routing Unit

The main function of the routing unit is the compilation of the *routing tables* to be used by the router. A routing table contains information on how to reach any destination known by the router, as well as additional information regarding that route: The distance (measured in hops, seconds or other metrics), the Autonomous Systems traversed, a bandwidth estimation, etc. Destinations are usually represented by a network address and its respective network mask. The number of address bits relevant for the route estimation is referred to as a *prefix*. Additionally, a routing table contains the identity of the output port to be used to reach every destination.

A routing table contains an indication of the optimum routes *only*. I.e., in the case, that several paths are possible to reach the same destination, the routing table has to evaluate them according to a certain metric and choose the one deemed best. The most common metric is the distance to the target, measured as the number of routers to be traversed underway. The information concerning all alternative routes is stored in a *routing database*. Should one of the optimum routes not be usable for any reason, the database will be used to compute a new path.

The number of routing tables in a router is not limited to one. If different metrics are considered relevant for different uses (e.g., for special traffic classes, like delay or jitter for real-time streams) or for different destinations (e.g. all packets addressed to the IBM corporation shall traverse partner networks only), several routing tables can be computed. It corresponds to the system manager to configure the Packet Classifier in such a way, that the right table will be used in every case.

Two elements are needed to compute a routing table: Information concerning destinations and how to reach them, and an algorithm to calculate what is considered optimum under some point of view, like e.g. the shortest path to the target. Both elements are provided by the *routing protocols*. On the one hand, a routing protocol controls the exchange of routing information among routers. Every router sends its information to all of its neighbors, which in turn add their own knowledge, summarize the total information and pass it on to their neighbors. On the other hand, every routing protocol implements a *routing algorithm*, in charge of calculating the best routes according to the metric chosen. The two algorithms used in commercial routers are the Bellman-Ford algorithm and the Dijkstra algorithm [Com00]. Nevertheless, other algorithms exist. Of special interest are the attempts to take more than one metric into account (so-called *constraint-based routing* [CNRS98]). However, it is known, that a multidimensional optimization problem is a mathematically hard problem. Although many approximations and heuristics have been tried, the inherent complexity of the algorithm plus the management complexity of using it in today's networks have relegated them to mere academic exercises.

Routing protocols, thus, generate the information used for building the routing tables. For the Packet Classifier to take a forwarding decision, though, the only relevant information is the prefix to search for and the corresponding outgoing interface. Furthermore, according to the specific lookup (search) algorithm employed, that information must be organized in a convenient way (a certain sort of tree, say, or a bit vector structure). Hence, the Routing Unit must synthesize the content of the routing tables into so-called *forwarding tables*, which are then downloaded into the Packet Classifiers. The table update is logically also a Routing Unit task, to be performed asynchronously, every time a change in the routing table occurs.

Routing protocols are software units, which typically run on a general purpose processor. The Routing Unit is thus basically a general purpose processor with specialized software.

2.2.7 The Management Unit

The last module present in Fig. 2.3 is the *Management Unit*. It veils for the smooth operation of all other modules, as well as the communication of router status information to a centralized network management infrastructure. Typically, the events that are of interest for the management infrastructure are those related to performance monitoring, accounting, configuration control and fault detection, isolation and correction. To that end, the Node Management Architecture, realized through the *Node Configuration Unit*, collects and maintains information about those events, provides them to the network management and responds to manager commands regarding changes in the router configuration. The communication with the centralized management infrastructure, as well as with other networking elements for the purpose of fault management (ICMP, e.g.) is realized by a number of signaling protocols. The most widespread protocol is SNMP [Sta98a], [Sta98b].

Additionally, the Management Unit is in charge of processing those packets, that, although deemed correct by the Packet Classifier, can not be associated to any existing packet aggregate (including best-effort) or require some form of special treatment. Examples of those packets are management datagrams themselves, sent by a signaling protocol, or reservation requests sent by means of RSVP, packets belonging to unknown protocols, etc. The Management Unit typically is realized in a general purpose processor, in which a richer variety of protocols can be supported than in other highly specific router modules. In exchange, the time taken to process such packets is orders of magnitude larger than in the default (hardware-supported) data processing path. That is why this second way is usually referred as the "slow path" of the router. Since management tasks are usually not time-critical and seldom events, performance is not a problem, as long as few data packets take the slow path. A possible network

attack consists in overflowing the Management Unit with formally correct yet bogus packets, that have to be processed by this module.

The modules described here correspond to a logical grouping of the multiple tasks, that any modern router has to cope with. And still, as has been seen, there are a number of design possibilities and trade-offs in their physical realization. The next section will describe how those trade-offs have been resolved through the last 25 years.

2.3 Evolution of Router Architectures

In the previous section, the main architectural trade-offs in router design have been exposed. At present, the evolution in router architectures will be explained through its main design decisions. Finally, the present design choices will be highlighted and their shortcomings regarding functional flexibility will be introduced.

2.3.1 First Generation: Single Processor, Shared Bus

Basically, commercial routers are composed of only three modules: A number of interface cards[6], containing the ports that connect the router to different transport links, some form of backplane to interconnect them and a controller unit to perform the software tasks (mainly routing and management). Nevertheless, which functions have been taken up by which unit has greatly changed through time [KLS98], [Awe01], [Par98].

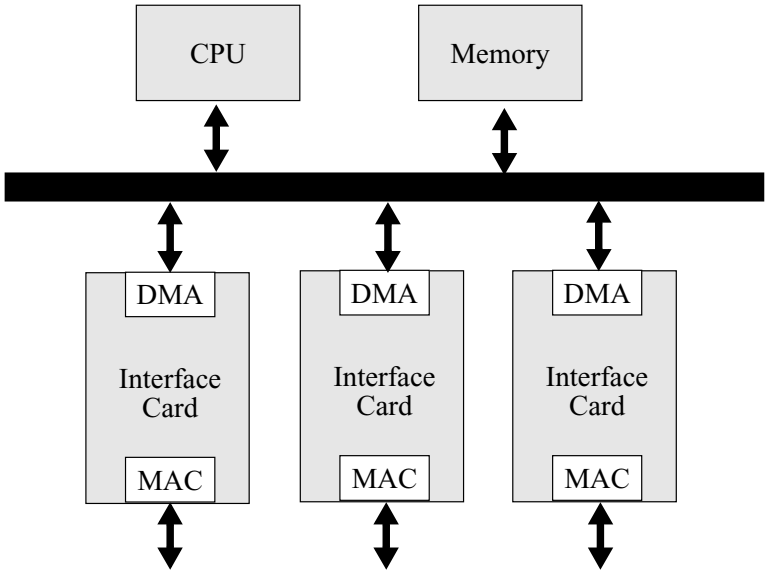


Figure 2.4: 1^{st} Generation Router: Single Processor, Shared Bus

[6]Interface cards are also known as "line cards", "network cards" or even "ports".

Since routers are expensive equipment and simultaneously critical for data transport, they have to consistently provide reliability and performance under extreme working conditions. One paramount goal of router design, hence, is to provide constant high throughput independently of traffic pattern arrival.

Up until the last eighties/early nineties, routers where built around a central CPU, which performed all the tasks (s. Fig. 2.4). The router was eminently a software instance, where a number of extremely simple interface cards forwarded all packets to the CPU for processing. Additionally, the CPU realized all the routing and management functions described in the previous section. While the routing decision was taking place, packets were usually stored in a centralized memory, controlled by the CPU. Hence, any packet coming into the router had to cross the internal bus twice, one to be transported to the CPU and a second time to be sent to its destination output card. With increasing network capacity, the scalability problems of this first router generation are centered on: The internal bus, which, as has been seen in section 2.2, has limited throughput scalability and the CPU. With increasing traffic and thus processing burden, the need arose to distribute the processing among several instances and away from a single software unit.

2.3.2 Second Generation: Cache-supported Interfaces, Shared Bus

Around the mid 90s, a new router architecture had been established. Its main goal was to increase the processing power inside of the switch, while reducing the effect of a slow internal bus. To that end, two main innovations were introduced: Caches and multiple packet processors (generally referred to as "forwarding engines"), s. Fig. 2.5.

The most common innovation was the introduction of some processing power on every interface, mainly to operate a local cache. Furthermore, several ports were packed on every card. The idea was to keep a cache of recently seen addresses and the respective outgoing interfaces, so that packets would not have to be sent to the main CPU for processing. Additionally, by supporting multiple ports per card, a fraction of traffic could be routed without leaving the line card. The main insight behind these decision was traffic locality: The fact that, due to the packet train effect [PCB+98], there was a high probability of seeing packets addressed to the same destination within a short period of time. Assuming that a cache hit was considerably less costly that a full route lookup, routing decisions could be accelerated. This solution worked well for some time. It could not hold forever, though, for the following reasons: First, caches have to be built and updated. The cost of thrashing the cache was very high. With increasingly frequent changes in the routing tables due to an ever expanding, highly dynamic Internet, cache update became costly. Furthermore, caches make only sense if the hit rate is sufficiently high and if its cost is notably less than a full lookup. As has been expressed before, advances in lookup algorithm design reduced the

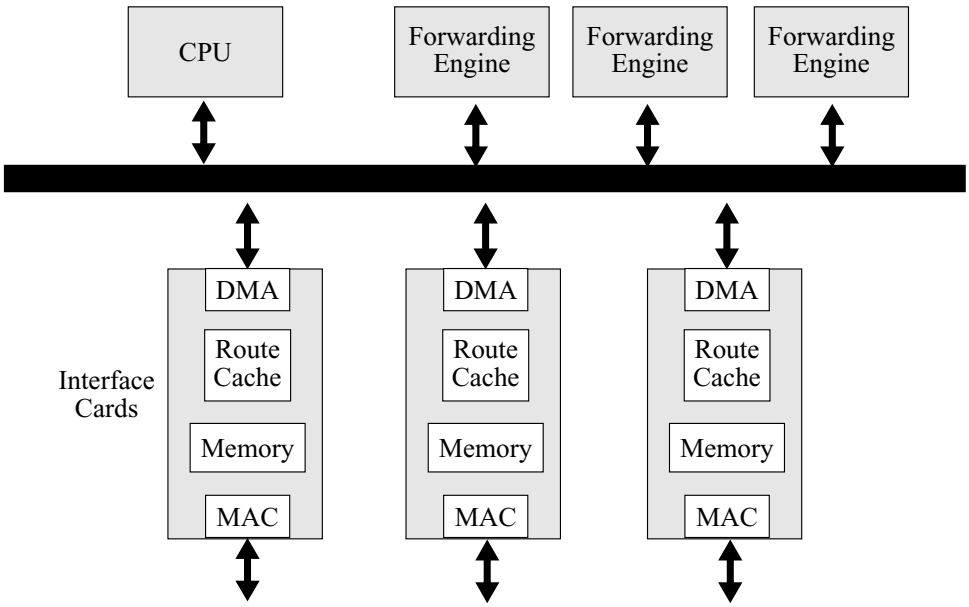


Figure 2.5: 2^{nd} Generation Router: Cache-supported Interfaces, Shared Bus

appeal of caches. Besides, higher network capacities also meant, that the number of active flows per router increased, thus reducing the cache hit probability [TMW97]. In any case, cache efficiency is traffic pattern dependent, which is a severe disadvantage. Hence, although caches are still useful, the exploitation of traffic locality alone to achieve higher throughput does not suffice.

A parallel evolution was the replication of processing capacity: The introduction of pools of forwarding engines. These modules would realize mainly full route lookups and other tasks (like sanity checks, CRC update or TTL decrement). Packet headers would then be sent over the bus to one of these engines, which would answer the requesting card with the desired destination interface. The packet, which had been buffered in the card, would then be sent directly to the output interface, thus reducing the burden on the bus. Nevertheless, with increasing throughput demands, the bus remained a bottleneck.

2.3.3 Third Generation: Fully Distributed Processing, Switching Fabric

In the last ten years, a new router design has come to dominate the high-end product offering. The two original bottlenecks (internal bus and centralized processing) were eliminated through the introduction of switching fabrics and fully distributed packet processing. On top of that, support for a bunch of new tasks (QoS, firewalling and the like) was added.

The scalability limits of the internal bus were by then evident. There was a necessity to introduce parallel forwarding capabilities between any two input-output interfaces. With the success of ATM switch design, the solution was at

hand: Most modern routers use switching techniques derived from those employed in ATM. The remaining bottleneck, thus, was packet processing.

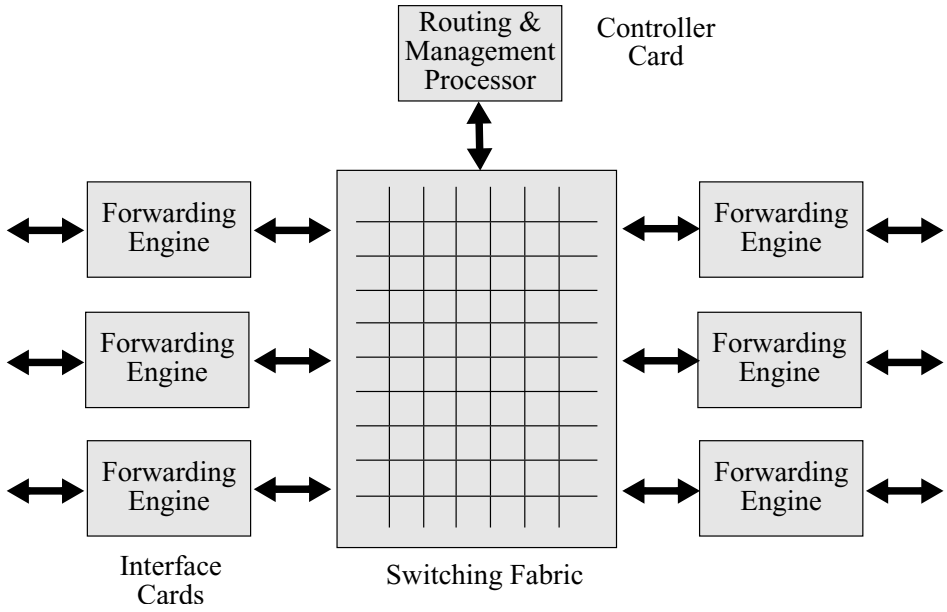


Figure 2.6: 3^{rd} Generation Router: Fully Distributed Parallel Processors, Switching Fabric

The solution employed was the introduction of packet processing on a per-card basis. Full forwarding engines, performing all tasks of *data* packet processing, were introduced on every multiport card. Caches lost part of their appeal in front of massively parallelized engines, with specialized modules to perform the different router functions: lookups, sanity checks, scheduling, etc. These tasks, divided into very small and repetitive operations, were highly amendable to hardware support, thus increasing performance yet again. Nowadays, most data plane operations are realized with hardware support. The controller card remains responsible for the generation of routing tables and for the management tasks. Every engine receives a full forwarding table, and updates are usually realized via hot-swapping memory banks. New modules have been introduced in the cards to take care of emerging operations like packet filtering, access control, scheduling, encryption, etc. The architecture of choice at present is depicted in Fig. 2.6 and can be found in such commercial products as the Cisco 12000 [Cisb], [Cisc], [Cisd] or Juniper's T640 [Juna], [Junb].

In spite of all evolutions, routers remain essentially closed, highly specialized systems. This provides reliability and high performance, but at the cost of flexibility. The last 5-10 years have witnessed a tremendous expansion in routing functionality, which has obligated router vendors to constant architecture redesigns and network operators to costly hardware and software updates. In times of crisis, such investments are highly unwelcome. A search for increased functional flexibility and adaptability started, which was pioneered by the *Ac-*

tive and Programmable Network research community. But before exploring their proposals for increased flexibility, a short revision of the router types and their special requirements should be offered.

2.4 Router Types

Not every commercial router supports the broad set of mechanisms described in Section 2.1. Depending on the application scenario, the requirements and boundary conditions change. In this section, the three main scenarios in which routers are employed will be described, as well as the resulting typical equipment configurations.

2.4.1 Core Routers

Core routers are also referred to as "backbone" or "high-end" routers. They are placed inside of the main backbone networks. They accordingly interconnect links at the highest aggregation levels, typically several tens to hundreds of Gbps. In this scenario, routers have a reduced set of high-capacity ports and typically support only one technology: Either IP over ATM, or IP over Frame Relay or (increasingly) Packet over SDH/WDM. Since they transport the bulk of internetworking traffic, twin paramount requirements are reliability and throughput. The first is achieved by means of dual power sources, module replication, protection, etc. The second is achieved by reducing the functionality to its minimum expression: Typically, backbone routers support a reduced set of routing and management protocols and perform no additional task besides simple packet forwarding (i.e., they do not perform packet filtering, firewalling or encryption). Nevertheless, they do have to support traffic differentiation, albeit in general in the form of a few large aggregates. Because of their strict requirements and large customer base served, price plays only a relative role in their purchase. It is in this scenario where full-fledged third generation routers can be found.

2.4.2 Edge Routers

Edge (or "Enterprise") Routers serve in two main scenarios: Either as interfaces between large backbones and smaller ISPs, or as interface between ISPs and large corporations. They can also be found performing a dual role as core/edge routers in medium sized corporations, that prefer a unified solution to a number of specialized appliances. Depending on the concrete environment, edge routers must provide a large number of ports to interconnect branch offices, smaller ISPs or even SMEs. Furthermore, a broad spectrum of (legacy) technologies must be supported, as well as different granularities for every technology. Because of this environmental heterogeneity, also a large number of protocols and services must be supported. On top of that, QoS and traffic differentiation usually depend on edge router processing prior to accepting traffic into the backbone. Flexibility

and a very broad service palette are thus the most critical requirements for enterprise routers. Nevertheless, price plays a big role in this market segment, since the customer base is much smaller than in core routers. Since the number of ports is large, an interface card cost as low as possible is mandatory.

The usual form of such routers floats between the second and third generations explained before. Performance is indeed a requirement, but falls at least an order of magnitude below backbone routers. Their distinctive characteristic is a large number of additional and specialized software and hardware modules, to implement the set of value-added services required in this environment: VPN, encryption, firewalling, web- and mail-filters, proxying, NAT, etc. Processing power is thus more important than forwarding capacity.

2.4.3 Access Routers

This "low-end" routers interconnect final users with their ISPs. Traditionally, they were little more than modem concentrators, directing traffic to an edge router. However, the sharp increase in broadband access technologies has forced more forwarding power and functionality into this segment. The hosting of value-added services into these routers has obligated to upgrade their capabilities. Nevertheless, their main requirement is still to remain cheap. It can be envisioned a convergence in functionality toward the edge router, especially if broadband access becomes commonplace. At the moment, though, they are still the last reduct of the first generation router, where a pure centralized software solution takes the whole processing burden.

SOHO routers are the last step in the router hierarchy. They are the counterpart of the access router in the end-user's home. Although they are taking up an increasing functionality (firewalling, packet filtering and VPNs, especially), they remain very simple devices: A few Fast Ethernet interfaces for internal use, an ISDN or DSL uplink and a software processor.

2.5 Router Technologies

To realize the "intelligence" or the processing elements in routers, four different technologies can be found as main building blocks: Application Specific Integrated Circuits (ASICs), General Purpose Processors (GPPs), Network Processors (NPs) and Field Programmable Gate Arrays (FPGAs). They represent different choices in the trade-off between performance, flexibility and cost. Accordingly, these technologies find their use in the different scenarios described in section 2.4: The access, edge and core routers. In this section, the nature of the trade-off mentioned above will be explored.

2.5.1 ASIC

ASICs and GPPs represent the two extremes in the performance vs. flexibility trade-off (s. Fig. 2.7). ASICs are the result of a careful design in order to obtain the fastest possible circuit to realize a certain well-defined function. To that end, the classical advantages of hardware systems are exploited, mainly parallelism and pipelining. Such application-specific circuits are consequently several orders of magnitude faster than the equivalent realization in software. Because of their hardware nature, however, these circuits are absolutely inflexible: No functional adaptation is possible. Accordingly, these circuits are not suitable for the realization of non-mature functions (i.e., those which are still in the standardization process). It also makes updates extremely expensive, since the chip has to be substituted. Such upgrades follow the natural evolution of service development, though, since every new release usually brings modifications and extensions to the original functionality. Furthermore, correcting bugs is also an impossible mission in ASIC design, once the fabrication process has started.

In general, hardware systems are well suited for the realization of repetitive, bitwise, processing-intensive tasks with limited variability. This derives from the intrinsic characteristics of hardware designs: Every new branch in a task has to be realized as an additional circuit, no matter how often it is called. Thus, high variability implies high resource usage and high cost. On the other hand, repetitive bitwise operations allow for a high degree of parallelism and pipelining with limited resources, presenting the optimum use of hardware systems. Accordingly, ASICs are not well suited for control plane functions.

The design of an ASIC mask is a long process, which is a drawback in times of ever increasing competition and pressure to bring new products to the market. The consequences of buggy or otherwise non-compliant designs furthermore increase the number and complexity of tests to be performed prior to fabrication. As a consequence, ASICs are a technology only adequate for mature, established functions, which can be used in hundreds of thousands or even millions of identical chips, so that the investment will pay off. ASICs can be found basically in the data plane of core routers, where performance is the most critical parameter.

2.5.2 General Purpose Processors

At the other extreme of the scale GPPs can be found. They present the standard in programmability and, even more importantly (as will be seen in the discussion about NPs and FPGAs), GPPs are easy to program. Furthermore, a large community of professionals exist, which master the software skills used in their programming. Accordingly, the development of software-based networking solutions is, relatively speaking, an unproblematic and cheap task. Nevertheless, GPPs present a clear bottleneck in terms of performance: In principle, all tasks related with packet processing would have to be realized by the GPP. They contain usually only one or two processors, which presents a limited field for par-

allelism even in the presence of multi-threading techniques. Context-switching, to configure the CPU with the necessary state for every application is a heavy overhead, as well as the successive software layers on top of the hardware, which add abstraction (and thus easiness of programming) as well as delay. Access to peripheral elements, like interfaces or memory, also add latency to data processing. As a consequence, GPPs are usually only employed to realize the control plane of modern nodes, since the control plane presents the most variability and complexity. Only access routers, where the price is the discriminating factor, use software solutions in the data plane.

2.5.3 Network Processors & FPGA

It is precisely between the two extreme positions represented by ASICs and GPPs, that the most interesting developments in router technology can be observed nowadays. Especially for the edge domain, although increasingly also in the access (as price decreases and performance requirement increases) and the core (as the performance gap with ASICs decrease and the need for flexibility increases), two competing technologies are found: NPs and FPGAs. As represented in Fig. 2.7, they combine a high degree of programmability with increasingly high performance, thus simplifying the trade-off exposed at the beginning of this section.

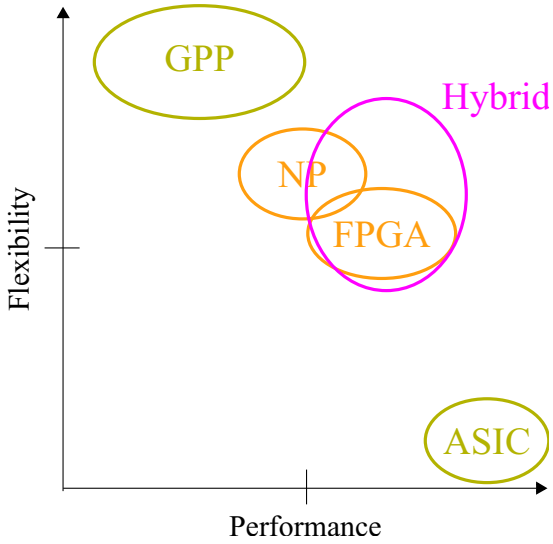


Figure 2.7: Performance vs. Flexibility in Router Technologies.

Network Processors are typically multi-processor systems [Eng03], [PC03], [VCY03]. They include a GPP, for the realization of complex, non-performance critical tasks, and a number of small processing elements, alternatively known as micro engines, pico processors, etc. These micro engines are in principle small versions of a GPP, but with a networking-adapted microinstruction code and

without intermediate software layers (generally not even an Operating System). The main rationale is the distribution of the data plane packet processing tasks among the micro engines, either in a parallel or pipelined structure. The first variation allows for the simultaneous processing of several packets, where every micro engine performs all tasks. The second allows for the distribution of the processing in several steps, so that every processor is specialized in one of them. Also in this case several packets can be simultaneously processed.

As can be seen, the main motivation for network processors is to achieve the advantages of classical hardware systems without sacrificing flexibility. Nevertheless, in order to achieve high throughput, a detailed knowledge of the exact NP architecture is mandatory: The programmer usually has to distribute the original application in a number of parallelizable operations, which is a complex task. Furthermore, those operations have to be described in the most efficient way for the exact micro engine architecture, which usually implies writing it directly in assembler. Additionally, the application has to be distributed along the processor hierarchy (GPP, micro engines), which brings along a communication overhead and additional complexity. Although NPs are still software systems, they are difficult and burdensome to program and need special skills [Her01]. Scheduling tasks among processors, so that they will all be doing useful processing most of the time is a non-trivial task [WPF03].

A limiting factor in NP performance is the access to peripheral elements, as in GPPs. Among them, memory is the most critical [McK04], [ABC$^+$03]. In order to hide the latency incurred in accessing off-chip memory (where typically packets and state will be stored), multi-threading is used in the micro engines: While a thread is blocked waiting for data to be fetched, another thread makes use of the processor cycles. This goes a long way in reducing the effect of latency, but it still represents a limiting factor, since networking is a very data-intensive domain.

In spite of its multi-processor architecture, NPs are still basically general purpose software systems. Processing-intensive tasks, like encryption, are still better realized in hardware, due to their inherent parallelism and heavy bit-shuffling characteristics. To that end, most NPs include dedicated hardware-based co-processors that perform these kinds of task: Address lookup, encryption, pattern-matching and related functions are common occurrences. This co-processors are implemented in ASICs or FPGAs.

FPGA-based systems present the complementary approach to NPs. FPGAs are composed of an array of logic blocks placed in an infrastructure of interconnections [Sik01], [Wan98]. Both the logic blocks and the interconnections can be configured. The first represent a basic logic function (i.e., "$A \bullet \overline{B} + C$"), while the interconnections transport the results of a logic block to another one. Consequently, an FPGA, which includes thousands of such logic blocks, can implement very complex logical functions. Additionally, modern FPGAs include complementary elements on-chip: memory blocks and interfaces are the most

common ones [Altc]. An increasingly frequent occurrence is the introduction of so-called "microprocessor cores" [Altb]: Synthesized functions, which provide the functionality of small processors, exactly as micro engines do. Alternatively to these "soft cores", "hard cores" are GPPs pressed on the same FPGA chip, with integrated interfaces to it [Alta], [Xil]. This reduces greatly the communication overhead between the GPP and the logic, in the same way that on-chip memory reduces the access latency. As a consequence, FPGAs provide an attractive alternative to NPs in exactly the same segment. The main difference between the two is the role given to pure hardware design: In NPs, software is still the tool of choice to program the functionality, even if it is in a hardware-close language like assembler. In FPGA-based systems, the logic is directly designed using hardware description languages like VHDL or Verilog, which are intrinsically adapted to realizing parallel processing and pipelining.

The drawbacks of FPGA-based systems are not unlike those of NPs: The parallelization and pipelining of complex tasks is a burdensome issue which implies a long design cycle. On top of that, the pool of programmers with VHDL or Verilog knowledge is much smaller than for C or even assembler. Even with the introduction of on-chip memory, most systems require external banks to be used as packet buffers. Bringing data in and out of an FPGA is as costly as in a NP. Still, when performance is a dominant issue, FPGA-based systems are the more powerful solution, while keeping absolute flexibility.

The edge segment is thus living a technological revolution. An increased push for more flexibility and functionality, while keeping up with performance has provoked the emergence of NPs and FPGA-based architectures. A clear trend can be seen to the inclusion of hybrid systems, that combine both elements to some degree. Many specialized products have emerged for different niche applications, which in some form try to merge the flexibility of software systems with the performance of hardware designs. These hybrid approaches will certainly claim an increasing market segment in the future.

3 Related Work

In this chapter, the previous work most related to the ideas presented in this Thesis will be reviewed. Essentially, three main fields are concerned: The area of Active & Programmable Networks, which lies at the heart of the Thesis' motivation, is the first one. Security in hardware programmable systems, which sets a number of boundary conditions and requirements to the design presented in chapters 4, 5 and 6 is the second. Lastly, the fundamentals of QoS in the networking environment will be explored, as well as the most prominent proposals in the area.

3.1 Active & Programmable Networks

Since the field of Active & Programmable Networks has seen an explosion of approaches in recent years, it does not seem plausible to review them all here. An introduction to their main common characteristics and motivation, on the one hand, and a review of the projects most relevant for the present work, on the other, will be provided instead. Fairly comprehensive surveys of existing proposals, including interesting analysis and comparisons, can be found in [CMK+99], [Pso99], [TSS+97].

The rationale behind the Active & Programmable Network idea is the transformation of data networks from simple bit-transporting pipes to service-aware programmable platforms. In a sentence, Active Networks try to re-introduce "intelligence" into the network, so that value-added data processing does not exclusively happen on external equipment (typically end-user equipment or third-party servers). In this context, the first and paramount goal is to simplify the development, introduction and management of new services, that either are realized in the network, or at least get support from it [CMK+99], [TSS+97], [Pso99]. The underlying assumption is, that the network can realize certain services (or parts of them) more efficiently than nodes located outside of the network. Some examples are: Distributed, dynamic network management, Virtual Private Networks, data mining, media transcoding and web caching.

Even beyond considerations of which value-added services can take advantage of this new paradigm, it is a fact, that modern networks are assuming an ever increasing number of tasks. Coupled with this, the acceleration in technical evo-

lution implies, that innovation comes about in ever shorter cycles. Enterprises compete in terms of time-to-market and life cycles shorten. The unfortunate consequence is, that non-mature or non-stable product versions are brought to the market, which subsequently need frequent updates, bug fixes and / or substitution, once more stable specifications or standards have been agreed upon. Nowadays, introducing these functional changes in public networks is a long and burdensome task, which implies manual software and hardware substitution wherever an update is needed. The second main goal of Active Networking is thus to prepare the network to efficiently accept frequent partial functionality changes, without disturbing other services running on it.

The third objective of Active Networking concerns network management. The increase in functionality, that modern networks have suffered, has also brought about an increase in network management complexity. The impact of new services on the existing infrastructure, the so-called "feature interaction", is always a source of concern for network operators. The cause resides, at least partially, in the monolithic approach to management: Every operator runs its network more or less as a single entity, where many different services and technologies coexist. The management must, accordingly, control them all. Active Networking takes a different stance: It asks for open management of the individual services, so that every service provider, or even every user can configure the network according to his/her needs. This certainly foresees the isolation of the different services and users from each other, so that no interference among them shall occur. The side effect of such an approach would be, however, that the management burden for the network operator would be greatly reduced. Only the basic functionality would have to be directly managed by it, leaving the individual value-added services to third-parties[1].

From the point of view of the network operator, however, a number of concerns arise: The impact of such openness on the security and safety of the network is paramount. On top of that, a possible degradation in network performance derived from additional functionality and the new mechanisms needed to support such openness are an additional fear. It is against these concerns, that the Active Network proposals have to be evaluated.

The transformation from "passive" to "active" networks implies furthermore a different view of the network: It now represents an intelligent platform, which provides three main resources to services running on it: Bandwidth, processing power and storage. This implies abstracting the network resources at different levels, in order to make them usable: From the data link, at the lowest level, through the virtual router to be managed by a service operator, up to the virtual

[1]This would presumably transfer a large portion of the profits to the service provider, and might therefore make it an unattractive model for the network provider. But first, most network operators are also service providers. And second, the value of the network as a service-aware platform would increase, making it appealing to a number of other service providers and end users, hence increasing the network provider's customer base.

network composed of all resources involved in delivering a service to a number of users. Accordingly, the node resources have also to be abstracted, so that they can present a homogeneous interface to the different network programmers. Network Application Programming Interfaces (NAPI) are, hence, a critical requirement of Active Networking. This NAPIs can be seen as the equivalent of Software Development Kits (SDK) for traditional applications: They describe the functionality offered by the network and how to use it. The idea is to provide a basic set of NAPIs, on the basis of which more complex applications and whole virtual network architectures might be constructed (s. Fig. 3.1). Another basic requirement is the provision of mechanisms for safe resource partitioning among virtual networks. Otherwise, service interference and inconsistent network configuration would follow.

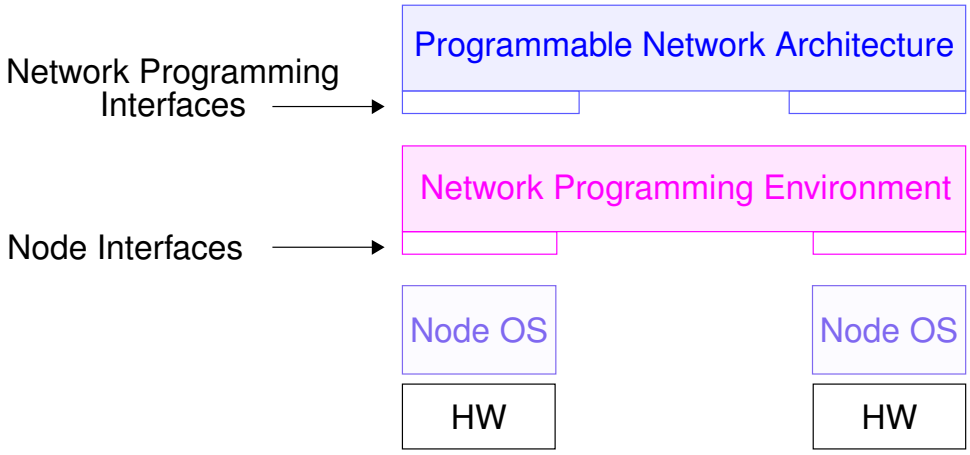


Figure 3.1: Generalized Model for Programmable Networks.

Another critical point in Active Networking, derived from the network abstraction discussion, is the differentiation between transport hardware and control software. In the eyes of this community, the role of hardware systems is a simple one: Accept instructions from the software, and move bits around accordingly. The whole "intelligence" resides in the control software, in which also more complex applications are realized. Classical AN approaches dismiss, thus, the role of hardware and typically let its characteristics be abstracted by the Operating System.

What most AN proposals share is then, an infrastructure for the uncoordinated deployment of services in a network, with support for the transport and installation of the code necessary for it, including safety and security architectures. The management of the services is left in the hands of the service operators and/or the users. A number of variants can be found in the literature, but most comply with this description [Cal99],[BS99], [CSZL01], [GPS+00], [KP01], [MRLC98], [CMK+99], [TSS+97], [Pso99]. Out of this broad characterization,

however, two main schools have emerged: Active Networks and Programmable Networks.

- **Active Networks**: Some proposals, like ANTS [WLG98] proposed a radical change of paradigm: Network nodes shall be seen as giant virtual machines, inside of which different programs would run. Every incoming active packet, now termed "capsule", would accordingly transport not only a set of data, but also the code with the functionality required to process it. The code would be compiled in the virtual machine and would operate on the data, prior to being sent out. Although this approach offered maximum dynamics and granularity of operation (network functionality was changing with every packet), it also presented strong problems in terms of performance and security. The reasons lie at hand: Since every packet contains code and thus a potential threat, security checks have to be applied on the data plane, to ensure its safety. Furthermore, the complexity of applications being transported by capsules was limited by the maximum size of a packet, thus preventing the realization of services beyond trivial operations.

- **Programmable Networks**: In this view, active packets are still supported by many proposals, but more complex applications are stored in repositories. These applications can reside locally on the node or be downloaded from an external repository, either on-demand (triggered by the arrival of a packet, that needs such functionality) or *a priori*, according to some scheduling algorithm. The main difference, besides supporting bigger applications, is the separation between "minor" and "major" functionality changes, especially in terms of their security implications. Bigger applications imply a bigger risk for the safety of the node, since they represent an important alteration of its functionality. Hence, more heavyweight security checks should be performed on them. This can happen before download or activation, thus, precluding performance degradation in the data plane. Smaller applications, that can be transported in active packets have a limited scope of action and they can be efficiently controlled by limiting their capabilities through Access Control. Only lightweight checks are necessary. In some approaches, even different programming languages were developed for both kinds of programs.

Most projects in the field of A&PN share a common node architecture, as defined in [Cal99]. The model, reproduced in Fig. 3.2, consists of four main elements: Active Applications, Execution Environments, a Node Operating System and a hardware platform.

The Active Applications (AAs) represent the actual services being implemented in the node. Typically, they are executed in controlled runtime environments, called Execution Environments (EEs). They are similar to the virtual machine concept: EEs control the access to common resources by the AAs, that

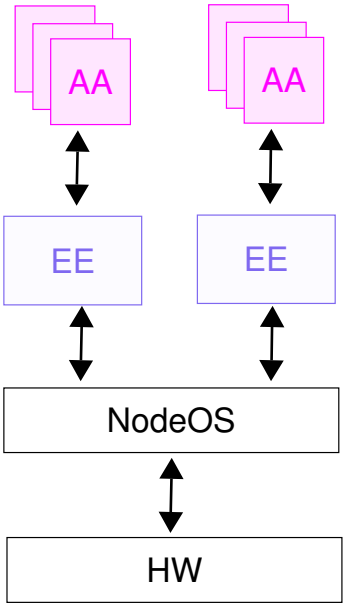


Figure 3.2: The Canonical Active Node Architecture.

run on it. In this sense, AAs are equivalent to Applets in Java. Several EEs can be simultaneously active in a node, provided that they all respect the interface with the NodeOS. Every service provider, for example, might develop its own EE optimized for its applications and download it into the nodes, in which they run. This is the reason why the ultimate node resource management has to be realized by the NodeOS. Analogously to the EE, the NodeOS controls access to the actual node resources (CPU, memory, interfaces, etc.) by the EEs. In this way, interference among EEs is prevented. Otherwise, its tasks are the usual ones for a networking operating system, with one capital exception: It presents a strongly abstracted view of the hardware resources to the EEs, so that portability and usability of networking AAs is increased. Typically, resources are summarized in four categories: Threads (computation), channels (bandwidth), memory and flows (data paths). Direct access by the EEs to the hardware resources is thus precluded for usability as well as security reasons.

In spite of all the efforts, though, the unsolved problem for both Active and Programmable Networks is how to solve the performance vs security vs openness trade-off. Typically, existing proposals achieve a good score on at most two of them. This obviously represents a strong drawback for its successful introduction in public networks.

A number of proposals have strongly influenced the work in this Thesis. Some present a commitment to performance and flexibility, like the work under direction of Prof. Zitterbart. Others try to combine strong security with performance, like the joint work at the Washington University in St. Louis and the ETH Zürich, or the Switchware project at the University of Pennsylvania. Resource

management at the network and node level is the emphasis of the Darwin project at Carnegie Mellon. Since the design presented here borrows heavily from the lessons learned from them, and because they exemplify the broader category of A&PN, Annex A has been devoted to their review.

3.2 Security Implications of Mobile Code in Hardware Systems

As was explained in the introduction, the openness proclaimed by Active and Programmable Networks brings along acute security threats for the network operator. The ability given to third parties to participate in node management and especially the introduction of Mobile Code[2] into such architectures introduces a number of additional threats to the correct network operation. This is undoubtedly one of the main reasons, why Active Networks have not achieved widespread success. Since the proposal presented in this Thesis also relies in open nodes supporting Mobile Code, the nature of such threats will be analyzed in this section. The emphasis will be put on the dangers specifically derived from the open node management and the ability to introduce functional changes through the download and installation of foreign code. Other threats derived from possible attacks to the code while being transported or to the software repository are not specific to an open node and will not be addressed here.

More concretely, it is assumed, that any code being downloaded into a node has not been compromised (which can be enforced by using encryption, signatures and/or hashing techniques) and that the sending source has been authenticated and authorized to perform such a task (s. Fig. 3.3) [Kar01]. Moreover, once introduced into the destination node, the code might suffer attacks from *within* the node, which could turn an otherwise safe program into a threat. This possibility arises only if the node management itself is unsafe. Because of that, the emphasis will be set on the threats that Mobile Code implies for an otherwise well-behaved node and on which measures can be taken to guarantee, that its safety will not be compromised.

Under these assumptions, Mobile Code can provoke two kinds of attacks on the hosting node [Eck01]:

- Passive Attacks: The goal of these attacks is to compromise the system confidentiality: To obtain unauthorized information from the system without altering its state. Access to memory ranges associated to other applications is a typical example.

[2]The term Mobile Code refers in this context to code which is introduced into a node dynamically. It is not meant to refer to self-transporting code, as is the case with Mobile Software Agents or analogous technologies [Pso99]. In the context presented here, code is retrieved and installed into a node under the control and surveillance of some (hierarchically superior) management instance.

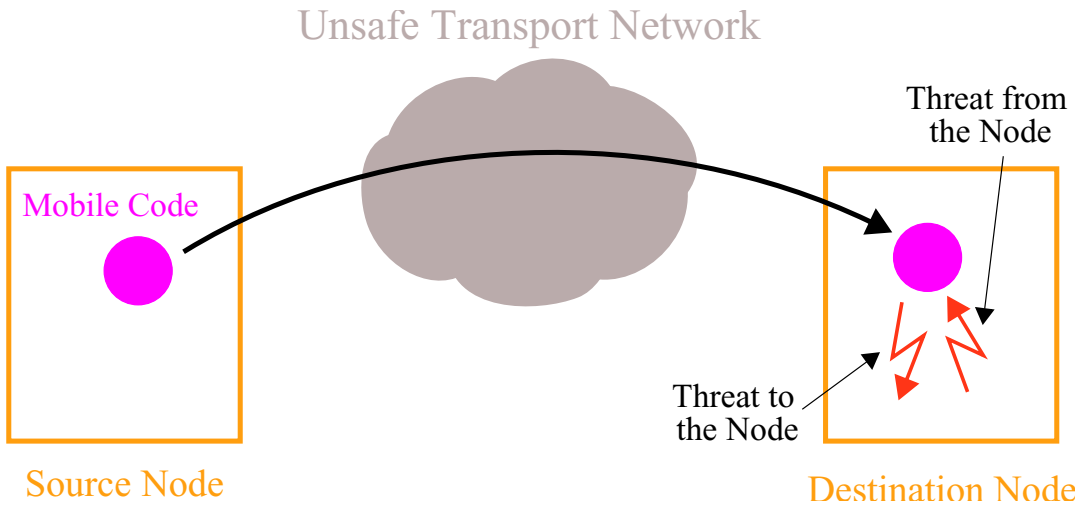


Figure 3.3: Threats associated with Mobile Code.

- Active Attacks: In this case, the state of the system is modified. This includes illegally modifying data stored in the node, as well as process state or even creating new processes. Alternatively, the availability of a certain resource might be compromised. This implies a restriction in accessing that resource for authorized parties. A classical example could be the thrashing of a memory interface with senseless requests, which although immediately discarded by the memory controller, nevertheless block the memory interface to other modules.

In order to protect a system from attacks, a number of techniques can be employed:

- Access Control: The task of Access Control is a double one: On the one side, it controls and schedules access to resources according to a set of rights associated to every user-resource pair. On the other, it has to ensure that the requesting units are properly identified and that their rights have not been altered. Although this prevents resource misuse, it does not ensure, that the application making use of it implements the functionality, which was proclaimed by its designer. Putting it in other words, it can not guarantee the *safety* of the code, defined as the compliance between the advertised functionality of a module and its real functionality [Eck01]. Still, Access Control is implemented in some form or other in all Operating Systems and other forms of software, like virtual machines. A good example of this technique is the widespread use of virtual memory concepts.

- Proof-Carrying Code: In order to guarantee the code safety, a relatively novel approach consists in sending with the code a proof, that its execution does not violate the safety policy of the receiving system [NL98]. This

eliminates the need for run-time checks. Nevertheless, the scalability of this technique with the number of policies and applications is obviously a matter of concern. In any case, in order to prove the safety of an application, a complete specification of its functionality has to be provided. This task could be performed with the help of formal methods. Unfortunately, a thorough description of all possible states, that a program might reach is a very hard task to achieve and the analysis of its compliance with a certain safety policy is far from being trivial. Hence, this technique is not widespread.

- Safe Programming Languages: Many attacks can be realized exploiting security holes in the programming language itself. Never ending loops, for example, can hang a process for ever, using up processing time in a useless manner. To eliminate such holes, a number of projects have tried to come up with safe programming languages [AAKS98a], [Int95]. Among others, the characteristics of such languages include strong typing, exception handling, transactions and no direct memory operations. The main problem with these languages is, that they strongly restrict the functionality and the expressiveness of the language. This reduces their usability and they enjoy therefore little success.

As a consequence, from the above mentioned mechanisms only Access Control is in widespread use in most modern systems. Although Access Control plays a very important role in protecting against passive as well as active attacks, it can not ensure the safety of the code. The possibility of "acceptable" behavior (from a syntactic and resource-usage point of view) which, nevertheless, results in a malevolent attack (from a semantic point of view) is still present. It is this semantic threat, which is mostly used by attackers: Security holes in complex applications, configuration errors, software bugs and defective security measures are exploited to that end. Access Control can only mitigate the effect of those attacks, but not prevent them.

In principle, in a node providing programmability at the software as well as at the hardware level, attacks to both software and hardware are possible. In reality, though, software attacks dwarf hardware ones. The reasons lie at hand: In order to perform a hardware attack, a detailed knowledge of the platform is necessary. Furthermore, the heterogeneity in the platforms (different components, different architectures) is much larger than for software systems. Last but not least, access to the hardware infrastructure is usually more complex than access to the software, since one of the typical tasks of an Operating System is precisely to abstract (read: hide) the concrete platform details.

Software attacks are easier to perform: On the one hand, the *de facto* monopoly in office software enjoyed by Microsoft presents a unitary software platform for attackers. This monopoly is also present in the networking world, where Cisco and its IOS play an overwhelmingly dominant role. Accordingly,

in the PC & Server market as well as in the networking equipment one, attackers can tailor their efforts to only two platforms and be sure, that their efforts will have a broad effect. On the other hand, as stated above, software controls the functionality of the whole platform, hardware included. Hence, exploiting a software bug can have disastrous consequences even at the hardware level.

In the face of these threats, a number of security architectures for Active Networks and Active Nodes exist [MLP+01], [Kar01], [LYR02], [AMK+01]. They all share a set of common goals and techniques, which will only be mentioned here. The goals they pursue with respect to securing the node against malicious mobile code are:

- Isolation among service providers: Assuming, that several service providers will share a common platform (a programmable node) to run their services, it is of capital importance, that they will not interfere with each other. Their individual levels of QoS have to be kept, and passive and active attacks by a service provider against the others have to be blocked.

- Isolation among services: Arguably, from the point of view of the node, it is irrelevant, whether two services pertaining to the same service provider and running on the same platform interfere with each other, as long as they do not affect foreign applications. Nevertheless, most security architectures try to come up with mechanisms to provide inter-service isolation in all cases.

- Enforcement of resource usage agreements: As stated above, individual QoS levels have been agreed upon *a priori*. It is the role of the security architecture to enforce such agreements, not only to prevent resource misuse, but also to provide for a coherent accounting.

- Node protection against unexpected service behavior: From the point of view of the network operator, the fundamental requirement of a security framework is the maintenance of node integrity. On the one hand, the node shall keep on working uninterrupted in the presence of any unexpected service operation. On the other hand, furthermore, the performance and safety of the node itself shall not be endangered thereby. Otherwise, the network operator would effectively lose control of its network, putting its reliability at risk. This is, without doubt, the main reason behind the lack of success for Active Networking: The fear of unreliable network behavior by the network operators, in spite of the different security proposals.

As stated in the previous section, the proposals found in the literature rely on a combination of 3 approaches to guarantee these goals: Execution Environments, Operating Systems and Safe Languages. Basically, all these proposals perform some form of Access Control, as defined above, by both the Execution Environment (for intra-operator isolation) and the Operating System (for inter-operator

isolation). The Execution Environment has a Virtual Machine-like role, while the Operating System regulates access to low-level node resources, that are shared by all applications. Since most Active Network architectures are pure software designs, in which the hardware resources are abstracted by the OS, their security architectures do not contemplate providing extra mechanisms directly at the hardware level.

Nevertheless, programmability at the hardware level presents a different panorama in which additional security threats are possible. As will be explained in Chapter 4, the design presented in this Thesis provides the operator with a platform within the node for the deployment of hardware applications, so that the performance requirements of modern networks can be kept up with. The evolution of IC technology in recent years makes this possible. The evolution in chip size, on-chip memory and gate density opened the door to the integration of whole systems on a single chip (SoC), thus facilitating the translation of whole tasks to the hardware domain. At present, the sustained development in VLSI technology allows to go one step further: The advent of Multiple-Systems-on-a-Chip (MSoC). In this context, companies are specializing in developing system *building blocks* to be integrated in bigger designs. Clear examples of these so-called Intellectual Property (IPs) blocks are memory controllers, interfaces, serializer-deserializers, etc.

As a consequence, complex systems consist of an amalgam of third-party and in-house developed blocks. The interaction among them is a source of possible conflict, since the system integrator has no access to the details of the IPs, which are delivered as "executables": Synthesized VHDL code with an interface description for compatibility. I.e.: Future chips could host several independent systems, composed of partially unknown and uncontrollable modules, which will nevertheless have to share a common pool of off-chip resources (memory banks, I/O, buses, etc). In this context of shared ICs, a number of threats to the hardware integrity suddenly become relevant.

The main difference to software attacks is the additional possibility to perform attacks at the electrical signal level [Had99]. The trick here is to provoke high currents in selected interconnection entities inside the chip, which can provoke the physical destruction of the hardware. This is possible even with restricted information about the hardware platform: It suffices with access to some I/O pins. Assuming, that they are used to interconnect the chip to some external data source, like a memory bank, by simply configuring them as output instead of input pins such currents will occur due to the mismatching electrical levels set at both ends. Although the destruction of the platform can not be guaranteed, the risk of damage and/or overheating is greatly increased. This new kind of threat can be downplayed (although not completely solved) by performing code sanity checks prior to installation. In such checks, the logical configuration of the interconnection entities are validated prior to installation. If these tests are

performed off-line, they do not necessarily represent an additional latency in the integration of new functionality to a programmable system.

A second class of hardware threat are attacks at the logical signal level. In those cases, signals are generated, which although electrically correct, make no sense for the rest of the system. The goal is to drive the system into an unpredictable state. In order to prevent such attacks, a thorough testing with known test vectors would be necessary prior to installation in the hosting system. As with Proof-Carrying Code, exhaustive testing is close to impossible for similar reasons.

The last type of threat intrinsic to programmable hardware systems are analogous to classical software threats: A valid operation cycle, from the electrical as well as from the logical point of view, is used to perform a malicious task. The dangers and mechanisms to reduce them are basically the same as for software systems.

In principle, it could be argued, that sufficiently strong surveillance by the OS should suffice to ensure adequate security. The problem with that argument is threefold:

- Hardware abstraction hits performance: The reason why certain functions are realized in hardware is basically to achieve better performance. To that purpose, the details of the platform have to be known to the designer, so that the solution proposed takes advantage of its characteristics. If such details are obscured by the OS, the advantages of hardware design are strongly diminished. As a consequence, hardware programmability only makes sense, if unrestricted access to the platform is possible. Furthermore, even to the OS many details of the concrete hardware implementation are hidden: I.e., which I/O pins are used to access memory is not an information, that the OS disposes of. Accordingly, a number of attacks could not be detected or solved by the OS, since it lacks sufficient information about their nature.

- Different time scales: In order to guarantee compliance with the security policy in the presence of programmable hardware modules, their behavior should be surveilled by the OS. Of particular importance is the access to shared resources like memory, buses, peripherals, etc. Unfortunately, the time scale relevant for these resources is orders of magnitude finer than for software systems. As an example, the Linux OS performs a context switch among processes every 10 ms. A hardware application trying to access an SDRAM memory bank expects an answer in a few ns. This discrepancy implies, that software can not effectively control hardware operations, for any reactions to unexpected behavior would take too long to be detected and solved.

- Security hole for pure hardware modules: In the presence of logic blocks realized completely in hardware, with little or no communication with software instances, the situation mentioned above leaves a number of security holes open: A hardware module, whose behavior toward the software interface is correct, might nevertheless misbehave at the hardware level without possibilities for the OS to efficiently control it. This opens the door for hardware attacks at all levels: electrical, logical and semantic.

Summarizing, the opening of network nodes to hardware programmability brings a forgotten set of hardware threats to new relevance, which traditional security infrastructures are not prepared to cope with. One of the aims of the present work is to explore the possibilities to control and prevent such threats by implementing Access Control directly a the hardware level. This will strongly influence the design choices presented in Chapter 6.

3.3 QoS in IP Networks

The definition of Quality of Service (QoS) is a matter of some dispute [ITU94a], [CNRS98]. In this work, and following [BDH+03], QoS will be used in the following sense: "A set of context-dependent (i.e. on the user and the service) requirements to be met by the network end-to-end to provide a degree of satisfaction to a user of the service". A QoS architecture describes a structured solution to meet those requirements. As stated by this definition, what satisfies the user requirements depends strongly on the type of service and the user environment. This includes parameters like technical capabilities (bandwidth, processing power on the end user's equipment), price and others. Nevertheless, the emphasis in this work, as in the definition, is set on the network. It is the QoS architectures, that have been developed for network QoS support that will be reviewed, as well as their impact on router design. Generally speaking, because there are diverse services and users, their requirements are best served by separating their traffic into classes and giving them a differentiated treatment. That is the purpose of the following two QoS architectures developed for IP networks.

There are a number of parameters that quantify QoS[3], the most common of which include total delay (caused by transport, queuing and processing), delay variations (jitter) and consistent data throughput capacity (bandwidth). The required level of each parameter, and thus the service level expected, is set in a contract between the users and the network. The contract is called the Service Level Agreement (SLA). Additionally, it includes other relevant data like the minimum mean time between failures, actions that will be taken in case of

[3]As already mentioned, other parameters are equally important in the setting of a QoS level, like the price or the disposability of the service. In this work, however, the focus will be on the *technical* strategies employed in the network to ensure a certain traffic behavior. Other techniques (i.e. economic), can be found in [BDH+03].

contract breach by any party, etc. This contract can be static or dynamic. In the first case, it is set among the parties a long time prior to data transmission, usually has a long validity and applies to large traffic aggregates. Dynamic SLAs, on the other hand, are negotiated in the connection establishment phase and are usually only valid for the duration of it. The granularity unit is usually the flow.

3.3.1 Integrated Services

The first approach to QoS in IP networks tried to emulate the hard guarantees given by the telephone network and by ATM. It was called Integrated Services (IntServ) and its basis was the reservation of bandwidth for traffic flows between the sender, receiver and all interconnecting routers [SB95], [Whi97], [XN99], [BCS94]. Each flow is an individual, uni-directional data stream from the sender to the receiver. IntServ is able to reserve resources from the source to the destination address with the requirements of the sender's application. Therefore special functionality in routers and applications is necessary. It also implies that information ("state") about every reservation and its corresponding packets has to be kept in every router in the path.

Since applications have different needs in terms of bandwidth, delay and jitter, the IETF has defined several service classes for IntServ to separate the flows into traffic types. As way of example, videoconferencing requires an absolute guarantee and an upper bound on delay, whereby non-critical applications like FTP do not need any higher guarantees than best effort. This classification helps to define the kind of service that a certain application type needs, but the actual reservation is done on a flow-by-flow basis.

Every service class has a different hard limit on bandwidth, delay and jitter. They are:

- Guaranteed Service: It provides stable upper bounds on delay, with the desired bandwidth between sender and receiver. Therefore this service class can be seen as a leased-line emulation.

- Controlled Load: This service class delivers the datagrams with a very high probability of small delay and little loss. Nevertheless, higher loss and delay are possible for a fraction of the traffic. It emulates a virtual lightly loaded best effort network.

- Best Effort.

To perform and coordinate the reservation process along the data path, a signaling protocol was necessary. Therefore the Resource ReSerVation Protocol (RSVP) was introduced [BZB+97], [Wro97], [Her00]. The RSVP sender sends a reservation request along the path. The actual reservation takes place on the reverse direction through the sending of a reservation message by the receiver.

Senders characterize outgoing traffic on lower and upper bounds on delay, jitter and bandwidth. Then, a RSVP path message (Path) is sent to the receiver to install reverse routing information in each router and to characterize the sender traffic with the traffic specification (TSpec). Each intermediate router on the path downstream between source and destination must update and store path state information (s. Fig. 3.4). The state information is identified by the sender template.

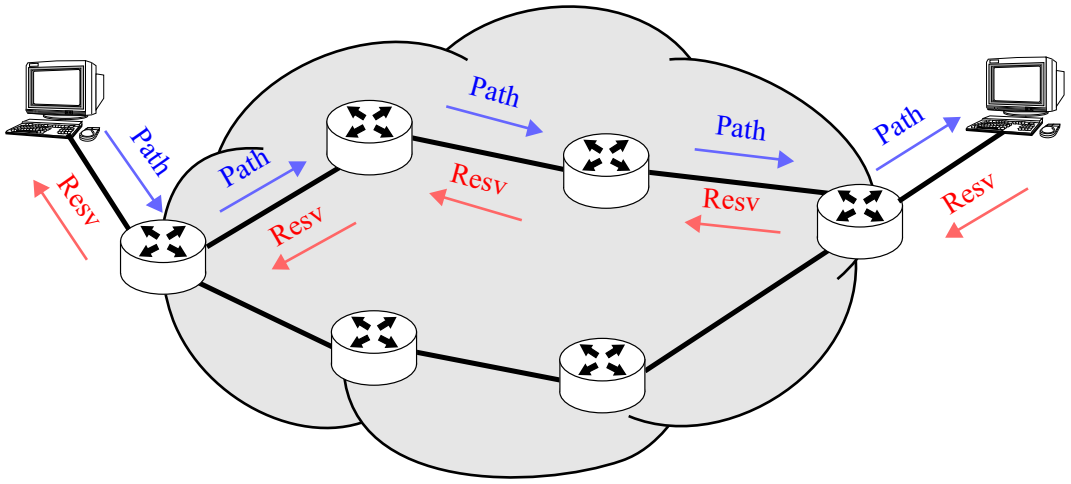


Figure 3.4: The RSVP Reservation Process.

For the reservation the receiver sends back a reservation message (Resv) after receiving the Path message from the previous hop. The Resv message contains a request specification (RSpec) indicating the service type, a filter specification (FilterSpec) characterizing the packets for which the reservation is done, an indication of the reservation style and optionally but generally used, a reservation confirmation object to indicate to the sender, that it should acknowledge the reservation establishment. The RSpec and the FilterSpec together are called the flow specification (FlowSpec). The Resv message goes back upstream the path to the sender and at all intermediate hops the resources are then reserved for the flows (s. Fig. 3.4). Hence, the reservation is receiver-based. This approach is advantageous for multicast transmissions, where (frequently only) one sender streams data to a group of receivers. If the sender had to reserve the resources itself, it would have to reserve bandwidth individually for every receiver. As it is, the sender requests bandwidth for a multicast group and the receivers, if they are available, reserve the resources individually. This also makes the addition of new receivers during a transmission easier.

Both states, the path and the reservation have a defined time-out, often 30 seconds. Therefore they must periodically refresh their states in all routers in order not to get the reservation torn down. The path state has to use the Path message and the reservation state has to use the Resv message for the periodic

refresh, known as "soft-state". Soft-state is used to enable the router to modify the resources very fast and to prevent dead reservations from blocking resources in the case of network or user malfunction or disconnection.

If something goes wrong when a router gets the Path message, a path error message (PathErr) is sent back toward the sender to indicate the error to him. A reservation error message (ResvErr) is sent back to the receiver if the reservation could not be established. Reasons for this could be an admission control failure, unavailable bandwidth, requesting a service which is not supported, a bad flow specification or an ambiguous path.

To end a normal communication in IntServ the messages path tear (PathTear) and reservation tear (ResvTear) are used to explicitly tear down the two reservation states. The PathTear travels toward the receiver and the ResvTear travels toward the sender.

The main drawback of IntServ resides in its poor scalability. The fact, that state information must be kept and periodically updated for every flow and on every router imposes a double burden on the router: On the one side, a storage burden, since routers do not usually dispose of the same amount of memory as PCs do. On the other, a processing burden, derived mainly from the classification operation: With every incoming packet (and at every router), the classifier has to detect to which flow it belongs and which treatment is adequate. Flows are usually identified by several header fields, which represents a hard search problem (s. Section 2.2). This is aggravated if the number of flows is large, as is the case in today's backbones.

Other disadvantages are related to its introduction in modern networks: In principle, every router along every possible path must support RSVP and its associated reservation mechanisms, if an end-to-end guarantee is to be given. This represented a strong drawback in its applicability, especially in the early stages of deployment.

The orthogonality between the resource reservation and the routing processes is another cause of concern in the IntServ architecture. Since data networks use a shortest-path routing algorithm to decide the route to be taken by a packet, in absence of routing changes all packets directed to the same egress point will take the same route, even if there are alternatives. As a consequence, the shortest paths between any two network edge points tend to concentrate most of the traffic. Inner backbone links, which are shared by many end-to-end paths are the ones most affected. As a consequence, it can very well be, that a certain reservation can not be accepted along the shortest route, because those links are saturated. The reservation request will then be rejected, *even if there are alternative routes, which would comply with the reservation specification.* The opposite also applies: If a routing change occurs and the route taken by packets belonging to a reservation is deemed suboptimal (although valid), they will be re-routed along a shorter path. That does not mean, however, that the new path

disposes of enough resources to accept the new reservation. MPLS, as will be explained shortly, leverages this problem.

Its scalability problems provoked the search for alternatives. Several reservation protocols (and their associated mechanisms) were developed [AFB98], [PS99], that tried to overcome its limitations. Nevertheless, it proved unfeasible to achieve hard guarantees and fine granularity with stateless routers and scalability. In this trade-off, a second approach emerged, that sacrificed granularity and provided only soft guarantees: Differentiated Services.

3.3.2 Differentiated Services

The major goal of DiffServ [GEH+98], [XN99] was to make QoS support more scalable by reducing the state, that had to be kept at the routers (albeit not eliminating it completely) and also the processing burden, especially at the core routers. To that end, the following strategy was devised: Incoming packets are classified at the network edges according to their header content. This complex classification associates the packet to one of a *small set* of possible traffic classes. Traffic differentiation takes place at this higher level of granularity. Furthermore, other routers along the path do not reclassify the packet, neither do they perform a route lookup on it. The edge router sets a mark in the IP header, indicating the class to which it belongs. All subsequent routers check this marking and forward the packet accordingly. Since this is a unidimensional, exact matching problem, with a reduced set of possibilities, the search time is much faster that doing a full route lookup. This procedure somewhat imitates the VCI/VPI checking scheme in ATM networks. By reducing the number of aggregates and thus the state to be kept and removing the classification process from core routers, scalability strongly improves.

Although resources are statically reserved for every class at configuration time, DiffServ does not provide hard guarantees, not even at the class level. Every class is set in a priority scheme. Different priorities define only a relative relationship among classes ("A will be treated better than B") but no quantitative values ("A will get 10 Mbps and 50 ms delay"). Hence, DiffServ is incapable of guaranteeing (in a strict sense) delay and bandwidth bounds, as opposed to IntServ. The use of resources by every class is monitored and enforced by metering, marking, shaping and scheduling.

Furthermore, the classes only define the service level on a local basis. I.e., although a higher priority class will always be treated with greater deference, every router can redefine the amount of resources associated to it and the exact treatment (in terms of scheduling, shaping and the like) that it will receive. Although a network-wide consistent policy can be implemented by the administrator, by coordinating the configuration of all routers, internetworking remains unsolved. Different network operators are usually involved in most data communications. Even if they all support the DiffServ architecture, class definitions and their as-

sociated resources are not subject to standardization. Thus, their policies do not have to be compatible. Consequently, end-to-end coherence in QoS delivery can not be guaranteed.

The mapping between traffic types and classes is also not subject to standardization. It falls on the hands of the administrator (or the end user, if such capability is provided) to decide, which traffic segment will be forwarded in which class.

The IETF introduced two further service classes for DiffServ:

- Premium Service respectively Expedited Forwarding: This class is often called Expedited Forwarding (EF) and is used to transmit the packets with the highest priority and to set a minimum on delay and jitter between the packets.
- Assured Service respectively Assured Forwarding: The Assured Forwarding (AF) is divided into four subclasses, each one is divided into three subsections with increasing drop precedence (i.e. packet loss probability). These classes ensure a higher priority than BE for traffic which can stand a higher delay and jitter than EF, like elastic video and audio streams.
- Best Effort

A marking for the packets was necessary to distinguish the service classes. Therefore a field in the IP Header was used. The so called Differentiated Services Code Point (DSCP) is a redefined Type-of-Service (TOS) byte in the IP header. References [NBBB98] and [BBC+98] define the DSCP of all Service Classes mentioned before. With these 6 bits an upper limit of 64 available classes is declared. This maximum amount of classes leads to the scalability of DiffServ, because the number of state information is proportional to the number of service classes, not of active flows. Since the marking is transported in every packet, no signaling is necessary to set up paths or reserve resources. This simplifies the operation of DiffServ networks.

A last issue in Differentiated Services networks corresponds to the usability of its 64 maximum classes. Studies have shown [DP00], that the interior classes enjoy a very similar treatment under a broad range of scenarios. The question thus arises, if it makes sense to sustain that many classes or if 3 or 4 would not suffice. From a market perspective, furthermore, it is difficult to convince customers, that they should pay more for a higher class, although the difference in service level is both unknown (since DiffServ does not quantify its guarantees) and small compared with adjacent classes. That is one of the reasons why commercial DiffServ offerings are restricted to very few classes.

3.3.3 MPLS

Multiprotocol Label Switching (MPLS) [RVC01], [AJ02], [XN99] represents an integration effort, drawing inspiration from ATM, IntServ and DiffServ. It was

not specifically designed as a QoS architecture, yet it has become the technology of reference for QoS support in modern networks. Its main goal was to attend to operator concerns regarding network reliability, survivability and utilization [AMA+99]. On top of that, the explosion in demand for VPN solutions triggered an interest in tunneling technologies, that helped to isolate traffic streams belonging to different customers.

Primarily, MPLS is a topology-driven stackable tunneling technology: It supports the establishment and management of paths between network edges at different aggregation levels (s. Fig. 3.5). In this sense, it is very similar to the functionality, that PNNI had for the establishment and management of virtual channels and virtual paths. The main difference is that in ATM networks the label stack (and thus the aggregation hierarchy) was limited to 2 (VCI/VPI). As in ATM, once established, the virtual paths are fixed and will not be altered by autonomous routing decisions, except in case of link failure or malfunction.

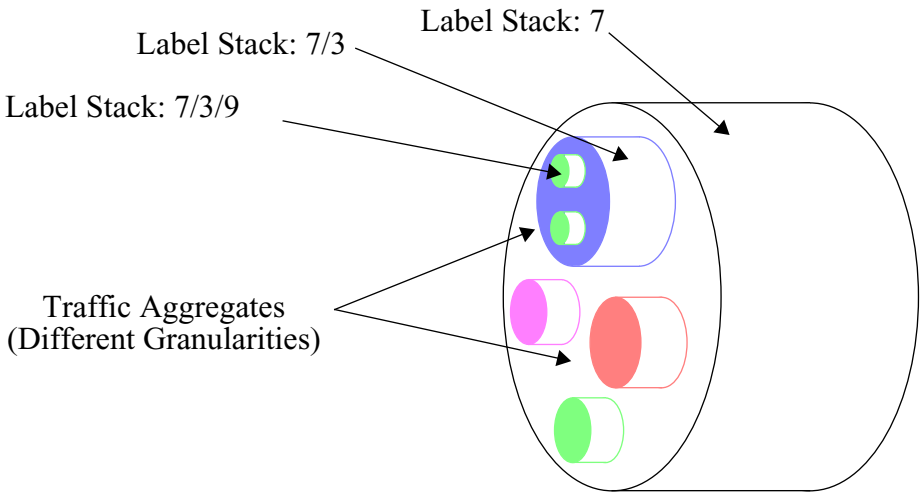


Figure 3.5: Hierarchical Tunneling in MPLS.

In MPLS, tunnels are set up with the help of two protocols:

- Constraint-based routing protocol: It is in charge of finding a way between the two endpoints prior to the tunnel establishment. A specific MPLS goal was to take into account more factors than just the hop count, especially in order to support load balancing and thus achieve a higher network utilization. Additionally, re-routing in case of link failure was to help improve network survivability and reliability.

- Signaling protocol: Once a path has been found, it corresponds to the signaling protocol the establishment of the tunnel. This includes the reservation of such resources as might be needed in the routers and the interchange of labels among them, exactly as was the case in ATM networks, for labels

have a local significance only. A MPLS tunnel is thus identified by the chain of labels being involved end-to-end.

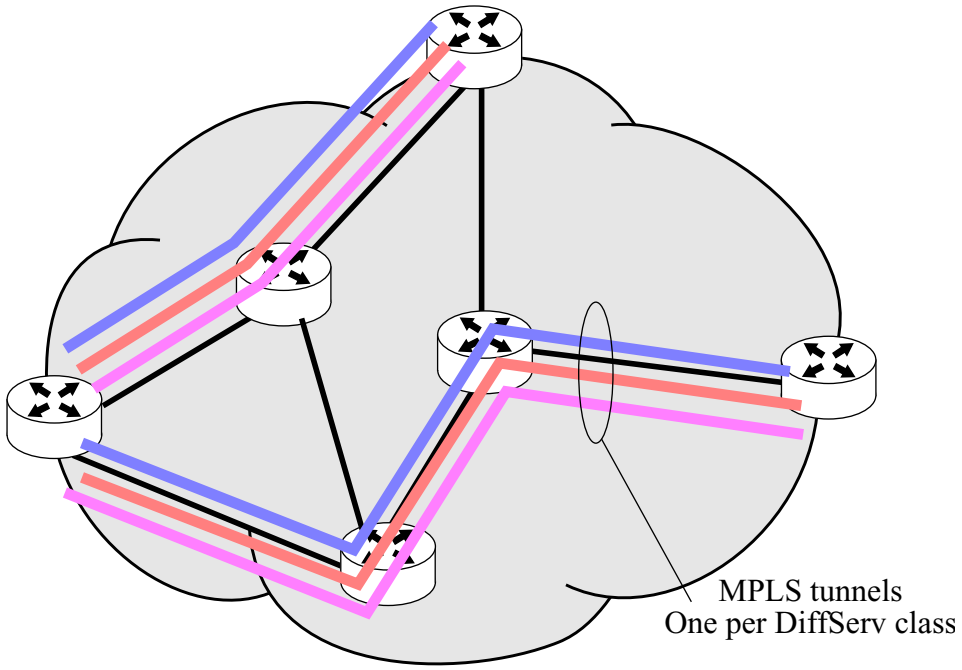


Figure 3.6: DiffServ on top of MPLS.

Classical protocols had to be adapted in order to integrate the new functionality envisioned by MPLS. For that purpose, routing protocols were extended with so-called Traffic Engineering capabilities. The two main exponents are Open Shortest Path First – Traffic Engineering (OSPF-TE) and Intermediate System-Intermediate System – Traffic Engineering (IS-IS-TE). Nevertheless, the multi-metric path calculation problem can also be used to account not only for operator needs (like higher network utilization) but also for customer needs: QoS parameters can also be taken into account. Thus, MPLS routing protocols are confronted with the task of establishing edge-to-edge tunnels responding to certain QoS and Traffic Engineering constraints. As was explained before (s. Section 2.1), general multi-metric optimization problems are NP-complete. This forced to include heuristics into the protocols to make the calculation tractable. Furthermore, in order not to induce a high processing burden, the number of tunnels and their dynamics are very low. As stated above, MPLS tunnels are topology-driven, i.e.: They are established between network edge pairs. Since network topologies have a slow rate of change, once calculated, they can be kept for long periods of time. Furthermore, they are generally calculated off-line and set up at configuration time, instead of being set up on-demand. This allows to outsource the heavy mathematical calculations to processors outside the routers most of the time.

Even if QoS support was desired, in order to keep the number of tunnels (and thus the management complexity) low, virtual paths could be only established for large aggregates. DiffServ is the architecture of choice for QoS support in MPLS networks: A reduced set of parallel tunnels is established between any two network edges, one per DiffServ class (s. Fig. 3.6). Resources are associated and scheduled according to their priorities. Nevertheless, contrarily to DiffServ, a signaling protocol was developed to automate the tunnel establishment and the corresponding resource reservations. For this purpose, two main contending protocols emerged, Constraint Based Label Distribution Protocol (CR-LDP) and Resource Reservation Protocol – Traffic Engineering (RSVP-TE), the last of which is an extension of the classical RSVP protocol designed for IntServ.

MPLS represents, thus, a control plane for the establishment and management of QoS tunnels. In the data plane, packets are treated like in DiffServ networks: They are classified at the edge and receive a label stack upon ingress in the network. Subsequent routers perform only a label swapping operation instead of a route lookup, plus the necessary metering, scheduling and shaping operations. This forwarding plane is usually realized in hardware to achieve better performance and scalability.

Lately, the MPLS framework has been extended to be used as a unified control plane for IP-over-optics networks. This is referred to as Generalized MPLS (GMPLS) [Man03]. It presents a coordinated set of protocols for routing, signaling and link management.

Summarizing, MPLS reunites a number of advantages both for operators and customers: First, it presents a (relatively) low management burden. Only a controlled set of tunnels is established. Since they are topology-driven, they are static over long periods of time. Second, it supports Traffic Engineering (in the operator's sense: as a means to increase reliability, survivability and network utilization) as well as QoS (DiffServ with a very limited number of classes). Both in a simple form, but also at low cost in terms of complexity. MPLS is meanwhile the architecture of choice in modern all-IP networks.

4 The Octopus Network Model

After reviewing in past chapters the most relevant work on Active and Programmable Networks, as well as its implications for the security of networks and nodes, this chapter will present the Octopus Network Model (ONM). It represents a new attempt at achieving a balance between openness, security and performance in the context of Programmable Networks. First, two case studies, that elaborate on the necessity and applicability of the architecture will be presented. Next, the main characteristics and properties of the ONM will be discussed. A more elaborated description of its main elements will conclude the chapter. Because of its central role in the architecture, the discussion on the Octopus Open Gateway will be deferred until next chapter.

4.1 Objectives of the Octopus Network Model

In a word, the goal of the Octopus Network Model is to transcend the openness vs security vs performance trade-off, that has severely limited the usability of Active and Programmable Networks in real-life scenarios.

The aims of this work are thus twofold: First, to augment network flexibility, i.e., its ability to expand its functionality quickly and efficiently. This aim derives from the observation, already mentioned in the Introduction, that an ever expanding set of services is being delivered online. Although the Multiservice Internet is based on a very flexible set of core protocols, the introduction of new services presents an increasingly burdensome task. First, because having a flexible data transport technology does not mean, that there is an equally flexible infrastructure to manage the network. And second, the interaction among services presents a threat to the safe introduction or alteration of functionality.

The second goal involves fostering the innovation potential of new network services by providing an infrastructure capable of managing disruptive new services without endangering network performance, reliability or security. Innovation needs freedom to develop novel and unexpected functionality and the ability to manage that functionality. Hence, simple and powerful network interfaces are needed, so as not to constrain that freedom. Furthermore, those interfaces must be low-level, providing access to the raw network capabilities. Abstractions from those capabilities would imply the impossibility of tailoring them to the necessi-

ties of new services, thus unnecessarily limiting what is possible and what is not. As a last requirement, not only must service development be as free as possible, but also its management. Since disruptive services can *per se* not be foreseen, management interfaces should be left to the design and control of the service developer.

Summarizing, this work foresees the separation of the network operator in two distinct roles: First, the "new" *network operator*, which would be in charge of data transport and basic functionality, as well as providing an extensible, service-friendly platform. Second, the *service operator* will have the role of introducing and managing new value-added functionality in the network. Obviously, both roles can be taken up by the same company, although they are logically separated activities.

In order to clarify the kind of service extensibility, that is herewith envisioned, the following two sections will review typical scenarios. At the same time, the relevance of Programmable Network-based solutions in those scenarios will be highlighted.

4.2 First Scenario: Security Gateways

Security Gateways are a relatively new kind of network node. With the increasing concern with security, specialized devices have appeared, that perform these tasks. In the past, routers had performed all networking-related activities. However, security-related tasks are usually very computing-intensive and can be seen as an add-on to pure bit transport and routing. Consequently, in order not to endanger the performance and stability of pure packet forwarding (the most critical part of networking), this new kind of appliance was created.

The role of Security Gateways can be simply stated as inspecting traffic and deciding if it represents a threat to the network to be protected. Should it be so, then some defensive measure should additionally be taken. This includes not only preventing incoming traffic from performing active attacks, as defined in Section 3.2, but also passive ones, like retrieving confidential information about the network configuration. In the first category such functions fall as firewalling, intrusion detection, deep packet inspection for application filtering, virus and worm scans, etc. In the second category, Network and Port Address Translation (NAPT), proxying, encryption, logging and accounting can be identified, among others. Most of these tasks involve some sort of packet classification up to the application layer. This is a repetitive, very computing-intensive task, well amended to hardware implementation. In fact, most Security Gateway appliances use some form of hardware support to achieve the needed performance.

It might nevertheless be justified to ask, if such appliances are needed at all, since anti-virus programmes and firewalls are typically also present in end user devices, especially PCs. The fact is, that some of the tasks mentioned above, like NAPT, can not be performed by corporate users, since their goal is precisely

to hide the existence of individual computers behind the Gateway in the first place. Furthermore, a number of additional concerns arise with the end user-only solution: First, not all end nodes have necessarily anti-virus programmes and firewalls installed. Even if this would be the case, there is no way to ensure, that those programmes are up-to-date and consistent with the corporate security policy. Next, typically the network administrator has no access to the end nodes, thus preventing her to enforce the corporate policy in the way that a Security Gateway does. Besides, common programmes have a tendency to forgo threat analysis coming from the *inside*, i.e., from computers within the network to be protected. Hence, to keep a coherent and up-to-date network-wide security policy, Security Gateways *plus* end user firewalls and anti-virus programmes are needed.

In spite of it all, attacks and/or erroneous configurations, especially in end nodes, can not be completely prevented. A typical example would be an insider disconnecting her firewall and bringing in an infected diskette. Consequently, a centralized Security Gateway solution, situated at the entrance to the corporate network (a very common solution) is also suboptimal. The reasons are clear: Such a solution can not prevent attacks from the inside. Indirectly, it relies on cooperative user behavior and is therefore prone to malevolent and/or dangerous actions. To prevent this, a distributed topology of Security Gateways has been proposed [CLS+04], [LMK+03], as depicted in Fig. 4.1. By placing appliances between critical network areas (departments, segments, server farms, etc.), the spreading and reach of attacks can be minimized. None the less, in order to coordinate the distributed set of appliances and keep a coherent policy, some remote management mechanism is needed.

In this context, a number of questions can be asked about the ideal characteristics of a Security Gateway and its relationship to Programmable Networks. First, Security Gateways will frequently fall under the management of a centralized management instance and will not be open, like in a Programmable Network environment. Second, most providers deliver their products with an integrated software suite, which can not be expanded with third-party code. Under these circumstances, there seems to be no relationship with Active and Programmable Networks. And yet, Security Gateways could greatly profit from Programmable Network-like architectures.

First, the security and safety of its firmware are strong concerns, due to the critical task performed. For such functions as are realized in software, in order to achieve good performance, they are typically implemented in the OS kernel. As many IOS examples have shown, the risk for network equipment derived from bugs or security flaws in networking code is tremendous. This is especially so for the appliance charged with protecting the rest of the network. Accordingly, it seems advisable to introduce protection mechanisms similar to those present in A&PN: Execution Environments (Virtual Machines) and Access Control for node-critical resources. For such applications as are not performance-critical (management applications, for example), this solution should be viable. For the

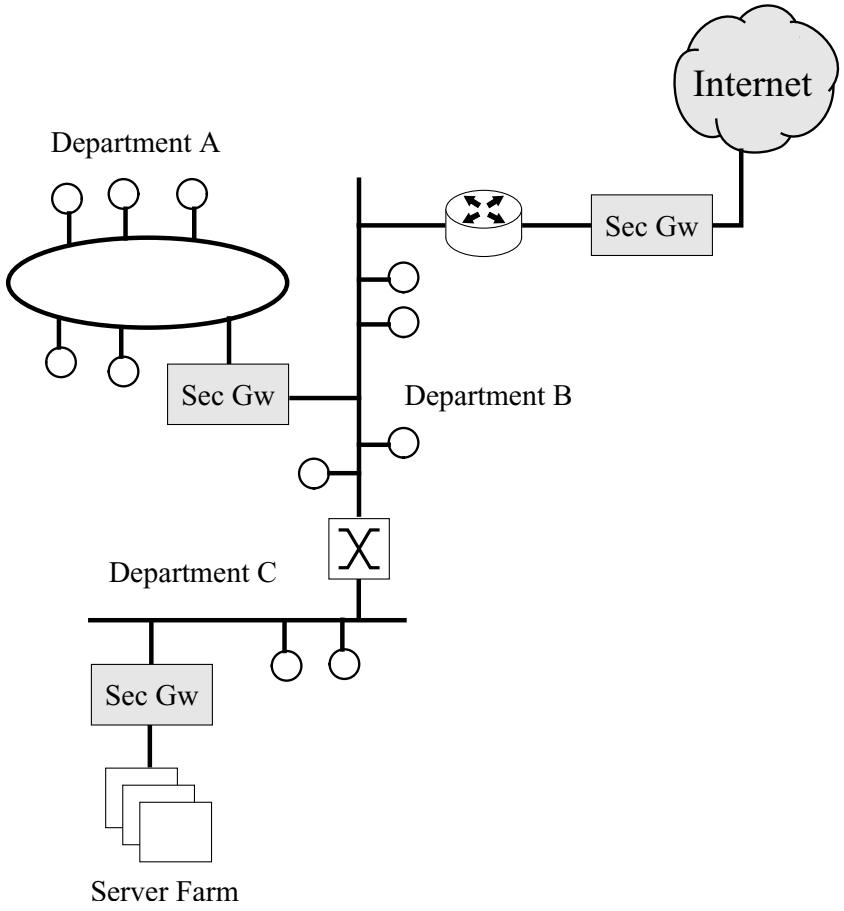


Figure 4.1: Distributed Security Gateway Scenario.

rest, a trade-off exists. A possible way out of it involves the use of programmable hardware. Nowadays, FPGAs support the inclusion of microprocessors in their structures, be it as synthesized soft-cores or co-located hard-cores (s. Section 2.5). By placing performance-critical software in such cores, a double goal would be achieved: First, the intercommunication burden to the hardware platform, which introduces a considerable overhead, would be greatly reduced. Second, by isolating the applications in separate processors, the risk of interaction and the effect of misbehavior could be diminished. Access Control mechanisms located at the hardware level between such processors and the rest of the system could help in the task.

The inclusion of hardware support is common in Security Gateways, as explained above, as the only means to achieve the necessary throughput. It might be argued, that software-based system performance is rapidly increasing. None the less, at least in the mid-term there is no sign, that software solutions will be able to achieve the necessary throughput. If, furthermore, traffic keeps increasing at its current pace, the mismatch between software system performance

and traffic processing needs might actually increase. In that context, hardware support will play an even greater role. Section 3.2, reviewed why software control of programmable hardware systems can not achieve good security. In proprietary solutions, like the ones presently in the market, it might be admissible to accept the lack of security checks at the hardware level, although it represents a clear risk. Nevertheless, the more functionality is placed in hardware, the more critical its security becomes. In this scenario, Access Control and QoS management at the hardware level could highly contribute to eliminate such risks.

Furthermore, besides the threats coming from potentially malicious (or maliciously modified) code, a second set of threats exist. The number of tasks implemented by a Security Gateway is large, and hence also the number of applications (in software and hardware) running on it. Without strong resource and security control, as provided by Virtual Machines and hardware resource managers, the risk of malfunctions due to application interaction is real.

Dynamics also represent an important factor. A critical element in Security Gateways is the ability to rapidly react to new threats. The inclusion of new classification rules to detect attacks or regular expressions to identify new worms are but two examples of updates. Additionally, new algorithms to better detect threats or to cipher code are continuously emerging. On top of that, the range of tasks being performed by these network elements has greatly expanded since their inception, only a few years ago. As a consequence, these appliances have to support a high degree of upgradability, at the software as well as the hardware level. There is no real difference between performing an upgrade and installing a new service, especially in programmable hardware. In both cases, the FPGA has to be (partially) reprogrammed. In this sense, Security Gateways have to support mechanisms for remote functional extension, exactly as Programmable Networks do.

Although a centralized Security Gateway solution is common, distributed approaches are increasingly relevant. In order to keep a coherent security policy and manage such distributed systems, mechanisms have to be provided to download and install programs in a coordinated way. This also involves authenticating and authorizing such downloads and digitally signing the code.

In short, a high-performance distributed Security Gateway architecture with upgrade capability presents the same characteristics as a classical Programmable Network and can profit from the mechanisms developed for them.

There are, nevertheless, some differences between this scenario and "classical" A&PN scenarios. The first is related to openness. As seen in Section 3.1, one of the goals was to open network nodes to service management by third parties. This scenario is not considered realistic for Security Gateways. The second main difference resides in the service placement. Most A&PN proposals foresaw the alteration of existing routers to support new functionality. In this case, Security Gateways are separate, independent devices situated beyond the public network. None the less, the case study presented here shares many commonalities with

those proposals, and the architectural implications certainly hold. The next section addresses the missing links with traditional Active Networks.

4.3 Second Scenario: Network-supported Digital Rights Protection

One of the main arguments behind A&PN was, that some services are better supported *inside* the network than on end-devices. This derives from the fact, that certain services can take advantage of information and capabilities, that only the network possesses. A number of services were mentioned in this context: Media transcoding, web caching, Virtual Private Networks (VPN). In this case study, a new and increasingly relevant example will be presented: Network-supported Digital Rights Protection.

The distribution of digital content over the web presents obvious dangers for the property rights' holder. The ability to easily and cheaply reproduce the content without quality degradation is seen as a tremendous drawback by music companies, video stores and the like, and prevents them from selling their products online. One promising technique to fight against piracy is watermarking: A discrete mark is introduced in the media stream (typically by imperceptibly changing some media attribute, like luminance or color in a video stream), that allows to identify the content's proprietor. To be really effective, though, it must be possible to identify the origin and destination[1] of every stream, so that illegal copying can be brought to justice if detected. An additional requirement is non-repudiation: It must be possible to univocally identify the customer, even if she claims not to have been involved in the transaction.

Individual watermarking can be performed by the content provider's server during download (s. Fig. 4.2), but this solution presents several drawbacks: First, watermarking is a processing-intensive operation, which, in order to be personalized, would have to be performed for every stream being downloaded from the server. This would greatly increase server load and accordingly decrease its scalability. Second, non-repudiation is a non-trivial task in the presence of address forgery and/or password stealing. Third, for every transaction, a new media stream has to be transported over the network, since individual watermarking prevents caching.

Network support can offer an advantageous alternative (s. Fig. 4.3). If the watermarking functionality could be distributed among several nodes, strategically placed near the customers, a triple advantage could be obtained: First, the load could be shared among the nodes, improving scalability. Second, caching would be an option, thus transporting only a few untagged streams through the

[1]The relevant information in this case is not the network address, but the customer identity. The relationship in an e-business scenario, like the one presented here, is established between the selling company and the buying customer.

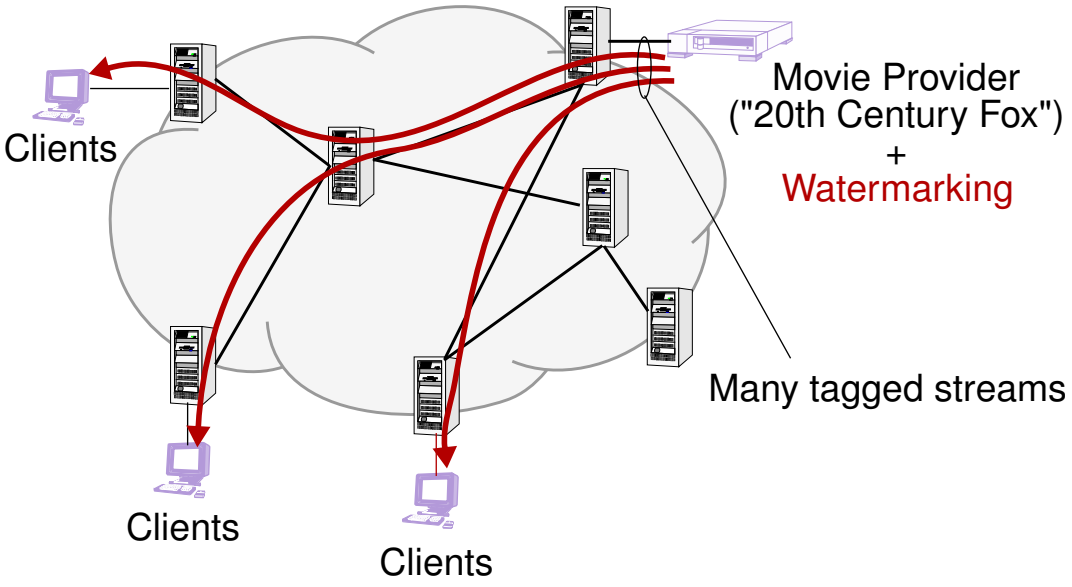


Figure 4.2: Individual Watermarking performed by Content Server.

network from the content provider to every caching/watermarking node. This would save costs both in terms of bandwidth and money. Third, since every customer is uniquely identified by its physical network termination point, non-repudiation is guaranteed. On top of that, a number of additional services could be bundled with watermarking: The network operator, which is generally seen as a trustworthy party by its customers, could perform the accounting and billing for digital transactions, in the way that Deutsche Telekom is already doing [Tel]. Plus, secure delivery, QoS, etc. could be offered as complementary services for an "integral network-supported e-business solution".

This approach presents also a better scalability with the number of content providers and services offered. Assuming that not every service is needed in every node at the same time, by using programmable nodes it would be possible to obtain a multiplexing gain in the use of the equipment, depending on the load and market situation. As way of example, consider the load variations according to the time of day. It would be impractical to have a static cache distribution in the network, since the requirements regarding which services are needed and where they are needed strongly depend on non-technical factors (an important news update, the release of a successful movie or song in a certain country, etc.) Programmable Networks provide the capability to dynamically relocate functionality on an as-needed basis. In practice, Network-supported Digital Rights Protection represents the establishment of a dynamic overlay controlled by the network. Should there be different overlays for the different providers or services, this scenario is indistinguishable from the classical A&PN proposals, except in one point: The management of the nodes remains under the control of the network operator.

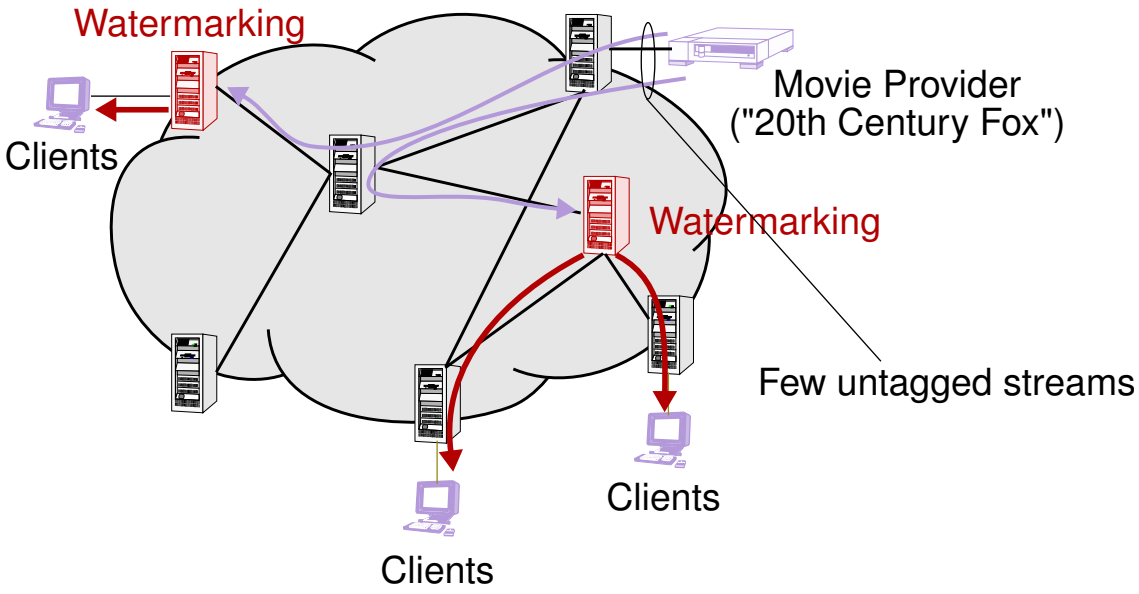


Figure 4.3: Individual Watermarking performed by Programmable Network Nodes.

Network support is nevertheless not mandatory. Content Delivery Networks, e.g., build a parallel server infrastructure, which provides caching and content delivery, using the network exclusively for transport purposes. It would certainly be thinkable, to perform watermarking in those servers. The point here is, however, not that an Active Network approach represents the *only* way of delivering this service, but that it presents an *advantageous* alternative. Furthermore, a CDN could equally profit from a Programmable Network approach, in as far as its ability to dynamically relocate functionality is concerned.

The other parallels drawn in the previous section, concerning the relevance of software and hardware security and management, the necessity of hardware support, etc. equally apply in this scenario.

Summarizing, the goal of the Octopus Network Model is to offer a Programmable Network-like infrastructure to support service models as depicted in these two case studies. It accordingly represents a superset of the functions and mechanisms needed by both scenarios, i.e.: It can be used to support centrally managed services in a small networking environment as well as multiple open network-wide overlays. It pays special attention to the related security and performance issues, while trying to achieve a balance with flexibility and usability.

4.4 Octopus Network Model Architecture

This section presents the *design* of a network architecture, which realizes the set of goals described at the beginning of the chapter. However, this architecture has not been implemented, since the emphasis of this Thesis lies on the resource

management structure needed at the hardware level, one critical aspect of Programmable Networking, which has not been addressed before. Accordingly, not all aspects of the network architecture have been analyzed in the same depth, and some details would only be thoroughly clarified in the context of a complete implementation. The goal, however, is to present the basic structure of such an architecture, including its main elements, and a sound analysis of their feasibility.

4.4.1 ONM Overview

Fig. 4.4 presents the main elements of the Octopus Network Model (ONM). This proposal is *a pure Programmable Network architecture*, i.e., it only allows the downloading of new services from external repositories. Furthermore, the possibility to perform such downloads in real-time when triggered by a (data) packet arrival are considered not viable. The reasons will be explained shortly[2].

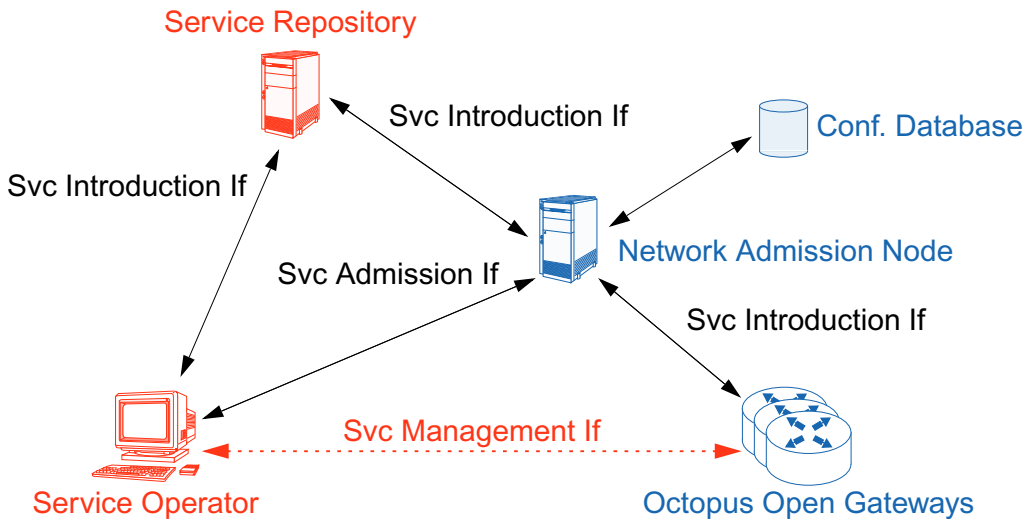


Figure 4.4: Basic Elements of the Octopus Network Model.

The ONM envisions five main architectural elements:

- **Service Operator.** It is the network representative of the firm wishing to deploy a new service. It keeps the implementation of its service (the "code") in a repository under its control, which might be placed outside of the network operator's network (e.g., in a server within the firm).

  The service operator negotiates with the Network Admission Node the conditions upon which its service can be deployed. After the deployment pro-

[2]However, once a new service has been admitted and installed in the network, as will be shown, it enjoys a wide freedom to implement a different behavior. In particular, any single service might decide to use its allocated resources to implement a capsule-like service or to perform packet-triggered downloads. The ONM allows it, as long as this is done within the limits of its Execution Environment.

cess, the service operator manages its service through self-defined interfaces, without network operator's intervention.

- **Service Repository.** Stores the code for all services belonging to one service operator. After the negotiation phase between Network Admission Node and the service operator, the former triggers and controls the code download.

- **Network Admission Node (NAN).** Represents a central element of the architecture, especially for security purposes (admission control). It is responsible for the negotiation of an SLA with the service operator, stating under which conditions a new service can be deployed. Especially relevant for the purposes of this work is the resource usage negotiation (s. Section 4.4.5). It furthermore performs the selection of which nodes will contain the service and the code adaptation to the Execution Environment contained therein. Lastly, it controls the download of the service code into the nodes.

- **Configuration Database.** It contains a copy of the actual configuration of all (active) network nodes. It is managed by the NAN for the purpose of code adaptation.

- **Octopus Open Gateway.** The second main element of the ONM, it presents a software *and hardware* programmable platform for active networking in an integrated active router. It ensures openness, security and performance at the system level.

  It interacts with the NAN for the purpose of service download. After the service introduction phase, the service operator directly manages the services running on every OOG.

In the next section, the interworking of these elements will be clarified. The process from service definition to service deployment and ulterior management will be explained. The rest of the chapter will then review the details of the main architectural elements introduced here.

4.4.2 Service Introduction Process

Fig. 4.5 presents the situation in which a service operator wishes to introduce a new service in a subset of a network operator's nodes. *De facto*, this amounts to instantiating a new overlay network by reallocating part of the network's resources. It should be noticed, that in principle, the whole process is *automated* and takes place through the use of communication protocols and without human

intervention, analogously to RSVP[3]. This happens in five steps (numbered 1–5 in Fig. 4.5):

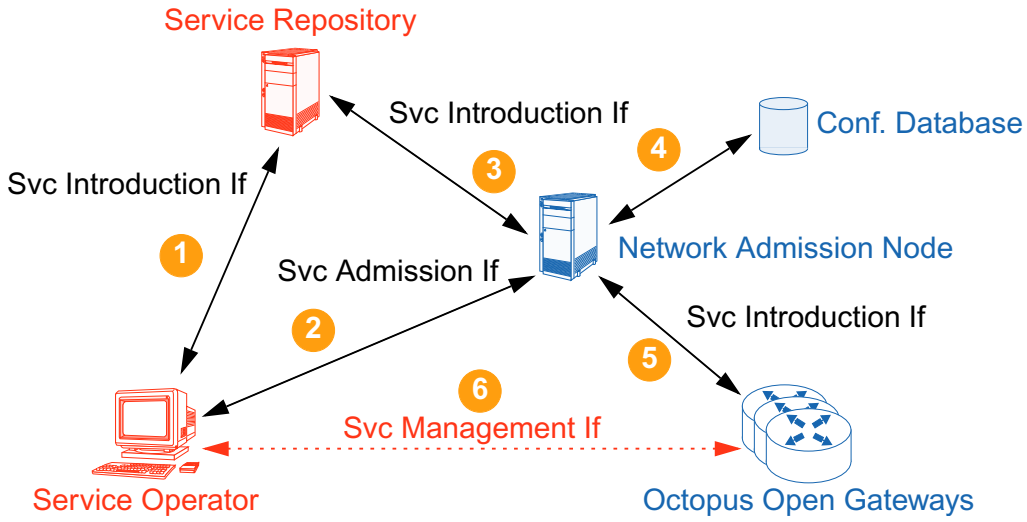


Figure 4.5: Basic Elements of the Octopus Network Model.

1. **Placing Service Code in a Repository.** In a first step, which might happen long before service introduction, the service operator stores the code relative to its new application on a service repository. This is a code server generally located outside of the public network and pertaining to the service operator itself.

2. **Service Admission Control.** In a second step, a negotiation takes place between the service operator and a network representative, known as the Network Admission Node (NAN), through the Service Admission Interface. This negotiation involves, first of all, the mutual authentication of the parties. Additionally, the NAN must authorize the service operator to perform a service introduction. As part of that process, a network resource negotiation takes place. The service operator must communicate which amount of resources, and of which type, it wishes to allocate for its new service. Next, the NAN will check both the availability of resources and if the service operator has the right to claim them (this process will be described in more detail in Section 4.4.5). Should the answer be positive, the third step is started. Otherwise, the admission process is aborted.

3. **Download Service Code to the NAN.** The third step involves the download of the service code from the repository to the NAN, which is triggered

[3]"In principle" because, as is also the case in RSVP, human intervention is not precluded. None the less, since one of the goals is the simplification of the service introduction process, an automated procedure is preferred.

by the service operator. This involves equally a mutual authentication, plus the authorization by the NAN. In order to ensure the integrity of the code, digital signatures are used. Additionally, encryption is used to prevent code theft. The safety of the code can not be proved, since the use of formal methods, including Proof-Carrying Code is not sufficiently advanced.

4. **Code Adaptation.** The fourth step involves the code adaptation to the node environments. First of all, the affected nodes are selected, as will be explained in Section 4.4.5. Next, the code has to be adapted to the node architectures and their actual configuration. The service admission process is critical for the network security, and will be accordingly performed only under NAN control.

5. **Code Dissemination.** Once the code has been compiled, the final step is its downloading to the corresponding nodes, which is performed by the NAN. Besides the code itself, the NAN downloads the resource reservation agreement for the service, so that it can be enforced at the nodes.

   For service admission purposes, the OOGs have no direct relationship with the service operator or its repository, which diminishes risks. Nevertheless, to further improve security, the OOGs build a web of trust with the NANs by interchanging certificates. This also scales better and is less complex than managing a certificate infrastructure among all repositories and nodes. Additionally, during service download, an authentication process and the use of digital signatures takes place among NAN and OOGs to ensure the validity and integrity of the code.

It should be highlighted, that once the service has been installed in the OOGs, no further intervention by the network operator will be needed for its management (step six in Fig. 4.5). The resource negotiation performed in step two together with the usage enforcement and security mechanisms implemented in the OOGs and described in Section 5.1 guarantee compliance. Accordingly, the service operator is left free to implement its own management interface and control and configure its service directly, without network operator intervention.

In summary, this fully automated five step process allows the introduction of new network functionality in a matter of minutes and gives full management control to the service operator, thus greatly improving the flexibility and openness of the network. How to achieve it without sacrificing performance and maintaining security will be addressed in the following sections.

4.4.3 Main Architectural Properties

The Octopus Network Model transcends the openness–security–performance trade-off, which was hampering the advancement of active networking, by means of the following architectural features:

- **Separation between service admission, control and management.** While service admission and resource usage control remain in the hands of the network operator, service management lies unconstrained under service operator control. This is a unique feature of the ONM in the A&PN realm. This decoupling allows for a great deal of service development freedom, without compromising the security of the network.

- **Unrestricted access by the service operator to the programmable hardware.** On the one hand, the introduction of FPGAs allows a much better performance than traditional software proposals. On the other, FPGAs present a very flexible platform to develop customized systems. Since there is no restriction imposed on the service operator through predefined APIs, the development of innovative services is greatly simplified.

- **Resource management in hardware and software at the system level.** By removing security checks from the data plane, performance is boosted. In order not to endanger security herewith, special resource managers are placed both at the software and hardware runtime environments. At the hardware level, this represents a completely new approach to secure active networking. This allows for a total openness and shareability of the node among different service operators in a controlled environment.

- **Separation between active and passive router parts.** As will be seen in section 5.1, the new active services run on an integrated, albeit completely disjoint platform within the router. The goal is, on the one side, to decouple traditional services, under direct control from the network operator, from new services. This also simplifies security enforcement. On the other hand, however, it also makes the architecture compatible with existing routers, which could be upgraded with an Advanced Hardware Platform (AHP, s. Section 5.1) on an as-needed basis.

- **Fully automated service introduction.** This was necessary to increase the usability of the architecture. Furthermore, the Service Admission Interface allows for the negotiation of dynamic SLAs between service operator and a network representative (NAN) without human intervention.

Hence, the ONM and particularly the OOG represent a further step in the development of A&PNs toward designs adequate for deployment in real-life scenarios. These features will be further discussed throughout the rest of the chapter.

4.4.4 The Burden of Service Management

This architecture envisions service providers behaving as true service *operators*. This implies the possibility not only of developing services free of unnecessary constraints (restrictive APIs, fixed tools, limited functionality), but

also directly managing its own services. This might represent too much freedom for some Operators, who like to be spared the burden of dealing with low-level, hardware-close APIs and having to develop their own management software on top of the code itself. Besides, the operation and maintenance of a network service is a complex and expensive task. The ONM allows for intermediate solutions, in which certain companies may develop such building blocks for third-parties and others might outsource the management, exactly in the way in which it happens today. The main difference, though, lies in the intrinsic *possibility* for any company to access the lowest level of network programmability, if ready-made solutions do not fit their needs. In this way, innovation is fostered.

4.4.5 Network Admission Node

The NAN is a critical element in the security architecture of the ONM. It performs three critical functions: Service admission, service adaptation and service dissemination. It is an intermediary for the negotiation of resources and the introduction of network services.

The main rationale behind this network element is the separation of the control and management plane security from data plane security. Security checks have an important cost in terms of processing time and power. In order not to hit performance, this work tries to place most of the security burden on the control and management plane, which has no real-time constraints and changes rarely. In exchange for more "heavyweight" checks in this plane, the requirements on the data plane (i.e., those which have to be performed for every packet or flow) are simplified. That is the reason why the Octopus Network Model can not support a capsule-based approach, since the performance cost on the data plane would make it impractical for real networks.

4.4.5.1 Service Admission

Beyond taking most of the burden of security control away from the data plane, the service admission process serves another purpose: The automated negotiation of the technical and administrative conditions, under which a new service can be introduced in the network.

Accountability and predictability are key concerns when introducing new functionality in a network. To that end, the ONM foresees the negotiation of a Service Level Agreement (SLA) between service operator and network operator prior to service introduction. The time scale is thus in the order of weeks or months. That SLA is valid for the whole life of the service, except if a renegotiation takes place. A reason could be an increase in traffic volume addressed to the service, which would request more resources, or the upgrade of the service code.

Such an SLA would include an administrative part (which will not be considered here) as well as a pure technical part. Especially relevant in the latter is the

characterization of how many resources and of which type will be needed by the service.

In this context, it is important to ask which resources are considered and how can they be described. The network resources considered here and included in the SLA are basically of three types: Link capacity (bandwidth), storage space of different kind (e.g. SDRAM for applications, but also FPGA logic cells to install services) and processing power ("CPU cycles"). Additionally, the traffic behavior should also be included (i.e., a description equivalent to the TSpec in IntServ or to the ATM traffic contract). In the SLA, the resource amount can be expressed as an absolute value ("the service needs 10 Mbps and 50 MB") or as a range of acceptable values ("either 10 Mbps and 50MB or 5 Mbps and 100 MB").

However, providing an absolute reference for these resources is a complex matter, since the "value" of a unit is technology and implementation dependent. I.e., a Pentium II "cycle" does not allow the same computational power as a Pentium IV "cycle". And the number of cycles needed by an application depends on the processor architecture as well as other factors. Furthermore, how to compute the amount of resources needed by an application in the general case is also a difficult task. In both cases, benchmarks exist, but an absolute solution is difficult to achieve. Nevertheless, this will not be dealt with in this Thesis. The reader is referred to the literature for proposed approaches [CCF+01], [ABC+03], [WPF03].

The NAN is in charge of checking the availability of the resources needed by the service. To that end, the NAN keeps a database with the actual configuration and resource usage of every node under its supervision. Since the NAN is only concerned with services and not with individual flows, it is the *aggregated volume of reserved resources* which is of main interest, not the instantaneous load. This information is then used to calculate which nodes should be selected to support the new service, according to their actual load and other constraints, like geographical location. With this information, the SLA can be signed.

Since network resources are scarce, choosing a constellation of nodes to install a new service involves an optimization problem. In most cases, this multidimensional optimization is NP-hard. It follows that heuristics have to be used. Transformations, meaning trading some amount of a resource for another, in the expectation, that the overall performance will be kept constant, are in principle possible, provided that the service behavior is known with enough detail. Again, the reader is referred to the relevant literature for additional information [CCF+01].

The SLA has to be enforced. To that end, the resource usage agreement is downloaded together with the service code, to every individual node, so that the NodeOS, for the software part, and the Embedded Hardware Manager, for the hardware part, can re-schedule their resource sharing policy. Both perform access control and resource management in the data plane in their respective domains (see Chapter 6).

4.4.5.2 Service Adaptation

Once a service has been admitted for introduction and a set of nodes have been chosen to implement it, code adaptation has to take place. This implies:

- Adapting the software code to the runtime environment. This can be obviated for platform-independent code, like Java. Nevertheless, in networking environments most code is optimized for the precise platform in which it will run, in order to obtain maximum performance. At a minimum, the code has then to be compiled for the new environment.

- Integrating the hardware code into the FPGA configuration. Unlike software systems, FPGAs have to be reprogrammed whenever a change in functionality is necessary. Nowadays, however, partial reconfiguration without interrupting running services is possible.

  Still, integrating new code in a partially full FPGA involves synthesizing the new code for that particular FPGA technology and placing and routing it so that it fits with the modules already present. This is a long and computing-intensive task.

- Reconfiguring the EHM and the NodeOS. As explained before, the new resource reservations have to be included in both resource managers. In the case of the NodeOS, this only implies a few system calls. For the EHM, which itself is a hardware module, a number of configuration registers have to be updated.

In principle, this adaptation could be performed by the service operator or by every individual node, instead of by the NAN. This would present a number of disadvantages, however: The service operator generally lacks the information concerning the exact configuration and equipment of every network operator's node. Additionally, the configuration of NodeOS and EHM being crucial for SLA enforcement, can not be left in the hands of the service operator.

Compiling or otherwise adapting code to an specific platform (e.g., performing the placing and routing for HDLs) is a relatively long and computing-intensive process. Network nodes are dimensioned to work under heavy loads and have thus not enough spare capacity to regularly perform such tasks without impacting their throughput.

Hence, the NAN is best placed to realize the adaptation, with the help of the configuration database introduced before.

In principle, code portability among platforms is a desirable feature, which has been often targeted by A&PN researchers. For the ONM, this is not deemed possible. In this architecture, the active network nodes, known as Octopus Open Gateways (OOG) present a programmable platform for software as well as hardware applications. As a consequence, the service code will include a combination

of software and hardware description languages (HDLs). While code portability is possible (with some non-negligible restrictions) for software applications, it is Utopian for programmable hardware. Since hardware designs are strongly dependent on the characteristics of the underlying platform, for which they are optimized, no portability beyond the behavioral level is possible. Furthermore, even in the software case, performance-critical applications are usually coded on a machine-dependent fashion at a very low level, commonly in C or even assembler. Under these circumstances, no completely platform-independent form of code can be directly downloaded from the repository to the network nodes.

4.4.5.3 Service Dissemination

The last step on NAN functionality is the download of service code and resource usage specification to the participating nodes. The Darwin project, introduced in Annex A, presents a suitable platform for the realization of this and other NAN tasks. It represents a starting point for the ONM software architecture. In this context, the Beagle protocol could serve the purpose of signaling and downloading protocol for code dissemination.

At the system level, code dissemination implies the reconfiguration of NodeOS and EHM, as has been explained before. Both tasks are taken up by the Service Introduction & Interaction Manager (SIIM), part of the Octopus Open Gateway architecture, and which will be introduced in section 5.4. Once this task is finished, the service can be activated and start operation.

In summary, a full automation of the whole process (negotiation, polling of node resource availability, optimization, code adaptation, dissemination and activation) is possible without disrupting node activity, as has been shown, e.g. in [LMK+03].

It remains to be said, that because of its critical role in the architecture, the NAN could represent a single point-of-failure and scalability bottleneck. Nevertheless, the NAN is only needed at service introduction, which is a comparatively rare event. Hence, scalability and performance is only a secondary problem for the NAN. A single NAN could very well serve as point of entry for a whole network, thus avoiding the coordination problems associated with distributed systems, as exemplified by the work on Bandwidth Brokers [NJZ99]. None the less, to prevent failures, a set of stand-by NANs should be in place to substitute the active NAN in case of malfunction.

This chapter has presented the fundamental elements of the Octopus Network Model, as well as its architecture. Within the ONM, the Octopus Open Gateway plays a predominant role, as the programmable software and hardware platform for value-added services. The next chapter is consequently devoted to its architecture and characteristics.

5 Octopus Open Gateway Architecture

The OOG is the cornerstone of the ONM, together with the NAN. This chapter is devoted to its architecture, which realizes the principles of openness, security and performance at the system level.

After an overview of the OOG characteristics, the description of the node and its different elements will be given: Line Card, AHP, AHPM and the interconnection thereof and with the BHP and CPU. A discussion of the OOG's impact on service architectures and the necessity of resource management at the AHPM level conclude the chapter.

5.1 OOG Overview

The OOG represents this Thesis' vision of an Open Active Node. Open, because it concedes great freedom to the service operator to design and manage its service instances without network operator interference. Active, because the introduction of value-added services in the nodes is allowed. Unlike most of its predecessors, it introduces programmability at the software *and hardware* level, in order to achieve both flexibility and performance. In this sense, it expands the classical understanding of an active node architecture with an additional application layer underneath the NodeOS (s. Fig. 5.1). Additionally, a Service Introduction & Interaction Manager (SIIM) module has been introduced, to explicitly deal with service instantiation and control, as explained below.

A representation of the OOG's basic architecture is depicted in Fig. 5.2. It is divided in three main modules: A Basic Hardware Platform (BHP), an Advanced Hardware Platform (AHP) and a CPU. The BHP implements the basic packet forwarding and routing functionality. In itself, it can be regarded as a common router. The AHP is the software and hardware programmable platform available to the application developer. Both are interconnected by a set of interfaces (s. Section 5.3). The CPU represents an additional software environment for applications, but it also integrates the SIIM functionality. Besides, it hosts the Routing and Management Unit (s. Section 2.2).

The OOG is conceived as an integrated node. The reason lies in the necessary interaction between some BHP functions, like routing table management and

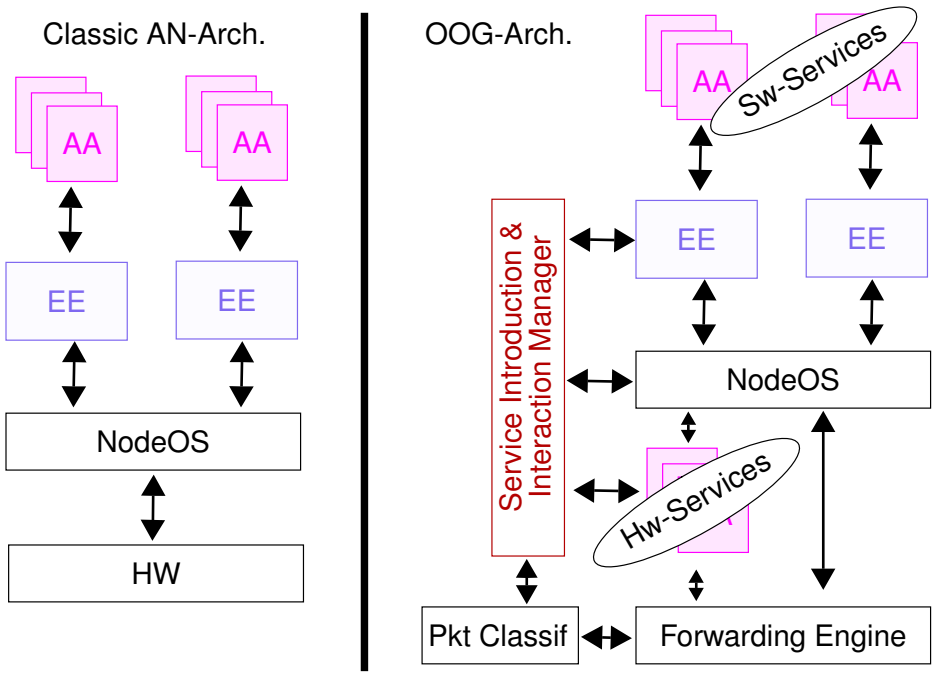


Figure 5.1: Enhanced Active Network Node: The Octopus Open Gateway.

packet classification, the SIIM and the AHP. Nevertheless, from a logical point of view, AHP and BHP are separated instances. If they are interconnected by an internal backplane or by a network connection is, in principle, an implementation detail without a big impact on the network architecture as a whole. It is none the less necessary, that the interconnection between BHP and AHP may take place at different points along the forwarding process. Value-added services do not necessarily take place always on the same processing stage: E.g., encryption is typically the last step before packet transmission, while decryption is the first. This represents another reason for the co-location of AHP and BHP, although they might be implemented in two separate devices.

The advantages of keeping the AHP and the BHP logically and architecturally separated are threefold:

- Security. The BHP realizes the fundamental functions of a communication node, i.e., packet forwarding and routing. This functions are necessary for *every* packet passing through the node, no matter which other service or destination it might be addressed to. Accordingly, the BHP will stay under the exclusive control of the network operator. For security purposes, a clear-cut interface toward the service operator-controlled AHP is a great help. The possibility to deny access to the BHP in case of malfunction is herewith granted to the network operator.

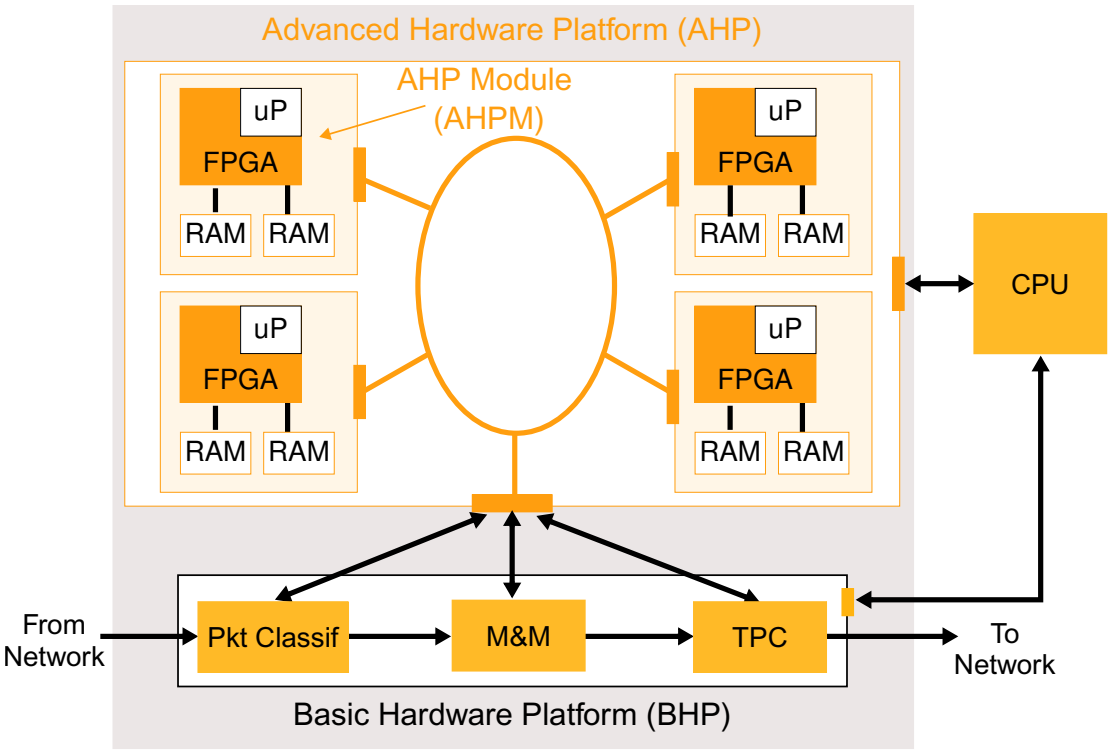


Figure 5.2: The OOG Basic Hardware Architecture.

- Extendability. Including a set of AHPs in a router implies an additional (costly) investment. In order to facilitate its upgradability (and hence its useful life), a modular design is to be favored. Furthermore, a modular design allows it to scale better with traffic demands. A complete separation between AHP and BHP, furthermore, allows the replacement of elements of both parts without disrupting the other.

- Retro-compatibility. Active routers will not be deployed worldwide at once. Hence, a migration scenario is necessary. By allowing the possibility of adding AHPs as separated modules in selected existing routers, a transition path is opened. By keeping interaction between BHP and AHP at a minimum, retro-compatibility is simplified.

The AHP is composed of AHP Modules (AHPMs) interconnected by an internal ring. Every AHPM disposes of an FPGA, with the possibility of integrated embedded microprocessors, some memory, and interfaces to the rest of the node. This modular approach allows to keep the node scalable with the number of services as well as with the processing power needed.

It is basically in these AHPMs, that data plane applications will be realized: The microprocessors provide a software development platform, while the FPGAs provide the hardware programmable environment. Their tight coupling allows a

reduced intercommunication overhead for both parts of a service. Management plane applications (or parts thereof) run on the additional CPU.

As stated before, the ONM sets a high importance on security. To this end, applications running on the CPU (by definition, not time-critical), are only allowed to run inside predefined Virtual Machines (VMs). The Virtual Machine performs resource management tasks for the applications which run on it. Typically, applications belonging to the same service operator will share one. Within a VM, a service operator is free to distribute its allocated resources as it pleases. In order to avoid inter-VM interference, the NodeOS enforces the resource usage associated with every VM. In this way, the network operator keeps control of the overall node resource management, while allowing individual usage policies for the service operators sharing it.

On the AHPMs, though, such OS control for software applications is not possible, since there rarely is an embedded OS present. The overhead associated with several software abstraction layers would tremendously degrade the microprocessor's performance. Because of this, it is not possible to safely share an embedded microprocessor among several service operators. However, it is perfectly feasible to share it among services belonging to the same operator, since only his applications would be affected in case of malfunction.

In addition to microprocessor code, applications will also be realized in the form of native hardware modules (typically as Verilog or VHDL code). These modules will also be synthesized and placed on the FPGA, sharing resources with the microprocessor and its applications.

In order for this to work, the equivalent of a Virtual Machine or sandbox at the AHPM level is necessary. Such a hardware sandbox would realize resource management functions directly in hardware, so that the microprocessor application modules would not interfere with hardware modules running on the FPGA or with other AHPMs. *Vice versa*, it must also be guaranteed, that hardware modules running on the FPGA will interfere neither with each other nor with the microprocessor, nor with the rest of the AHP.

This interference by hardware and software modules sharing an AHPM can happen at several levels: First, direct interaction, through attempts to intercept another module's interfaces. Second, an indirect, and by far easier and equally effective way of interfering relies on access to shared off-chip resources, like memory banks or intercommunication channels to the rest of the platform. These could take the form of denial-of-service attacks (blocking access to a certain resource by other modules) or passive attacks (accessing memory ranges allocated to other applications, thus getting information on them).

As noted in the Introduction, growing IC density allows to place whole systems on a single FPGA, as well as multiplexing hardware modules corresponding to different applications on the same chip. The number of I/O pins per FPGA, which allow to bring data in and take data out of the chip, can by far not grow that fast, for technological reasons. Furthermore, there is a multiplexing gain in

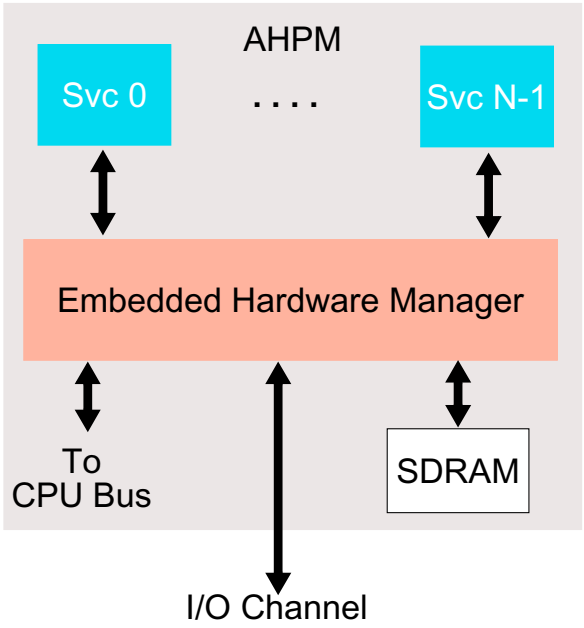


Figure 5.3: AHPM Functional Representation.

pooling certain resources, like memory, together in bigger modules rather than providing an individual smaller module per application. The reasons comprise performance, overall size requirements, power consumption and, as a whole, cost. Hence, in the midterm applications running on a FPGA will have to share a common set of off-chip resources. In this context, some form of regulation has to be implemented in the access to those resources.

As a consequence, the hardware sandbox mentioned before has a double role to accomplish: On the one hand, it must control direct interactions among application modules within the FPGA (including the microprocessors). On the other, it must regulate access to shared off-chip resources (s. Fig. 5.3). Both of these tasks are taken up by the **Embedded Hardware Manager** (EHM) described in the next chapter, and which constitutes the first attempt at coordinated resource management for open and programmable hardware systems, that the author is aware of.

5.2 Integrated Active Router Architectures: 2.5G vs 3G

As explained in section 2.3, the trend in router design is toward a fully distributed system in the data plane, with some central intelligence for control and management plane tasks. The main rationale behind it is to increase data plane throughput with respect to older, more centralized schemes. The idea behind an active router is to perform an additional set of value-added functions in the data

and control planes. Hence, the necessity of ensuring high performance is even more stringent than on common routers.

Nevertheless, two different architectures can be envisioned, depending on the role played by the active router (corporate edge, ISP edge or core, s. Section 2.4) and the amount of traffic requiring "active" treatment.

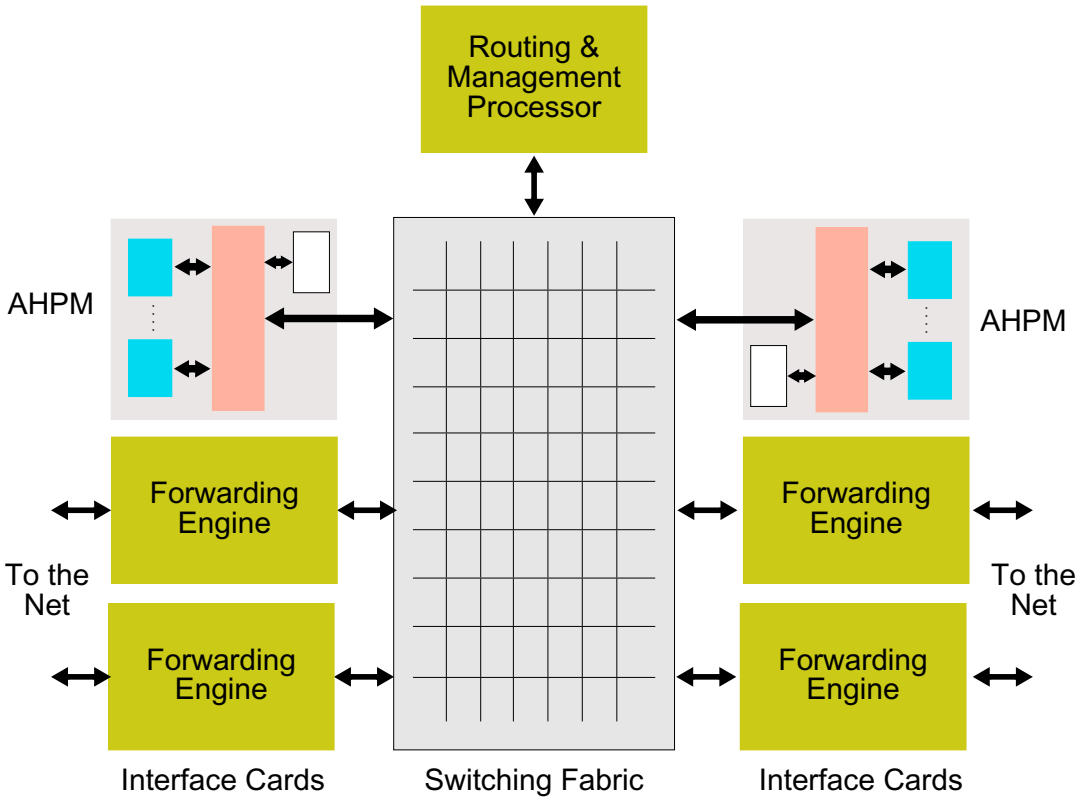


Figure 5.4: OOG Architecture. First Option.

Fig. 5.4 depicts a design based on a mixture of the principles of second and third generation routers, hence the naming "2.5G router". On the one side, it presents a set of "traditional" 3G line cards, with fully distributed packet forwarding functionality, interconnected by a parallel backplane for increased internal throughput. On the other, it envisions a common pool of AHPMs (the "active" part), analogous to 2G forwarding engines for additional functionality. These AHPMs dispose of an internal data packet cache, so that packets sent to them for processing do not have to be written back to line card memory.

The main disadvantage of this design lies in the repeated traversal of the backplane for active packets. At a minimum, a packet requiring active treatment has to traverse the backplane twice: Once to reach the AHPM, and a second time to be sent to the destination line card. However, the possibility of "chains of services" (i.e., packets that have to be processed by several services consecutively, like watermarking plus encryption) implies, that packets might have to

wander from AHPM to AHPM several times, before reaching the output line card. Hence, the aggregated backplane throughput needs to be strongly larger than the aggregated line card speed, or only a small fraction of the traffic could be served by the AHPMs.

The advantage, nevertheless, lies in its architectural simplicity, its lower cost and its compatibility with existing systems: This design could be a first step in the transition toward deploying active routers, since it fits well in existing 3G designs, which present specialized line cards for specific operations, on top of forwarding ones [Cisb]. AHPMs would then appear as "normal" cards for existing routers. With only a moderate software update in the Management Unit, the basic advantages of A&PNs could be realized, albeit with some restrictions. Basically, the interaction between AHPMs and BHP would have to be kept to a minimum for lack of adequate interfaces, and probably would have to be delegated to the Management Unit.

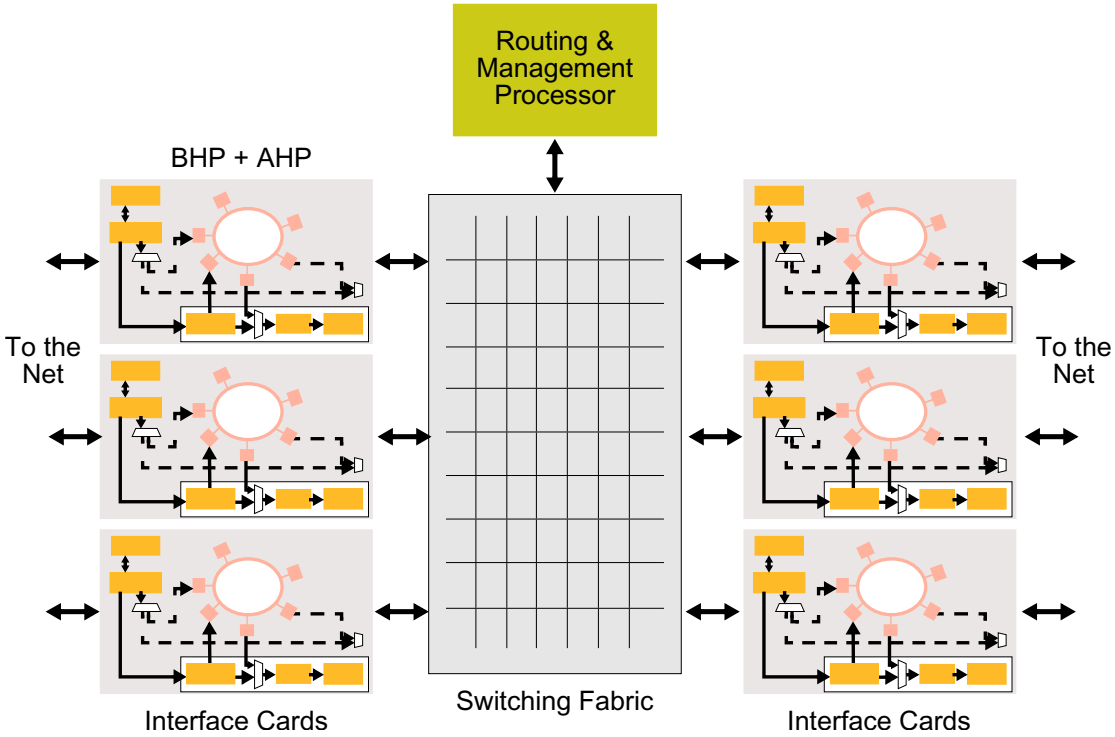


Figure 5.5: OOG Architecture. Second Option.

A fully integrated solution is presented in Fig. 5.5. In this case, the AHP is integrated in every line card, although it remains a separate unit inside of it (represented by a ring in the figure). Active processing is realized inside of every line card. Hence, packets have to be transported only across the ring in case of processing by multiple AHPMs. Full active processing is possible, without hitting backplane throughput. Furthermore, the communication with the BHP is a lot

easier, for a direct set of interfaces have been envisioned for it (s. Section 5.3). Fig. 5.5 represents the logical evolution toward active 3G routers: The throughput and scalability of the 3G design is far superior than in the 2.5G case, but so is also the cost. Edge routers in big networks will probably necessitate the former, while lower speed networks or transitional semi-active routers will fall back on the latter.

5.3 3G Line Card Architecture

The AHPM design presented in section 5.3.3 fits equally well in both designs, i.e., it can be implemented as a stand-alone 2.5G line card or as a module inside of a 3G interface card. The main difference lies only in a modification of its interfaces, as will be explained shortly. Hence, this section will be centered on the 3G line card architecture, which represents the next step in router evolution.

5.3.1 3G Line Card Functional Description

A functional representation of the line card design is presented in Figs. 5.7 and 5.8. It has been divided in two for clarity, but both modules together compose a single card, as depicted in Fig. 5.6. Accordingly, fig. 5.7 contains the elements involved in pre-switching processing, i.e., before an incoming packet is sent across the internal backplane to its outgoing interface card (marked as "input" in fig. 5.6). Fig. 5.8, which has an analogous structure, contains the post-switching modules (marked as "output" in fig. 5.6). An example of pre-switching processing is the decryption of an incoming data stream, previous to its routing, watermarking, etc. A post-switching example is the encryption performed on an outgoing packet, after having been routed, its TTL decremented, etc.

The card is composed of the following main elements:

- Packet Buffer and associated Memory Controller. Incoming packets are stored in the Packet Buffer and kept there until a switching decision has been made. In a router, typically only control information and the packet header are passed to the different modules for processing.

  The Memory Controller regulates access to the Packet Buffer for all other card components.

- Advanced Hardware Platform (AHP). Presented as a ring, it contains the AHPMs, which will perform the value-added packet processing. Besides the AHPMs, the ring contains a number of additional stations, which perform the role of interfaces to the BHP elements as well as to the Packet Buffer.

- Basic Hardware Platform (BHP). It is composed of the main functional blocks of any router architecture, as presented in section 2.2. Together, the BHP elements realize a traditional stand-alone router.

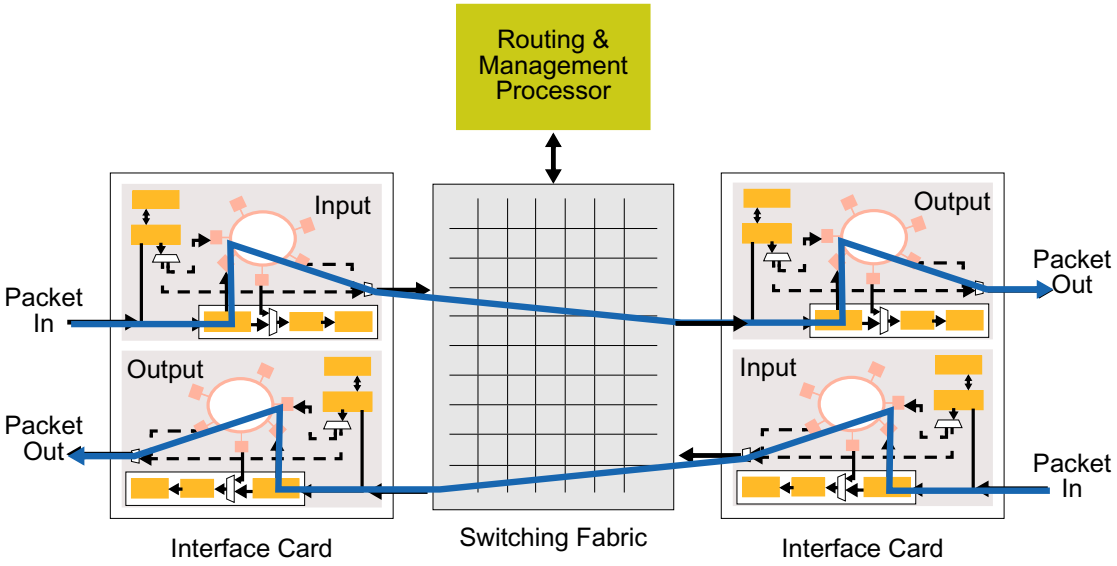


Figure 5.6: Decomposition of a Line Card in an Input and an Output Module.

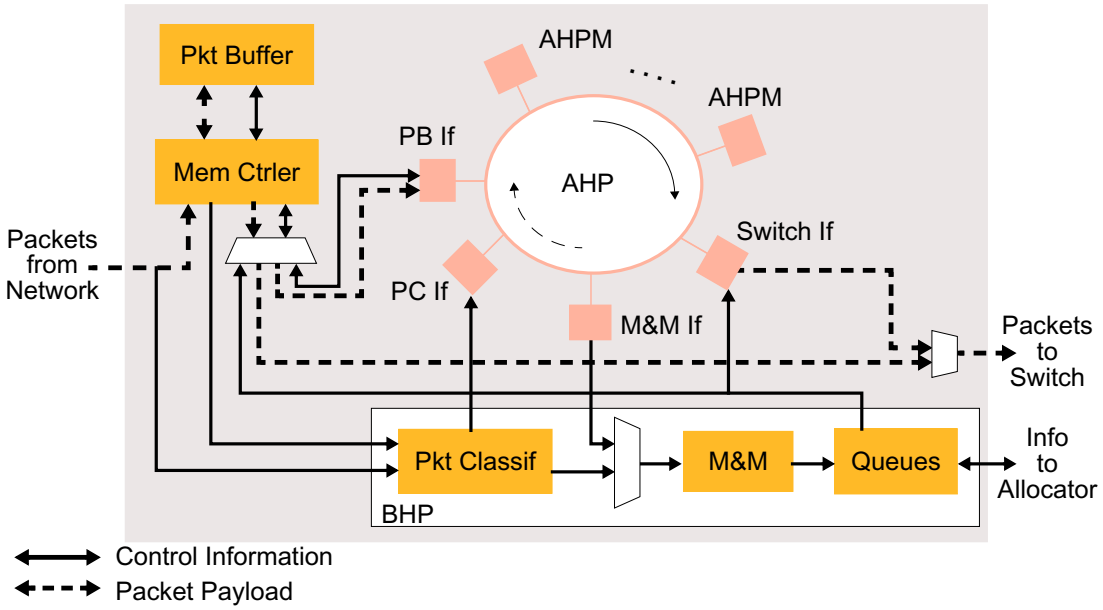


Figure 5.7: OOG Line Card Architecture. Pre-Switching Part.

To better illustrate its characteristics, the path of an incoming packet across the line card will be explained.

Upon arrival at the input interface, the packet is stored in the Packet Buffer. Additionally, a copy of the packet header is sent to the Packet Classifier. Subsequently, the Packet Buffer sends a reference of the packet storage location to the Packet Classifier, so that it can be retrieved after processing.

Traditional routing decisions affect only the packet header, especially its destination address (for route lookup), TTL (decrement) and CRC (integrity check). Even more complex functions, like packet classification or firewalling, generally involve only information contained in the layers 3 (IP) and 4 (TCP/UDP) header fields. Hence, to improve performance by reducing the amount of information to be transported from module to module, only the IP and TCP/UDP headers, plus some extra control information, will be used in most router modules.

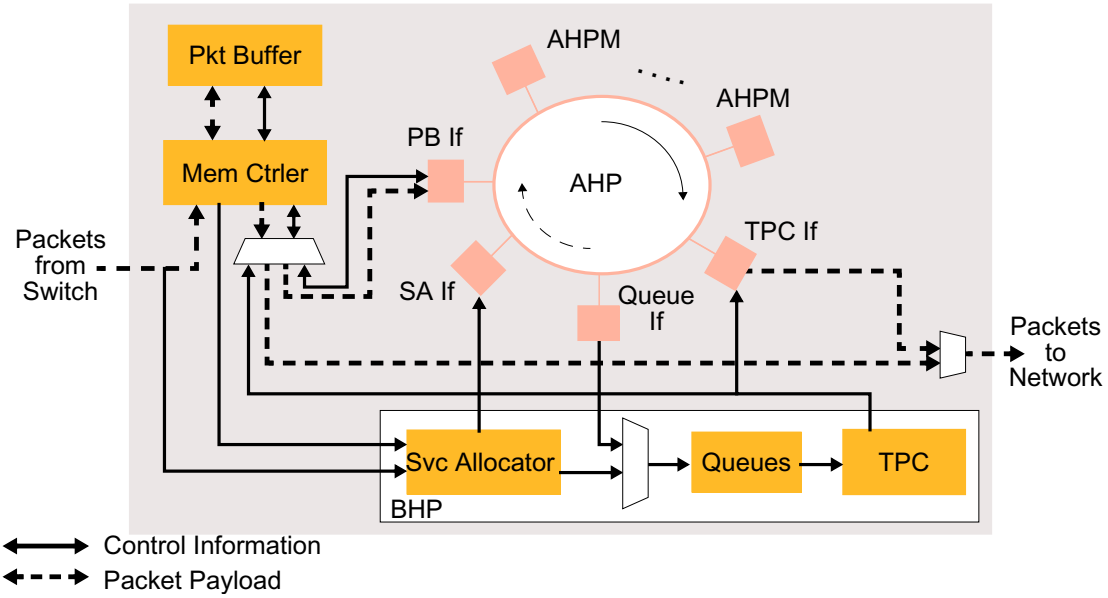


Figure 5.8: OOG Line Card Architecture. Post-Switching Part.

It is the role of the SIIM (s. Section 5.4) to establish the order in which packets will be sent to the different services by which they have to be processed. These "chains of services" are associated to the corresponding classification rule in the Packet Classifier and contain:

- A list of the services, that shall process the packet. This is given in the form of a hierarchy of addresses: The card in which the service resides, the AHPM address within the card, and the service identifier within the AHPM.
- An indication, whether a service in the chain also performs processing on the packet payload or not. This information is passed to the SIIM by the NAN during the service introduction phase. This serves to accelerate and facilitate packet retrieval in the AHP.

When the packet header reaches the Packet Classifier, two possibilities arise: If it was a packet, which does not need processing in the AHP, its service chain only contains the address of the output interface. It is then directly passed to

the Meter & Marker (M&M) and eventually to the Input Queues. Otherwise, the Packet Classifier passes the header, the service chain and the pointer to the payload position in the Packet Buffer to the first AHPM in the list. If that AHPM lies in the same card, the information is passed via the Packet Classifier ring interface (PC If). If it resides in a different card, it is passed to the M&M.

In the ring, the AHPM containing the first service in the chain receives the header, service chain and pointer from the Packet Classifier. If additionally the payload is required for processing, it is retrieved from the Packet Buffer via the Packet Buffer ring interface (PB If) and cached on the AHPM. The pointer is accordingly updated to signal the new packet location.

After processing, the control information is sent to the next service in the chain. If it happens to be on the same card, it is sent across the ring, together with the payload, if needed ("push" operation). If not, the control information is sent to the M&M via the M&M ring interface (M&M If).

After arrival of the control information to the Input Queues, the whole packet is retrieved and switched. The updated pointer indicates if it was still stored in the Packet Buffer or in some AHPM. In the second case, the packet is directly retrieved from the AHP via the Switch ring interface.

On the post-switching side, the operation is analogous: The complete packet is again stored in a Packet Buffer and header, service chain and the updated pointer are passed to the Service Allocator.

The Service Allocator is a simple demultiplexer: If AHP processing is necessary, it passes the control information to the AHP via the Service Allocator ring interface (SA If), analogous to the Packet Classifier. From this point, the procedure is identical to the pre-switching case, with one exception: The Traffic Pattern Conditioner (TPC) takes the final scheduling decision, retrieves the packet and sends it to the network.

The architecture presented here keeps a clear-cut separation between AHP and BHP, which facilitates ONM security policy implementation. Besides, the interaction between them is reduced to a minimum set of clearly defined interfaces.

As is typical for modern router architectures, it increases performance by working mainly with headers and storing packets in a separate buffer. Only if active applications require the packet payload will it be retrieved from memory.

In the following section, the AHP ring, which interconnects all AHPMs in a card, will be described. Special attention will be paid to the mechanisms introduced to improve performance.

5.3.2 AHP Ring

Processing elements in a router typically exchange packet headers. AHPMs, additionally, also have to interchange packet payloads for more complex applications. Additionally, AHPMs have to retrieve and send data to the BHP at

different points in the processing chain. Hence, a high throughput is necessary in this scenario[1].

Having an AHP per card, complex and costly interconnection structures should be avoided, for scalability reasons. Besides, due to the small physical scale, mechanisms with large delays in access to the medium (like some token-passing rings or shared buses) are also impractical.

The final choice for the AHP fell on a slotted ring, after comparing three different options: Bus, crossbar and ring. In summary, the advantages of rings against its competitors in this scenario are:

- Like crossbars, slotted rings allow parallel transmission of several pieces of information. This is a performance advantage vs buses.
- Although crossbars can have even better scalability and performance than rings, they also present a much higher complexity and increased resource consumption. This derives from the control elements needed for their operation (allocators, etc.)
- For small distances and slots of short duration, as is here the case, rings present a small delay overhead, especially if explicit tokens are avoided.

Furthermore, rings are very efficient (i.e., they have a very high utilization), since they avoid collisions in access to the medium. This derives in high throughput. All in all, a slotted ring presents a good compromise between high throughput, simplicity, moderate resource consumption, small delay, and high efficiency.

The ring is divided in slots of constant duration and equivalent to 64 byte. This size allows for the transmission of a typical packet header (roughly 40 byte), plus the service chain and a pointer to the packet storage location[2]. Thus, in most cases AHPMs can interchange all relevant information in only one slot time.

To ensure fairness, one slot is reserved for every station in the ring. To that end, every station lets N-1 free slots pass by before occupying one itself, N being the total number of stations in the ring. The slots are freed upon reception of the information ("early slot release"), which allows for slot reuse and increases efficiency.

However, packets take much longer to send than control information. A typical 1500 byte packet needs roughly 24 slots to be transported across the ring,

[1] As way of example, assume an aggregated card bandwidth of 10 Gbps. If 10% of packets require AHP processing, of which 50% need the payload, the data volume entering the ring yields 500 Mbps. Nevertheless, these orders of magnitude do not present any kind of problem for current technology.

[2] A simple calculation suffices to show this: A complete ACK TCP packet is 40 byte in size. Plus, 32 bit to transport the storage pointer. This leaves 20 byte for the service chain. Accord 31 bit to code every service identifier (maybe 15 bit to code the card – clearly too much –, 8 to identify an AHPM in the card and another 8 to identify the service inside of the AHPM), plus 1 bit to signal if the payload is needed. This gives a total of 5 elements per service chain *per slot*.

so it would take 24*N slot times. To increase the packet transport speed in the ring, the total amount of slots is bigger than the amount of stations. The extra slots are reserved for packet transmission and regulated by a Ring Master (s. Section 5.3.3).

Whenever an AHPM needs to send a whole packet to another station, it requests the extra slots from the Ring Master via an interrupt line. Upon receiving the grant, it can use the extra slots consecutively (i.e., in a burst) on top of its reserved one. Assuming a mere 4 slots reserved for the purpose, a whole packet could be transported in 6 bursts or 6*N slot times[3].

The choice of two slot types is a compromise between fast packet transmission and long waiting times for control information exchange.

By reserving one slot per AHPM, a roughly equal load per station is assumed. Certainly, this does not have to be the case. A way of overcoming large deviations of this assumption, is to periodically evaluate the AHPM load and rearrange the services therein to optimize the traffic distribution. This task could be performed by the SIIM on a daily or even weekly basis.

5.3.3 AHPM Functional Description

The AHPM represents the software and hardware programmable platform put at the disposal of the service operator. Physically, it is composed of a FPGA with embedded microprocessors, some external memory and interfaces to the ring, the CPU (s. Section 5.3.4) and the external memory. The FPGA provides the programmable hardware environment, while the embedded microprocessors provide the software environment. Additional, non-time-critical applications run inside Virtual Machines in the CPU.

The AHPM is functionally divided in three blocks (s. Fig. 5.9):

- Ring Attachment Subsystem: Composed of a Slot Manager, input and output queues, a delay register and a multiplexer. Its function is to regulate access to the ring.

- Service Chain Management Subsystem. Composed of the Packet Cache and its Manager, the Service Chain Manager and the Service Chain Table. It manages the service chains corresponding to packets being processed in this AHPM, as well as the passing and retrieving of control information and payload associated to them.

[3]Again as way of example, assume a ring with 6 AHPMs and 4 interfaces to the BHP, plus 4 reserved slots for packet transmission. Hence, N=14. For a 32-bit ring at 200 MHz, the time needed to transmit a whole packet is: $T = 6 * N * t_{slot} = 6 * 14 * 40ns = 3.36\mu sec$. On an interchange between Stuttgart and Berlin, taking roughly 10 ms and traversing some 50 routers (all of them active), this implies an overhead of 1,68% in total delay.

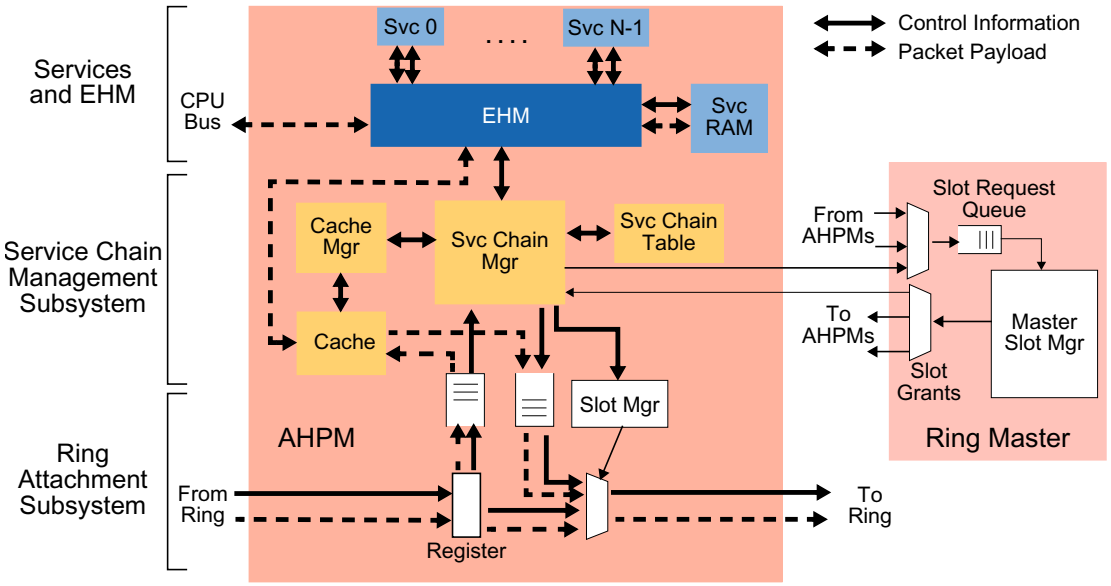


Figure 5.9: AHPM Functional Representation and Ring Master Structure.

- Services and EHM. The services are introduced by the service operator and are in charge of processing the packets. Their functions can be very diverse: Encryption, watermarking, transcoding, intrusion detection, etc.

  The EHM regulates the access of the services to the ring, the CPU, the off-chip memory, the packets and any other shared resources present in the system. It is the critical element to enforce the security and QoS policy of the network operator at the node level.

Additionally, a Ring Master is present to schedule access to the extra slots reserved for packet interchange. Because of its simplicity, it can be integrated with one of the AHPMs or be present as a separated module[4].

In the following paragraphs, the operation of these blocks will be further explained.

5.3.3.1 Ring Attachment Subsystem

This block manages the AHPM attachment to the ring. Basically, incoming data are delayed in a register to check if the slot was free or busy, and, in the second case, if the information transported therein was addressed to this AHPM. As explained before, addressing is coded in every service chain element as a 3-tuple (card, AHPM, service). The Ring Attachment Subsystem checks only the

[4]This second option would be more appropriate, especially if additional functionality is placed on the Ring Master. The introduction of QoS in the ring to give precedence to some packets or stations over others would be a case in point. Such extensions have not been included in this work.

address contained in the *first* element of the service chain, since the ordering in the chain is also the processing order by the services.

Should the slot be free, and assuming that this AHPM has data to send, two possibilities arise: If the Slot Manager signals, that this is "his" assigned slot, information will be retrieved from the output queue and sent. Otherwise, the slot will be passed to the next station unaltered.

The Slot Manager is basically a counter. It lets pass by N-1 free slots before granting access to the ring, in order to ensure fairness. If additional slots have been assigned by the Ring Master to send a packet, the counter is updated accordingly.

If the slot was not free (recognizable either by a "free bit" at the beginning of the slot or by a non-empty service chain) or if this AHPM had no information to send, the slot will be again forwarded unaltered.

If the information contained in this busy slot was addressed to this AHPM, its content is copied into the input queue and the slot is freed before passing it over. In this way, it can be reused by the next station and higher efficiency is possible.

5.3.3.2 Service Chain Management Subsystem

The main element in this block is the Service Chain Manager. It keeps a table containing all service chains corresponding to packets being processed in this AHPM. Additionally, it is charged with retrieving and passing control information and payload associated to those packets.

When control information concerning a new packet arrives, the Service Chain Manager stores it in the Service Chain Table. If payload was not needed for processing, it passes the packet header to the EHM. Otherwise, two possibilities arise: If the payload is still in the Packet Buffer, the Service Chain Manager sends a request to the Packet Buffer Interface and waits. If the packet had already been cached at some AHPM in this card, it will come shortly after the control information, by using the packet transmission slots. No request is necessary in this more common case, hence reducing delay.

Upon payload arrival, the Service Chain Manager directs it to the Cache Manager, which stores the payload in the local Cache. The Cache is built of on-chip SRAM modules, much faster than off-chip SDRAM[5]. Hence, passing packets among AHPMs is much faster than retrieving them from the Packet Buffer. Additionally, memory interfaces are a common performance bottleneck in modern communication nodes (s. Section 2.5.3). Writing packets back to the Packet Buffer whenever a service has finished processing would greatly strain the memory interface. With local caches, the load is distributed and performance

[5]Modern FPGAs include a large amount of on-chip RAM modules (e.g., several Mbit in an Altera's STRATIX II device [Cor]). Hence, storing a few packets on-chip presents no technological challenge nowadays.

increased. These are the main reasons for introducing a local cache in every AHPM.

The Cache Manager basically keeps track of the storage location of all packets in the Cache and passes this information to the Service Chain Manager, in order to update the control information in the Service Chain Table. It also passes the payload to the EHM and retrieves it from there, under Service Chain Manager supervision.

Whenever the service is finished with the processing, the Service Chain Manager receives the (maybe altered) header and updates the control information: It writes in the new header, removes the first element in the service chain (already processed) and, if the payload had been retrieved from memory or some other AHPM, it updates the storage location with a pointer to its local cache. The payload is written back into the Cache.

If more than one service shall process this packet, several possibilities arise:

- Next destination is also in this AHPM. In this case, the header (and the payload, if needed) is passed to the EHM with a new service identifier, exactly as before.

- Next destination is in another AHPM in this card. The updated control information is retrieved from the Table and sent to the ring by using the reserved slot. If the payload is needed by the next service, a request is sent to the Ring Manager for additional slots. Upon receiving the grant, the packet is retrieved from the Cache and sent ("push operation"). The Cache is freed. Otherwise, the packet stays in the cache, until some other service or the switch send a request for it ("pull" operation).

- Next destination lies in another card, or this was the last element of the service chain. In this case, the control information is sent to the M&M Interface, so that it will be eventually switched across the backplane. If the payload had been cached, the Switch Interface will send a request for it whenever the switch's allocator allows. Additional slots will then be requested from the Ring Manager, and the payload will be sent directly to the Switch Interface.

In all cases, after passing the control information (and the payload, if necessary), the Service Chain Table is updated and this information removed.

5.3.3.3 Services and EHM

As stated above, the services are the instances in charge of processing the packets. The goal of the EHM, which will be analyzed in the next chapter, is to ensure that no interference can arise among services, or between the services and the node itself. The EHM is hence the most critical element in the OOG architecture, for it falls upon it to ensure that the additional performance and

openness derived from the AHP are not at the price of diminished security or resource usage control.

5.3.3.4 Ring Master

Finally, the Ring Master manages the extra slots reserved for packet transmission. Basically, it contains a queue, where the AHPMs write their requests, and a point-to-point signal to every AHPM to give the grants. The algorithm employed to distribute the slots can be a simple Round Robin, in order to ensure fairness with respect to the stations. Alternatively, Weighted Round Robin could be employed, where the weights are proportional to the average number of requests sent in the past. In this way, fairness with respect to the actual load of every AHPM could be achieved.

All extra slots in every cycle are assigned to only one station. The rationale is to quickly complete the transfer of a whole packet, so that the receiving service can begin its processing. Multiplexing several requests in one burst would delay the start of service processing in all requesting stations.

5.3.4 Interface to the CPU and Configuration Procedure

The CPU plays a double role in most router architectures. On the one side, it realizes the routing and management functions. Hence, it must receive and process all signaling and routing packets. On the other, though, it is also charged with the configuration and management of the node itself and the modules therein.

The architecture of the CPU itself will not be considered here, since it is a common general purpose processor plus the corresponding periphery. However, the interchange of information with the BHP and AHP does imply important architectural decisions. Those are the aspects that will be dealt with in this section.

For the double function performed by the CPU, two interfaces have been foreseen in the OOG. First, the CPU is connected to the backplane as an additional station (s. Fig. 5.5). Hence, all signaling and routing packets will be forwarded to it by the line cards as if they were addressing any other interface card. This simplifies packet forwarding.

Second, a configuration interface is also included. Typically, configuration is a rare process, seldom with tight real-time constraints. Furthermore, the amount of data involved is much smaller than for packet forwarding. On top of that, configuration implies a 1 to N relationship between the CPU and the line cards, hence making parallel interconnects useless. Consequently, a bus presents the simplest but also sufficiently high-performance solution for this task.

Accordingly, a separated configuration bus interconnects the CPU with all line cards in the router. Inside of every card, this bus is terminated by a Bridge (s. Fig. 5.10). The Bridge is interconnected via two internal buses with all BHP

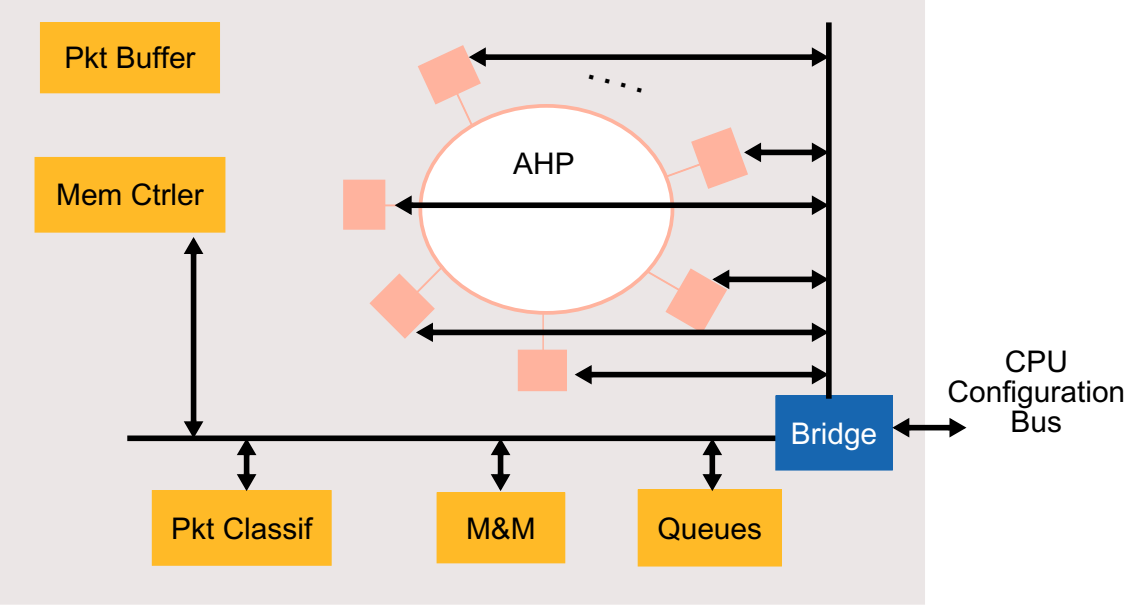


Figure 5.10: OOG Configuration Bus between CPU and All Line Card Modules.

and AHP modules in the line card. The function of this Bridge is to interpret the address of the commands sent by the CPU and select the module to be addressed. Furthermore, configuration of hardware systems mainly involves the writing or reading of registers. Hence, a second Bridge function lies in interpreting the commands sent by the CPU (using any proprietary management protocol) and translating them to simple write and read operations over the adequate registers.

However, in order to allow for management freedom by the service operator, the Bridge shall not interpret such configuration messages as are addressed to services inside of the AHPMs. In that case, the Bridge simply forwards the CPU message uninterpreted. In the AHPM, they are internally forwarded to the service. In this way, the service operator can design and use any management protocol of its choice, without restrictions. This fosters development freedom and innovation.

Such a configuration bus, nevertheless, is not adequate for the transmission of big amounts of data, as would e.g. be needed to reprogram an FPGA if new services had to be included. This would introduce an unnecessarily long configuration delay, on the one hand, and block the configuration bus to other cards or modules, on the other. For that purpose, the backplane is used. Large amounts of configuration or management information are sent across the backplane with a special packet header. In this way, no extra complexity in packet forwarding is introduced, while fast configuration is enabled. On the CPU side, the control of the configuration bus falls on the Device Driver, while the control over the backplane connection falls on the scheduler (s. Section 5.5).

5.4 Service Introduction & Interaction Manager Functionality

Network reliability is one of the critical assets, that network operators depend on to build a reputation. Accordingly, they would very reluctantly endanger that reliability by sharing node management with third parties. In order to achieve this goal, two requirements have to be met: On the one hand, the node integrity must be protected from service misbehavior. That is the task of the different resource management mechanisms introduced in this and the next chapter. On the other, the network operator must keep the final control over the node. Since there is no such thing as "absolute security", the possibility to control the activation and removal of services must lie on its hands. This is the task realized by the Service Introduction and Interaction Manager (SIIM).

In a shared node, furthermore, it is in the interest of all, that a third party acts as a judge of last resort. Service interaction can not be solved by service brokering among the interested parties, except if cooperation mechanism have been built in in their design [Kec02]. This is a strong and unnecessary constraint in service design, if an overruling entity can solve such interactions in the interest of node and service stability.

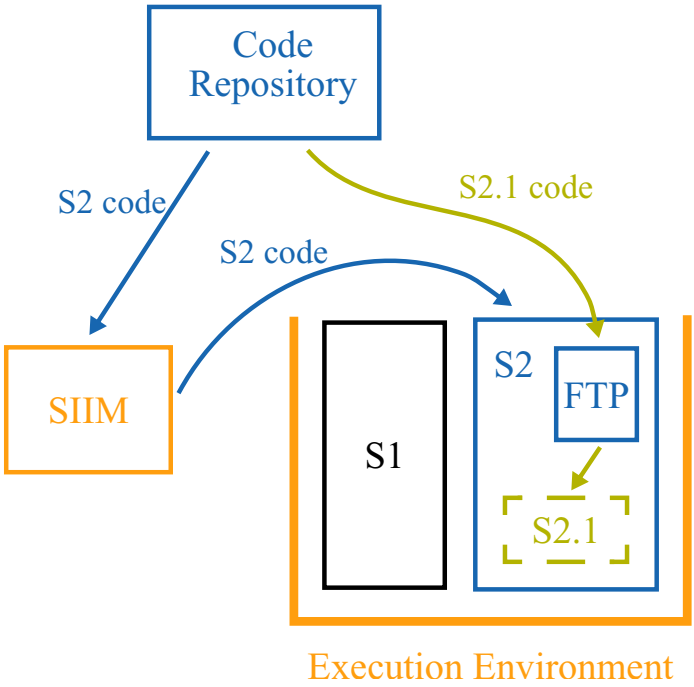


Figure 5.11: Service-triggered Functionality Extension.

The SIIM acts as the interface to the NAN. It is in charge of keeping track of, and communicating resource availability in the node. It also downloads the code belonging to a new service and installs it in the appropriate (software and

hardware) Execution Environment. Additionally, it must reprogram the different resource management mechanisms to redistribute node resources and enforce its usage agreements. The SIIM must also introduce the necessary filtering rules in the packet classifier, so that packets needing treatment by the new service will be accordingly forwarded. This is another information, that is negotiated between NAN and service operator in the Service Admission phase. For certain services, also the routing tables might have to be updated. This, nevertheless, is a process which, unlike traditional Active Network Nodes, can not be performed by the service. Routing and classifying are critical processes for the reliability of the node. A mischievous service might divert packets, which it does not have the right to handle, by modifying those mechanisms. In order to prevent that, only the SIIM itself carries out such tasks. In general, the OOG architecture does not allow direct interaction between services and resources, which are common to the whole node for security reasons.

Once this task is performed, though, the service operator is free to further manage, refine or enhance its service as it pleases. Since a service runs on a safe environment, it is allowed to download additional modules from service operator-specific repositories without NAN intervention. As represented in Fig. 5.11, a service instance (S2) might instantiate, as part of its service offering, a download service like FTP. It might then retrieve an additional sub-service (S2.1) and share its resources with it. The interfaces to the system as well as the overall allocated resources to S2 must remain unaltered. Since to enforce this is the competency of the SIIM, S2 can not endanger applications beyond its sandbox. It must be pointed out, that the ONM presents no mechanisms to check the *safety* of the code, since that presents unsurmountable technological challenges. Only its authenticity and integrity was guaranteed. It is in the interest of the service operator, to perform analogous checks in its own extensions, even if the network operator does not force him to.

A particular case is represented by the realization of capsule-based approaches *within* a service. The ONM foresees the forwarding of canonical IP packets, which precludes the inclusion of different layer 3 headers or even of radically different capsule structures. Nevertheless, as long as capsules present externally an IP header for packet forwarding and classification, services are free to encapsulate within their own formats. This allows the transport of code within packets, which will be interpreted by the service in traditional capsule fashion. Analogously, service extensions can be downloaded upon packet arrival, as long as they can be performed by the service itself, as in the previous example (Fig. 5.11). In this sense, the ONM presents a Programmable Network framework, within which Active Network architectures can also be realized.

An additional task performed by the SIIM regards such traffic aggregates, that shall be processed by several services consecutively (so-called "chained services"). An example could be a data stream, which shall first be watermarked and then encrypted prior to transmission. Assuming that both services are implemented

on an OOG, the SIIM has to handle their necessary interaction. As stated before, it would represent a strong development freedom restriction, if services had to be aware of each other, in order to cooperate. Furthermore, it would open a new avenue for possible conflict, since misbehaving services could deny cooperation, e.g. by not passing a packet to the corresponding next service, *de facto* performing a denial-of-service attack. As in the previous case, the alternative consists in introducing a middleware element in charge of chaining services. During the packet classification update process, if the SIIM detects two overlapping rules, it internally creates a list of services, that correspond to that rule. The packet classifier then passes the packet to the first service in the chain, together with the list of services. As was explained in section 5.3, a special module in the AHPM manages the list. Upon processing completion by the first service, that module intercepts the packet in its way to the network and diverts it to the next service. In this way, cooperation among services is not needed.

In spite of this, a problem exists: How to handle services, which alter the content or the state of a packet and how to establish the right order in the service chain. It might be, that the next service in the chain will not be able to perform its processing due to that alteration. As an example, consider the watermarking plus encryption example: If the processing would be performed in the opposite order (first encryption, then watermarking), the second service would be incapable of recognizing the packet stream as a video session, and watermarking would fail. This is one dimension of the feature interaction problem. To be able to solve such dependencies, the first step is to provide an accurate description of system behavior, not unlikely with the help of formal methods. In this concrete example, though, it might suffice with a description of which information regarding packet and/or associated state is needed by every service, and which changes will be performed on them. This amounts to an interface description. With that information, dependencies should be recognizable and an adequate chaining possible. None the less, this topic is beyond the scope of this Thesis. Further details can be found in [Kec02].

5.5 Intra-service Communication

A value-added service developed for the OOG can consist both of software and hardware modules. As already explained, software modules can be developed for the management plane, in which case they would run on a Virtual Machine inside a general purpose processor. Software for the data plane, on the other hand, needs to be tailored to the platform and can not be placed on top of several abstraction layers. To prevent security threats and, at the same time, enhance performance, it will run on embedded processors on the FPGAs. The hardware modules will be directly developed for the FPGAs. In such distributed applications, the intra-service communication and data interchange can have a determinant impact on performance. Depending on its implementation, it can

also negatively affect application interference, especially if applications are allowed to directly communicate with each other. To prevent both effects, a dual communication system for intra-service communication is foreseen. Inter-service communication is not allowed beyond Execution Environment boundaries, i.e., only services belonging to the same service operator can "talk" to each other.

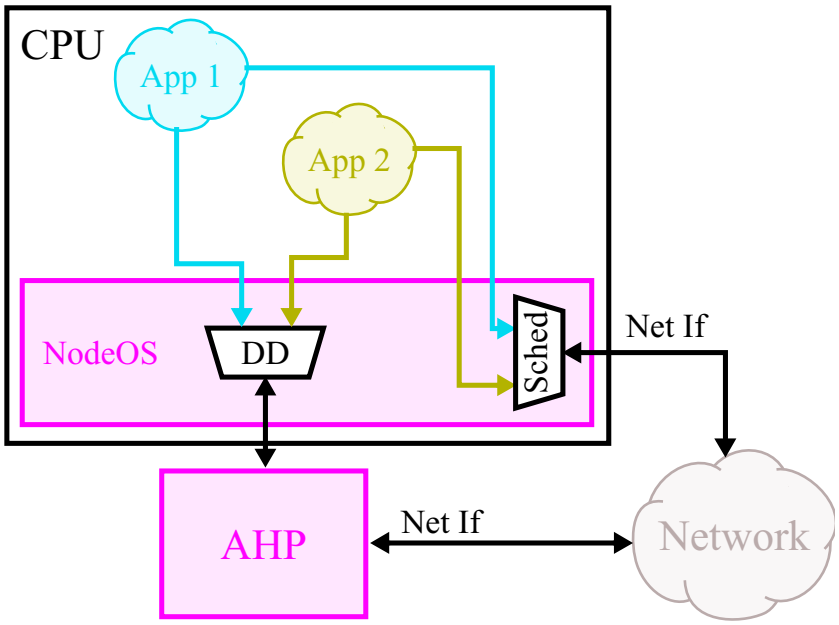


Figure 5.12: Intraprocess Communication Channels.

Traditionally, a software component interchanges data and control information with a hardware component via a Device Driver (DD, s. Fig. 5.12). The use of Device Drivers presents two disadvantages: On the one hand, they are usually not developed for high throughput. On the other, Device Drivers are multiplexers, in the sense, that they have to provide access to the hardware for all applications running on the system. As a consequence, a DD can be seen as a shared communication resource, that has to be shared according to the resource usage policy. Besides, a DD is a kernel instance, and has thus the capability to endanger the system stability, if it would be compromised. It must accordingly be protected from the applications. The ONM Device Driver for intercommunication with the AHP consequently presents a very simple API to the applications, which simplifies its management. Additionally, it includes a scheduler to regulate access to the AHP, instead of simply serving requests as they come. Together with the scheduler, Access Control mechanisms restrict the visibility of AHP resources to the applications. The rationale behind it is to prevent sniffing information from the AHP by software modules. This channel will be used for the interchange of configuration and control information, as well as small amounts of data, since it can not provide high throughput.

The second communication channel is a traditional network interface, like Gigabit Ethernet. This interface will be used for massive data interchange (e.g., to download a new FPGA configuration). The network interface inside the NodeOS foresees the use of a scheduler to regulate bandwidth usage, like Deficit Round Robin. The cost incurred with this solution, which provides good isolation and throughput, is the additional delay incurred by the network interface. On top of that, the interchange of small amounts of data would incur a high overhead penalty, since packet formats include a constant amount of header information. This solution, however, directly allows the implementation of AHP and BHP in different devices interconnected by a network, if desired.

5.6 Security & Resource Management in the OOG

Security is the main concern of the OOG, together with performance. To that end, besides the security checks performed for the introduction of new services (described in the previous sections), runtime checks are performed. Since the validity and the integrity of the source code has already been validated through the service admission process, runtime checks should concentrate on safety and QoS issues, i.e., resource management mechanisms. Taking advantage of the information provided by the NAN on agreed service resource usage, resource management is performed at four different levels:

- At the Execution Environment level: EEs schedule access to their allocated resources. This comprises Access Control, on the one hand, and QoS, on the other. This second aspect corresponds to the access frequency and bandwidth associated with every application and every resource, like memory or I/O. Nevertheless, since several EEs can be active on a OOG at the same time, these checks are not sufficient.

- At the Operating System level: To distribute resources and enforce usage among EEs, a third party is needed. This role is played by the OS. As explained before, the OS gives the network operator the ultimate control over the node, in case of application misbehavior not caught by its respective EE.

- At the Device Driver level: As a part of the OS, the Device Driver plays a very special role, since it regulates access to the other application modules residing in the AHP. For that purpose, access control and scheduling are replicated here.

- At the AHPM level: Modules implemented directly in hardware can not be satisfactorily controlled by the OS (s. Section 3.2). Thus, to prevent interaction at the hardware level, an Embedded Hardware Manager has to be implemented. It will take up the Access Control and scheduling tasks for the embedded microprocessors and hardware applications.

As can be seen, the functions realized at the four levels are essentially the same. In principle, it could be argued, that this represents an unnecessary overhead. Nevertheless, this replication is necessary, since the different mechanisms solve the same problems within different scopes and time granularities. This derives from the fact, that different resources and applications have different access patterns. A memory access by a hardware module must be, e.g., satisfied within a few clock cycles, while a packet transfer to (or from) the network interface may last for thousands. The granularity of control spans several orders of magnitude. On top of that, some resources and applications are directly managed by the service operator, while others are controlled by the network operator, which implies a different scope. Within the node, it was shown, that the OS does not have a thorough sight of all resources, since many platform-dependent details at the hardware level are hidden from it.

This chapter has presented the Octopus Open Gateway architecture, this Thesis' vision of an active router. In such shared nodes, resource management plays a critical role in guaranteeing network reliability. Without that, network operators would never accept this new paradigm, in spite of its other advantages. While most Active & Programmable Network proposals have dealt with the issue of resource management for software systems, its counterpart for hardware or embedded systems has not been widely studied. The reason lies in the neglect of shared programmable hardware systems in those proposals. The following chapter will analyze an architecture to share the AHP while providing safety and QoS guarantees to every individual application.

6 The Embedded Hardware Manager

The last two chapters presented a general description of the Octopus Network Model and its main components, which represent this Thesis' vision of an overarching programmable network architecture. In this section, one of the most critical and less studied elements of such an architecture will be reviewed: A security and QoS management entity for hardware resources, denoted as the Embedded Hardware Manager (EHM). A first section will summarize the necessity of such an entity. Next, the overall EHM architecture and its main characteristics will be presented. Last, a more detailed description of the composing elements will take place.

6.1 Why Resource Management in Hardware for an OOG

As it is, modern network nodes perform a myriad of functions in hardware. The OOG, more concretely, presents a programmable hardware platform to support value-added services in the form of digital circuits. It is the *sharing* of this platform, which demands resource management within the AHP.

The OOG has been designed as an open node, in which different service operators can place and operate their services directly and independently of each other and of the network operator. The risks associated with sharing a platform among different services could be listed as: Degraded performance caused by a misbehaving application, breaking the QoS agreements with the other services, illegally retrieving information related to other applications, mischievous service manipulation and node crash (s. Section 3.2).

Even for simpler, centrally managed nodes with programmable hardware platforms, similar concerns arise. Such hardware designs, like any other big software project nowadays, are not entirely developed by the service programmer. Parts of it are bought from other companies in the form of pre-processed, synthesized building blocks. Such is the case with standard interfaces, microprocessor cores, PLLs, etc. The result is a variety of applications, including blocks from different manufacturers, running concurrently on the same platform. In this scenario, bugs and unpredictable service interactions are always a possibility.

If open nodes are foreseen, the risks are accordingly higher. To prevent such dangers, an effective *isolation* among services has to be introduced. Section 3.2 reviewed the mechanisms employed in software, and also the reasons why OS control of hardware applications is not sufficient. Hence, to guarantee safety, security and QoS in a programmable hardware platform, access control and resource management at the hardware level have to be performed.

The goal of the Embedded Hardware Manager is, thus, to take up these tasks and hence to replicate the functions performed by the OS at the hardware level. To a large extent, similar mechanisms can be used, with adequate translation to this new domain. The development of a coherent, integrated security and QoS policy for programmable hardware platforms has not been attempted before. With increasingly powerful FPGAs and myriads of functions being performed in hardware, such a module as the EHM becomes indispensable.

6.2 EHM Design Criteria

The OOG in general and the EHM in particular were designed with three goals in mind concerning their field of application:

- Shareability. Envisioned as an open node, it had to support different applications from different service providers.
- Applicability in different networking scenarios. To be able to cope with innovative services, it must be capable of coping with very different requirements.
- Performance. The goal is to design an active router capable of working on real networks, especially in the MAN and edge area.

From these goals, several design criteria for the EHM were derived:

- Universality of design: The goal was to create an instance, that could be adapted to a variety of node architectures and, especially, to a variety of applications and off-chip resources. Nevertheless, hardware modules are too close to the platform to be portable: Bus widths, number and kind of SDRAM modules, type of CPU, etc. are parameters, that have to be known for a working implementation of the EHM. Hence, the goal was to design a portable architecture, although accepting that any implementation thereof would necessarily be bound to the chosen platform.
- Service independence: The EHM should be capable of accommodating a broad palette of service behaviors in order to be useful. In that respect, the interface between the EHM and the applications plays a fundamental role, since it limits the interactions possible between both. In order not to restrict service functionality, it has to support a varied set of modi: Single

word access, bursts of known or unknown length, a flexible set of control signals, etc.

- Individual QoS guarantees per application: Different applications will have different requirements when accessing the same set of resources, depending on their uses of them. Conversely, they can also require different kinds of resources, like SRAM or SDRAM. Accordingly, the access pattern must be set per application and be flexible enough to accommodate individual uses.

- Security / isolation among applications: As stated before, application interference is unacceptable. This encompasses not only QoS interference, but also access to and/or modification of data or state associated with another application. This involves *what* and *to which purpose* to access, while the previous point deals with *when* and *how* to access shared resources.

- Scheduling specialized per resource: Different off-chip resources present very different operation patterns. SDRAM memory and the network interface, to choose but two, both can be seen as burst-oriented sources of information for the applications. However, their operation time scales are very different, with SDRAM working in the ns area (a few words) and the network interface in the μs area (a few thousand words). Accordingly, to regulate access to them different scheduling algorithms will be needed.

- High throughput and utilization: Performance is the reason to introduce hardware in networking systems. Hence, the EHM must not greatly reduce system throughput in order to be useful. In order to be efficient (and not only effective), utilization must be kept at a high level, too.

- Modest resource consumption and complexity: FPGAs are increasingly resource-rich, in terms of logic cells as well as additional building blocks, like on-chip memory or interfaces. Nevertheless, advances in FPGA capacity and power should derive in more real-estate for the application programmer and not for EHM consumption. Only a moderate resource usage would be acceptable.

- Limited additional delay: The price to pay by introducing an intermediary between the services and the off-chip resources is additional delay and jitter. Since overall end-to-end delay nowadays is limited by queuing and processing delay at the intermediate nodes, the EHM must not add long loops in the process, in order to be adequate for real-time as well as non-real-time applications.

From the above mentioned criteria, an inherent limitation emerges: Limited scalability. The complexity of a module dealing with scheduling N inputs and M outputs will be somewhat in the order of $O(N \times M)$. Although this strongly

depends on the architecture chosen, it remains intuitively true e.g. for the resources used by the EHM in terms of interfaces. Furthermore, there is a limit to the multiplexing gain achievable with limited delay. Sharing a SDRAM bank among 100 applications, no matter what access pattern they show or how good the chosen scheduler is, does not seem very reasonable due to the long maximum access delay. Hence, the architecture presented here is only useful for a limited number of applications running concurrently on an FPGA. Nowadays, the possibility of sustaining more than one application per device is barely emerging. Thus, the solution presented here will be adequate still for years to come.

6.3 Embedded Hardware Manager Architecture

The role of the EHM, thus, is to act as an intermediary among the services running on every AHPM FPGA and between those services and external resources to be shared, like SDRAM modules, network interfaces, CPU, etc. (s. Fig. 6.1). It will be in charge of guaranteeing isolation among services. To that end, access control mechanisms and scheduling algorithms will be used.

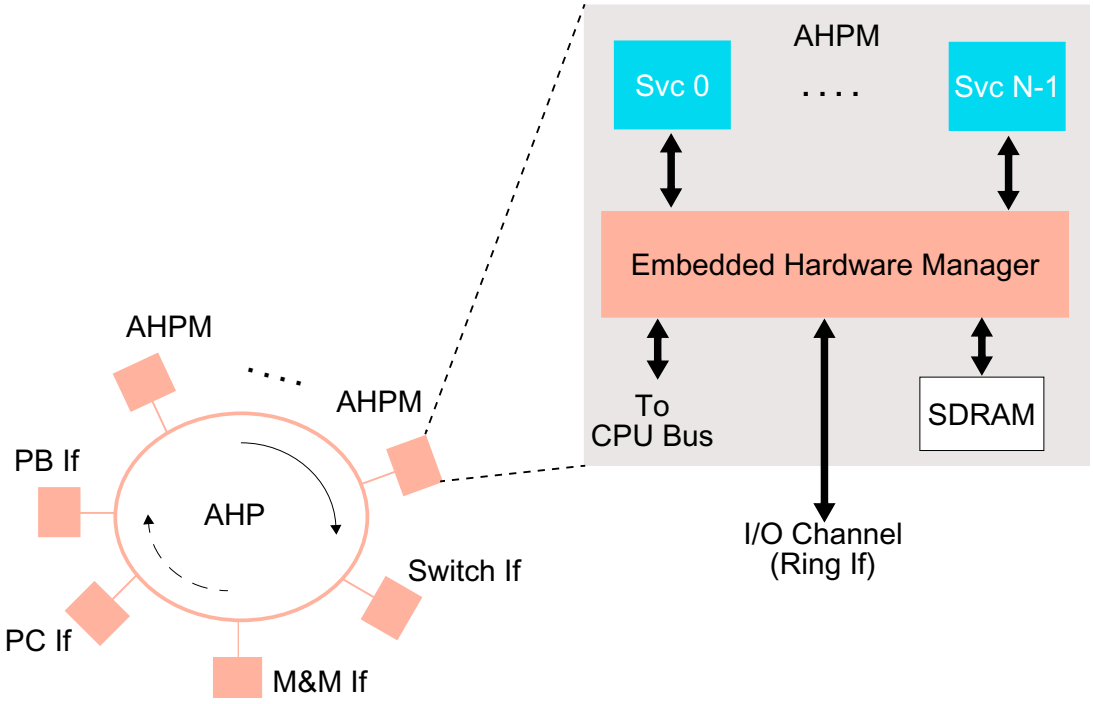


Figure 6.1: The Placement of the EHM in the AHP.

From the point of view of the realization, hardware applications are synthesized VHDL or Verilog code. Accordingly, the EHM is implemented also as a VHDL module. The main difference is, that it can be conceived as a piece of "in-

frastructure": For the applications, it represents their interface to the resources and is a fix element of the system. Figure 6.2 presents the basic architecture.

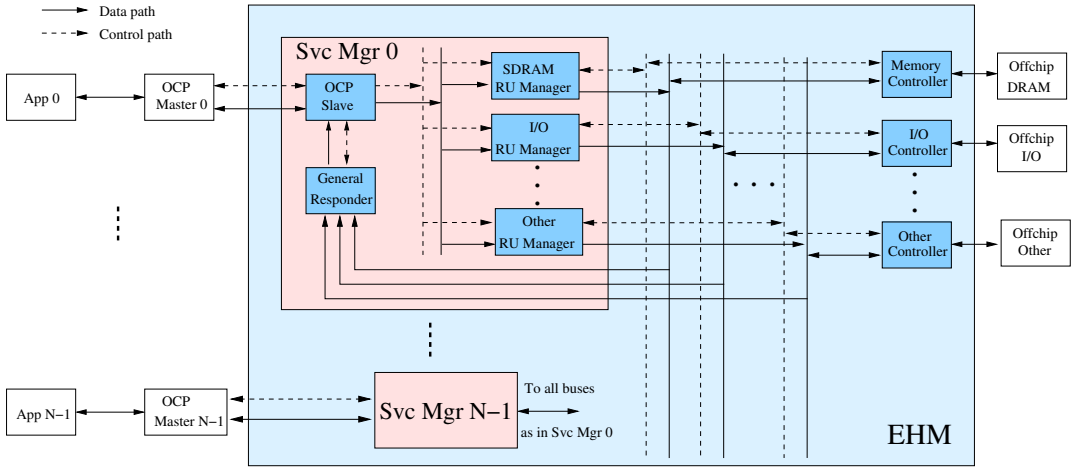


Figure 6.2: The EHM Architecture.

Every system or application present in the chip has access to the off-chip resources *exclusively* through the EHM. In this way, bypassing the security and scheduling mechanisms is prevented. As an interface between the applications and the EHM a fully compliant version of the Open Core Protocol (OCP) interface [OCP01] has been chosen, as explained in section 6.4.1.

Access control and scheduling are implemented in the Embedded Hardware Manager in the form of *distributed algorithms* running on the Service Managers (Svc Mgr). There is one Service Manager per application. Every Svc Mgr controls access to all shared resources (memory, I/O, etc.) for its application. To that end, it is divided in a set of Resource Usage Managers (RUMs), one per shared resource. It is in the RUMs, that the scheduling and access control algorithms are actually implemented for every application-resource pair, as will be described in section 6.4.3.1 for the DRAM case and in section 6.4.3.2 for the network interface case.

Obviously, an application's QoS and security requirements vary depending on the resource: E.g., a computing-intensive application like image processing might need a lot of memory bandwidth and less I/O capacity, while encryption barely uses memory access but requires packet processing at wire speed, thus intensively using I/O channels. Furthermore, the capabilities and access patterns of every resource are also different: Single-access (as in SRAM) vs bursts (as in DRAM), single burst (as in memory access) vs multiple bursts (packet transfer), etc. Every RUM is therefore optimized for its specific resource.

The EHM receives the relevant information to configure these elements from the Network Admission Node during the service introduction process. The QoS

and access rights associated with every new application are downloaded from the NAN together with the application code, as was explained in chapter 4.

Communication between an application and a resource can be bidirectional. RUMs control the transfer of requests from the service to the corresponding resource, responses being handled by the General Responder.

Access requests are passed to the Resource Controller, which polls every Svc Mgr following a certain strategy. E.g., for the SDRAM Controller, two different strategies have been implemented to give access to memory. The first and simplest is a round-robin scheduler, that grants access to the bus to every Svc Mgr in turn. The second implementation tries to optimize memory usage by dividing time in so-called *epochs*. Requests being served in an epoch will be rearranged to better utilize memory and minimize the overhead associated with DRAM management.

The interconnection between the Service Managers and the Resource Controllers has been realized with a set of buses, two per Controller: One for control information and one for data transfer.

These buses can be seen as multiplexers: Every bus interconnects all Service Managers with one Controller, and *vice versa*. More precisely, every RUM inside a Service Manager has access to one of these bus pairs, the one associated with its corresponding resource, as depicted in Fig. 6.2. The scalability of such a solution is limited, as was analyzed in the section on router interconnects (s. Section 2.2.4). In the general case, a switching matrix would be a better alternative. Nevertheless, the election of a bus array was made considering the inherently limited scalability of the overall EHM design. A switching matrix would need far more resources, due to its higher complexity (allocator, VOQs, buffers, etc.) The bus array, thus provided sufficient scalability with far greater economy of resources and simplicity.

For clarity, the operating mode of the EHM has been exemplarily represented in Fig. 6.3: Two applications, I and J, have a request (e.g., read, write, interrupt) for a certain shared resource, k. This request is sent to their corresponding Svc Mgr, which forwards it to the appropriate RUM for resource k (RUM k). The RUM performs two checks on the request: If it has the right to access the resource in the way stated in the request (AC, in the figure), and if it has enough resources available to do so (represented as a token bucket with parameters b and r, see also section 6.4.2.1). The value of the token bucket parameters and the access restrictions are individually set for every application and resource by the SIIM, hence the notation $AC_{I,K}$, $b_{I,K}$, etc. Obviously, the token bucket parameters are not independent of each other: The SIIM ensures, that the overall request rate does not exceed 100% of the resource's capacity.

Should the RUM deem the request acceptable in terms of QoS and security, it is signaled to the resource controller responsible for resource k. The controller then schedules the requests following the algorithm chosen by the designer and serves them accordingly.

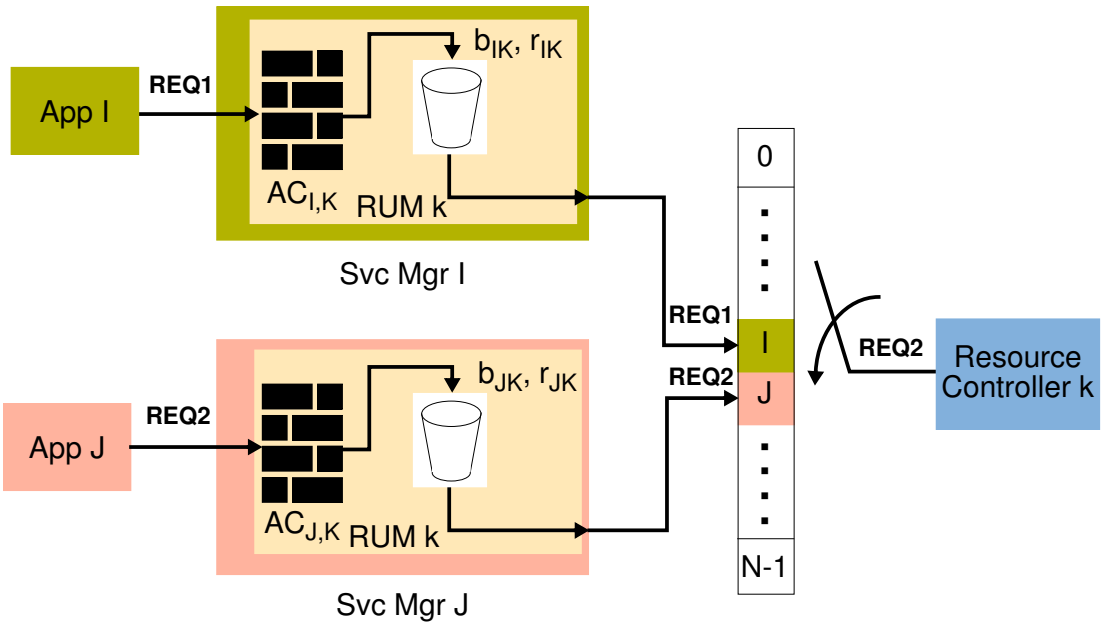


Figure 6.3: Distributed Access Control and Scheduling in the EHM.

6.4 EHM Main Elements

Since the EHM contains modules, which are either application or resource specific, adequate examples had to be chosen to illustrate the architecture. To that end, two RUMs and two applications were designed and implemented, which represent two extremes in their resource requirements and behavior.

First, an SDRAM RUM and its corresponding Controller were implemented. Memory being one of the fundamental resources for any application and one of the constant elements of any system architecture, it seemed an appropriate choice. The second set consists of an I/O RUM and its corresponding Network Interface Controller. Since this Thesis is concerned with networking equipment, access to the transmission medium was a second fundamental choice. Additional modules could be included, since the EHM presents the claim to be applicable for most shared resources.

None the less, a clear limitation exists: The EHM adds delay to resource access. Hence, it is unsuited for applications/resources with extremely stringent deadlines. An example would be an application accessing an SRAM module. The scheduling overhead to share the SRAM interface alone would dwarf the usual access delay of around 3 clock cycles. Nevertheless, such tight deadlines are usually dealt with within the FPGA scope by integrating such elements on-chip. SRAM being again the best example, any modern FPGA has a pool of SRAM blocks, that can be individually associated to designs running on the chip. Since such memory blocks are not shared, no need to supervise them via EHM exists in the first place.

To illustrate the usage of the EHM by different applications, two were designed: An accounting application and an encryption module. The first is a very memory intensive service, while the second requires packet processing at wire-speed, being thus very I/O intensive. Together, they represent two extreme cases in resource usage for both types of RUM and Controller presently included in the EHM. Details on these applications will be given in the next chapter.

The following sections describe the different architectural elements in more detail.

6.4.1 The Interface Between the Services and the EHM: OCP

Independently of the concrete standard finally chosen, a number of requirements were set on this interface. Isolation among applications being a key concern, it followed, that a point-to-point interface between service and EHM was a convenient solution, since it absolutely prevents interference. An alternative would have been to use tristate buffers on the application side. The choice was a trade-off between logic and interconnection resources: The former solution is more interconnection-intensive, the latter more logic-intensive. Finally, the overall decision falling on the use of the OCP standard, it followed that the point-to-point solution had to be adopted, for it is the one used by OCP.

The advantages of that interface were many: It is an open standard, thus not tying to any vendor-specific solution. Furthermore, its characteristics are well adapted to the needs of a system on chip: It is an extremely flexible interface thanks to its many optional signals, which allow to adapt to a vast range of system behaviors. It is general-purpose, thus allowing the interconnection of a wide range of applications (microprocessors, memory controllers and peripherals). It allows bursts of different sizes and is a point-to-point interface, thus eliminating any possibility of interference in accessing the EHM among the different applications on the chip.

Since the EHM design is a general, platform- and technology-independent resource manager, it can also support other standard, non-proprietary interfaces, that could further ease the integration of third-party products, like the Wishbone [Sil01] and AMBA [ARM99] interfaces. The EHM was successfully tested with all three interfaces (OCP, Wishbone, AMBA).

Any interface presents two sides. In this case, as represented in Fig. 6.2, an OCP Master module has to be integrated with the application, while an OCP Slave resides within the Service Manager. This represents the only severe constraint set by the EHM to application developers. For that reason, the interface had to be sufficiently flexible to accommodate a large variety of access patterns. At the same time, it had to be close enough to the hardware platform, so as not to artificially obscure the exact desired behavior. Additionally, being a module, with which all applications have to comply, it was advantageous to rely on a

non-proprietary standard, which would facilitate its adoption by developers. The OCP standard accommodates all those requirements, as explained in [OCP01].

Requests coming from an application are locally acknowledged by the OCP slave, in order not to block the OCP interface, which otherwise would have to wait for the response (e.g. data coming from a read request to SDRAM) before being freed for subsequent transactions. This provides a certain multiplexing gain in interface usage. Whenever a response is ready to be transfered to the application, it is sent to the General Responder, which acts as a multiplexer, serializing responses coming from different resources, potentially simultaneously. Although the present implementation serves every resource in a round-robin fashion, the General Responder could assign priorities to them, in case that certain requests had more stringent delay requirements.

The Service-EHM interface is the *only* access point to off-chip resources for any application. This is the only way to ensure access control and QoS enforcement. Otherwise, applications could circumvent the EHM and directly access the I/O pins with help from its own logic. To prevent this, the EHM has to surveil all FPGA I/O pins. This measure also prevents most hardware attacks at the electrical level, as explained in section 3.2.

Additionally, an aggressive resource usage policy is implemented. Should an application try to access unauthorized resources or surpass its QoS share (as defined by the Service Manager), the interface is shut down by the OCP Slave, preventing any further interchange between application and resource. In case of QoS exhaustion, the policy so far is to keep it closed until the usage level falls under its maximum allowed limit. Should any form of security breach be detected, the interface will be shut down indefinitely and an alarm will be sent to the system administrator.

The OCP interface presents to the application a single attachment point to off-chip resources. Internally, though, the application may include a number of logical interfaces: For different purposes (e.g., data/signaling), for different execution paths (e.g., threads), for different resources (e.g., memory, I/O) and even for different instances of the same resource (e.g., memory bank 1, memory bank 2). Every application thus has to multiplex access to all shared resources on the OCP interface. How that multiplexing is done is application specific and falls under the responsibility of the application programmer.

The interface is a critical factor in the EHM design, for it defines which kind of behavior is possible. In that sense, it could constrain development freedom, if not chosen carefully. The choices made here try to combine a maximum of freedom with an effective security and QoS policy.

6.4.2 Service Managers

A Service Manager reunites and coordinates all Resource Usage Managers associated with an application, as well as being the interconnection point to off-chip

resources. It serves the requests from the application and delivers the responses from the resources. To illustrate its behavior, in this section two representative RUMs will be described.

The total number of Svc Mgrs depends on the number of applications, while the total number of RUMs depends on the number of shared resources. In principle, these modules could be dynamically instantiated in the EHM whenever a new application or resource is included in the system. Modern FPGAs allow this, thanks to the partial reconfiguration support. The EHM architecture is modular enough to allow it.

None the less, in its present form, the EHM presents a constant number of Svc Mgrs and RUMs. In order to include new ones, a whole new EHM instance has to be programmed in the FPGA, hence interrupting service operation. This is due to the fact, that the FPGAs employed at the IKR do not support partial reconfiguration.

6.4.2.1 SDRAM Resource Usage Manager

The SDRAM Resource Usage Manager (RUM) regulates access to the data interface according to the security and QoS policies chosen. For that, it performs two functions: It controls resource usage and also checks the validity of memory addresses being accessed. Figure 6.4 presents the basic architecture.

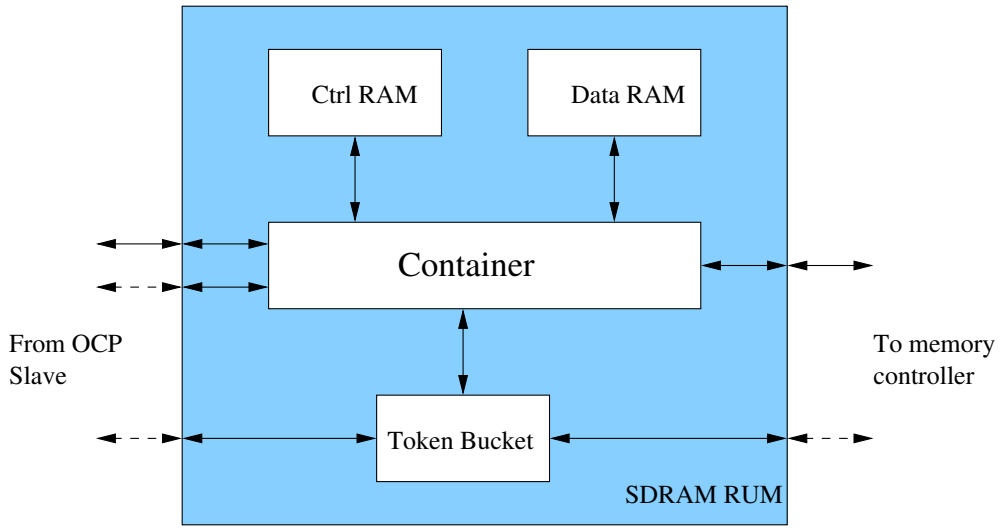


Figure 6.4: SDRAM RUM Architecture

The SDRAM RUM is functionally similar to a Memory Management Unit (MMU) in a computer, with a number of important differences:

- The EHM is a distributed system. An MMU is a central instance regulating access to memory for diverse subsystems and peripherals. In the EHM, the

equivalent functionality is distributed among all SDRAM RUMs, plus the SDRAM Controller.

- Emphasis on individual QoS guarantees. MMUs are usually designed to achieve high utilization. The EHM, additionally, provides hard individual QoS guarantees and isolation to every application.

- Runtime programmability. The EHM presents a set of registers, where the scheduling parameters and the address range associated with every application (to be explained shortly) are written. In this way, the QoS policy can be changed by the SIIM (under NAN control) at runtime without disrupting service operation.

Since access to memory will be granted according to a scheduler, it can not be guaranteed that a request will be answered immediately. Hence, some form of intermediate buffering is needed. The SDRAM Resource Usage Manager performs that function. It can be seen as a queue storing the write and read requests, plus the corresponding data in the case of a write. This role falls on the Container and its associated Control and Data RAM in Fig. 6.4.

To control resource usage, a scheduler has been used that represents a good compromise between ease of implementation, performance and effectiveness. Especially important is that the scheduler will not allow large jitters in the access to the bus. In packet schedulers, for example, the time unit is large: It is the time needed to send a whole packet to the network. But in memory access, the time unit is much smaller, since it is only the time needed to send a data burst, which is typically at least an order of magnitude shorter than a packet. Hence, delaying a burst the equivalent of a packet transmission time might be unacceptable for delay-sensitive applications. That eliminates many current schedulers used in networking.

To set an upper bound in delay and jitter and yet allow the burstiness typical of memory access, a combination of token buckets [Tan00] is used. There is a Token Bucket for every application. Its parameters (denoted as b and r in Fig. 6.3) are chosen to reflect the bandwidth and burst size accorded to that application and thus represent the mechanism to grant different QoS to different applications, according to their needs. b represents the bucket depth, which indirectly sets the maximum burst size allowed by the bucket. r is the token rate and sets the long-term throughput assigned to this application.

Furthermore, the maximum time that an application can use the bus is limited by the maximum burst size. Once exhausted, the Memory Controller grants access to the next application with a pending request. In this way, the maximum delay is known and bounded to $(N-1) \times MBS$, where N is the number of applications and MBS represents the maximum time allowed to transmit the maximum burst size on the OCP.

The Container stores the requests and corresponding data in a small set of on-chip RAM. Access to memory is granted to these requests in a first-come-first-served basis. Storing requests is independent from actually delivering them to the Memory Controller. That process is controlled by the Token Bucket. Once a request has been made to the Memory Controller, the RUM must wait for a grant before the data can actually be delivered. Depending on the version of the Memory Controller, requests are served either in a round-robin fashion or according to a performance-maximizing strategy, as described in section 6.4.3.1.

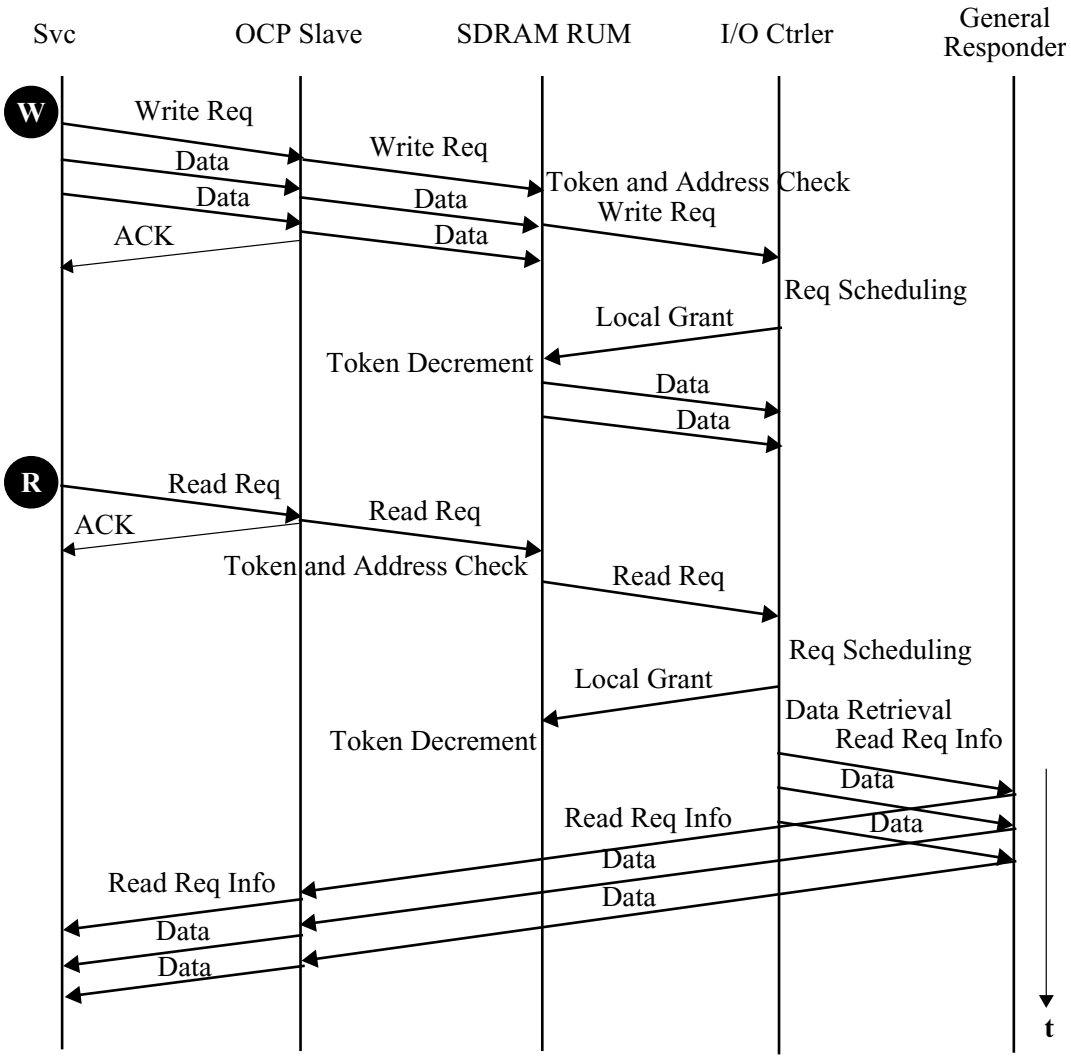


Figure 6.5: Memory Data Retrieval and Write Procedure.

Nevertheless, a malicious application might request bus usage to transmit a shorter burst and once granted, try to use up the maximum allowed time. To prevent this, the RUM keeps track of actual usage and discounts tokens accord-

ingly. Once the limit has been reached, the transaction is aborted if it was not finished by then. In this way, bus misuse is prevented.

The second task of the RUM is to map requested memory addresses to the address space granted to the application. For that purpose, a virtual memory scheme is used. Only the low bits of the address bus are taken into account by the RUM. It then prefixes the high order bits according to a virtual address table and compares the resulting address with the last valid address for that application. If the requested position falls out of bounds, the request is dropped. In this way, peeking into memory or even altering its content is prevented.

The implemented strategy is certainly not the only one possible. Nevertheless, this choice was made looking for bounded jitter, good isolation and implementation simplicity (so that it would suit a hardware implementation).

A typical write (denoted with a "W") and read ("R") operation have been depicted in Fig. 6.5. In the write cycle, the SDRAM RUM stores the data and waits until enough tokens have been amassed. At the same time, the address is checked ("Token and Address Check"). The Memory Controller schedules the requests and gives the grants accordingly ("Req Scheduling"). Upon reception of the corresponding grant, the RUM decrements the token counter and sends the data. The counter can be decremented before transmission, since the data are locally stored and the transmission time is constant.

In the read cycle, since the data are delivered to the application with a certain delay, some extra control information ("Read Req Info") has to be given back in order to univocally identify the request.

6.4.2.2 I/O Resource Usage Manager

The I/O or Network Interface Manager follows a similar approach to the one described above, with some important alterations derived mainly from its different operation time scale. A typical packet send and packet retrieval cycle has been exemplarily depicted in Fig. 6.6. In principle, the I/O RUM tasks are analogous to the SDRAM RUM's: To schedule access to the network interface (to receive or send packets) and to check if an application has the right to access a certain incoming packet.

Packets are relatively big pieces of information. In the previous case, bursts could be stored in an intermediate container and then scheduled and rearranged because the delay introduced by the process was small in absolute terms. Furthermore, rearranging requests in accessing SDRAM made sense, because there is a variable overhead associated with preparing memory banks to be read and written to. Both conditions do not apply in this scenario. First, the delay associated with sending or receiving a packet from the network interface is several orders of magnitude larger than normal memory burst lengths. Accordingly, the additional delay incurred by buffering a packet in the RUM prior to scheduling it would be unacceptable. Secondly, access to the network interface has a constant

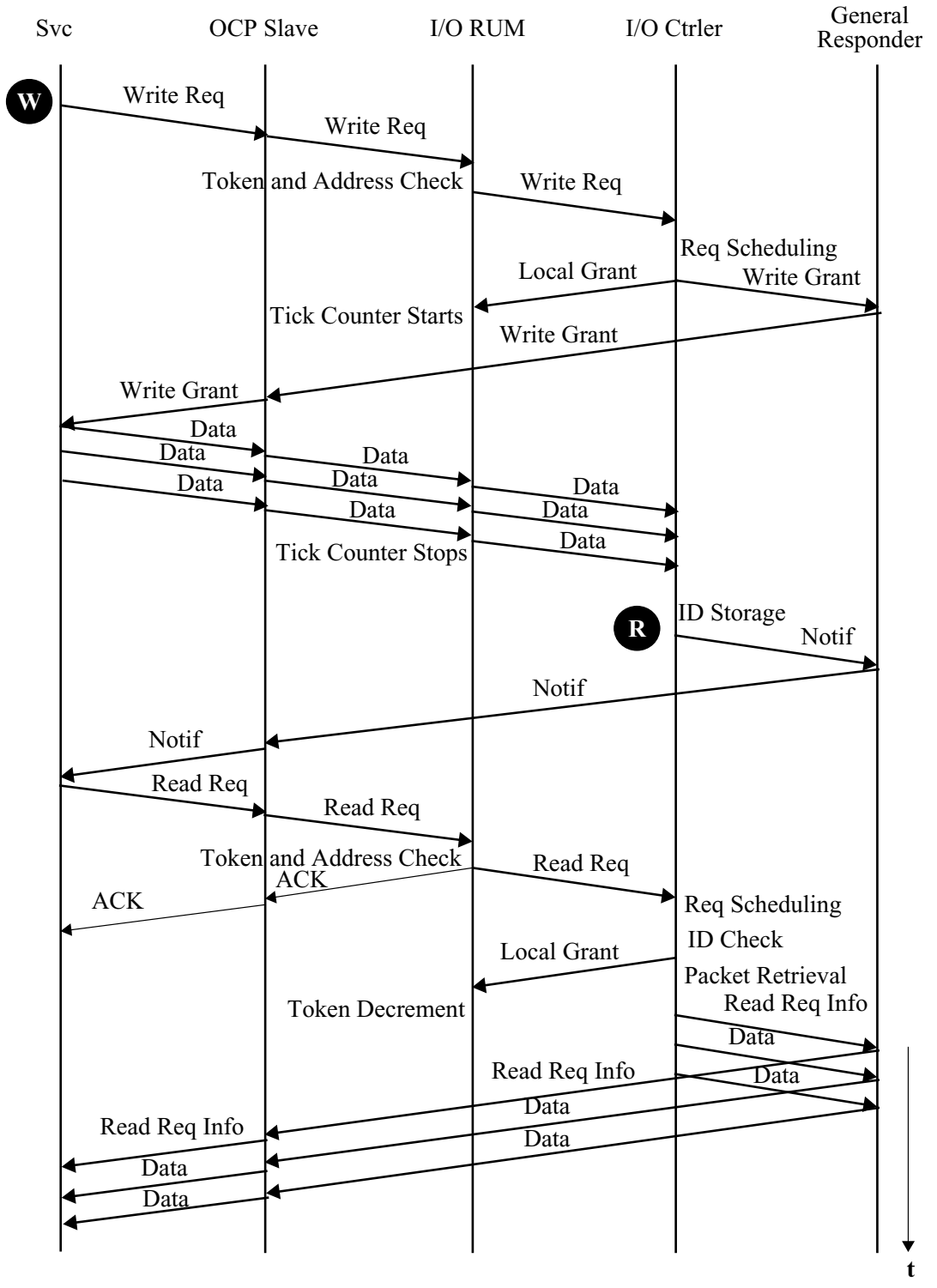


Figure 6.6: Packet Retrieval and Send Procedure.

overhead, so that it makes no sense to reorder requests at this stage[1]. On top of that, the on-chip memory bits necessary to buffer packets in every RUM prior to transmission would be considerable, while providing no real advantage. As a consequence, no intermediate packet storing takes place in the RUM, which is then reduced to a Token Bucket plus some control logic.

As in the previous case, the Token Bucket controls resource usage by the application. Prior to sending a packet, a check on the amount of available tokens is performed ("Token and Address Check" in the figure). Should they not suffice to serve the request, it will remain open until enough tokens have cumulated. Since requests are not stored in the RUM, there is no local acknowledgment by the OCP Slave. When a request can be served by the RUM, it notifies the I/O Controller, which schedules grants for every RUM in round-robin fashion ("Req Scheduling"). Upon grant reception, the General Responder notifies the application, which can then send its packet directly to the network ("Write Grant").

Misuse would still be possible, in that upon grant reception the application refuses to send data. Since the bus between RUM and I/O Controller is reserved for the duration of the transaction, a denial-of-service attack would *de facto* be performed. To prevent this, as soon as the grant is received by the RUM ("Local Grant"), a Tick Counter is started. This Tick Counter keeps track of the time actually used by the application. This value will be decremented from the Token Bucket state, no matter how long the packet was. Additionally, a limit is set to the maximum transfer duration, which roughly corresponds to the maximum packet size. Should the Tick Counter reach this limit, the transaction will be aborted.

In this way, a double test is performed: First, upon reception of the request, the RUM checks if the minimum necessary amount of tokens to transmit a packet of the notified size is in principle available. Second, when the transfer actually takes place, the application is billed for the resources really used, and denial-of-service attacks are prevented.

Receiving packets from the network follows a somewhat different procedure: Upon packet arrival at the node, a notification is sent to the EHM. The I/O Controller then notifies the corresponding application (via its RUM, "Notif"). It then falls on the application to request the packet from the network interface ("Read Req"). The RUM then checks the number of tokens, and whenever they suffice, it forwards the request to the I/O Controller and acknowledges the operation to the application ("ACK"). The I/O Controller will eventually deliver the desired packet via the General Responder. The rationale behind this separation between read request and read response lies in the additional delay incurred

[1]Note that the EHM is *not* responsible for performing output scheduling. The scheduling implemented here only concerns access to the Traffic Pattern Conditioner by the applications, for packets being sent to the network. There, an additional scheduling based on classes, marks or reservations can take place. Conversely, packets arrive to the EHM from the Packet Classifier on a FIFO basis.

during a read response. Since packets are not stored locally on the EHM, a read implies requesting and retrieving the data from the external network interface, which might take considerable time. In order not to block the application for so long, once the read request has been granted and issued, it is acknowledged. In this way, the application is free to employ the time until data arrival in some useful processing. The General Responder will send an indication to the application whenever the data are ready.

The second task to be performed by the RUM would be to validate requests by an application to receive certain packets. Otherwise, it would be possible for an application to claim a packet addressed to some other application, which implies an identity theft. This is prevented by building a table storing the application associated with every incoming packet upon arrival[2] ("ID Storage"). This table will be checked every time a read request is issued by an application ("ID Check"). Since network nodes have to process packets at wire-speed, the number of packets awaiting processing will be small[3]. Accordingly, this check can be more efficiently centralized at the I/O Controller. Distributing the task among RUM would unnecessarily use a larger amount of resources. The exact mechanism will be described in section 6.4.3.2. Obviously, no checks are necessary for packets originating *from* the application and going to the network.

As can be seen, the mechanisms are in principle very similar to the ones employed in the SDRAM RUM, but adapted to the intrinsic characteristics of the network interface. Consequently, similar advantages have been achieved, mainly resource misuse prevention, identity theft prevention, QoS guarantees and bounded maximum delay.

6.4.3 Resource Controllers

These elements are the direct interface to the off-chip resources. They must accordingly be optimized for the exact amount and type of resource present in the platform. I.e, their protocols toward the resources can be proprietary and the controllers themselves are not portable among platforms. It should be noted, that these are the only elements of the EHM, which are not platform-independent. As far as a standardization process succeeds in developing common protocols for the most common off-chip resources, like SDRAM modules or Gigabit Ethernet interfaces, the controllers will improve their portability. As way of example, SDRAM products from different manufacturers are, to a large extent, already compatible.

[2]This information is delivered by the packet classifier together with the packet header itself, as explained in section 5.3.

[3]The packet ID table has been implemented as a circular list. If an application refuses to retrieve its packets (e.g. because of internal processing overload), after a certain time they will be overwritten. Hence, packet loss is possible, but only if the application can not process packets at the arrival rate.

6.4.3.1 Memory Controller Architecture

The Memory Controller[4] depicted in figure 6.8 is designed to maximize memory usage within an epoch. An epoch is defined as the time needed to serve all active requests from the SDRAM RUMs at the moment of polling. The controller will then rearrange those requests to minimize the overhead associated with DRAM operation, as described shortly, and serves them accordingly. Once finished, a new epoch begins with a new polling phase.

DRAM efficiency is highly dependent on the access pattern. In particular the throughput depends on the burst length of the access, the address being accessed and the type of access (read/write). DRAM is partitioned in multiple banks to enable parallel access to the memory locations. Within each bank it is arranged in rows (pages) and columns. In case of successive memory requests on the DRAM, typically, there is very little delay involved between the two memory transactions if the second memory request operates on the same page as the previous request (page hit). There is a larger penalty if the second request operates on the same bank but different row than the one activated by the previous request (page miss). The penalty is smaller if the second memory request incurs a bank miss. In order to improve the efficiency of DRAM, some optimizations are required which can avoid the page miss, bank-miss and maximize the page-hits. The arbiter (memory controller) needs to permute the memory requests in such a way that it reduces the number of conflicts and improves the throughput.

The memory controller developed for this project uses Micron SDRAM memory MT8LSDT864 [Mic99]. However the design principles discussed here apply to any generic SDRAM.

Figure 6.7 shows how two successive memory operations can be executed under different conditions for the SDRAM used. A simplified representation of the timing diagram is shown where each box is equivalent to a clock cycle. Only the command and data bus are shown in the figure, the upper row showing the commands.

The mnemonics and the color code for the commands are also illustrated in the figure. The idle time in the two data transactions on the data bus is the reason for a loss in the SDRAM efficiency. There is a delay between two data transactions whenever there is a turn around from a read to a write operation or vice-versa. The penalty is larger when the following memory access incurs a page miss. All the transactions ending with SM in the figure are page misses. The penalty is maximum when there is write to read turn around and a page miss, too. In order to improve the efficiency of the DRAM, the page hits have to be favored and turn arounds and page misses have to be avoided.

[4]This Controller was mainly designed and implemented by Sarang Dharmapurikar [DL01], currently a graduate student at the Applied Research Lab, Washington University in St. Louis (sarang@arl.wustl.edu), during the author's stay at that institution. My contribution consisted in introducing the "epochs" and integrating it with the overall design.

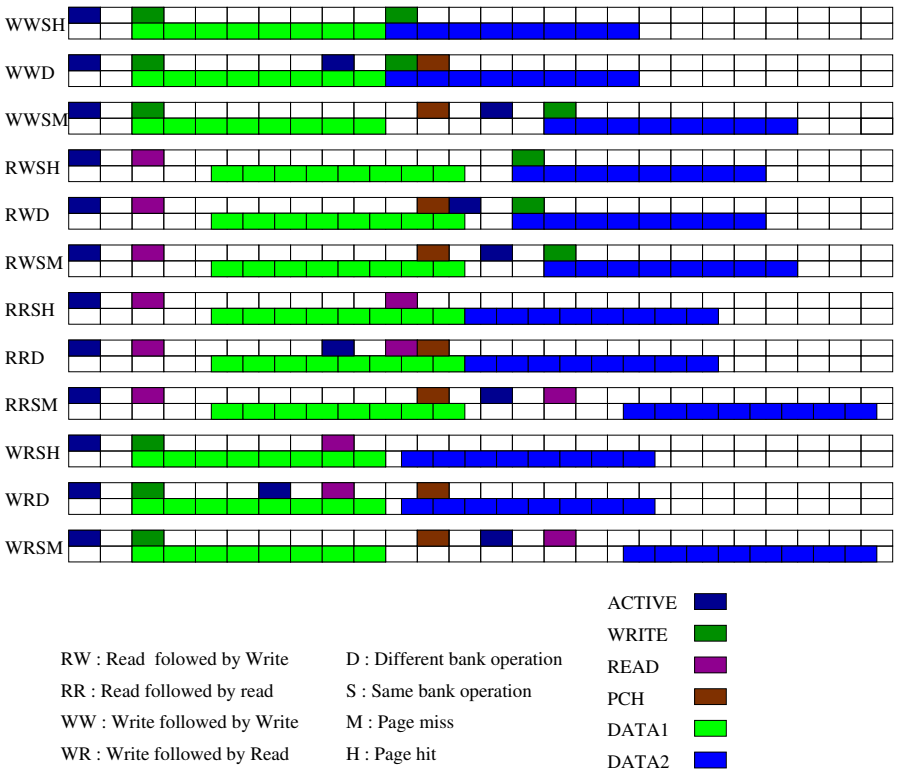


Figure 6.7: Timing Successive Read and Write Operations in Different Conditions.

The SDRAM controller is split up in a Request Selector and a Request Executer (s. Fig. 6.8). The Request Selector logic takes the pending requests (one from each module) and selects a request among them which, if executed at the appropriate time, maximizes the SDRAM data bus utilization. It makes use of a brute force technique to find the best among the pending requests. It compares the pending requests supplied to it against the current memory operation and decides which of the transactions shown in the figure 6.7 the pending request corresponds to. The address of the pending request and the operation type (read or write) are compared against the address and the operation type of the current access. The result of the comparison indicates the transaction type that a memory request represents. The request which corresponds to the transaction with the least penalty is chosen and passed to the Request Executer.

The Request Executer logic executes the selected request at the appropriate time. It issues commands to the SDRAM at appropriate clock cycles in a state-machine fashion. Since the basic scheme allows two requests to have overlapping operations (termination of one request and the initiation of the next request) two state machines are required to control the SDRAM interface. If one state machine is busy executing commands for a request then the new request is loaded into the other state machine. If both state machines are busy, then the Request

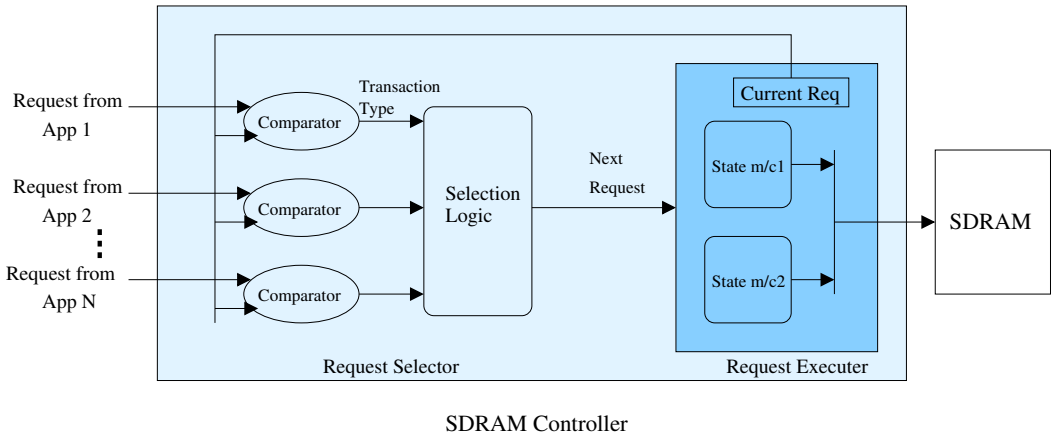


Figure 6.8: Block Diagram of the Memory Controller

Executer refuses to accept any new request and the scheduling operation has to be repeated.

It is the epoch concept that ensures throughput optimization without compromising QoS. The order in which memory requests are enqueued depends only on the Token Buckets. This ensures that every application will get its accorded bandwidth. At the cost of some extra jitter, the memory controller rearranges the requests that the Token Buckets have enqueued to achieve better overall throughput. Since the request permutation occurs only at the beginning of every epoch, the additional delay that a request can suffer is bounded.

6.4.3.2 I/O Controller Architecture

Contrarily to the SDRAM Manager, the network interface manager does not perform any optimizations in packet delivery. Basically, packets are received and delivered in FIFO order. It must be noted, though, that the order in which packets are delivered to the EHM by the Packet Classifier might already be policy-dependent and hence responds to an implicit optimization. Conversely, packets are taken from the applications following a round-robin sequence to achieve fairness. This fairness is weighted by the Token Buckets placed in every RUM.

Packets arriving from the network contain an indication regarding the application to which they should be delivered. This indication is used for two purposes: One, to send a notification to the corresponding RUM, so that the packet can be claimed by the application in due course. And two, to build a small database of pending packets. This database contains a univocal packet ID and the application ID to which it should be delivered. Upon reception of a read request from a RUM, the packet and application ID contained therein are compared against the ones in the database. If they do not match, an identity theft was underway and the request is silently dropped. Otherwise the packet is requested from the network interface card and upon arrival, it is sent to the General Responder.

The I/O Controller must also adapt the packets to the data format used by the network interface card internally. In this case, a proprietary format existed for the platform for which the prototype was developed. Since this is platform-specific, it will be discussed in more detail in following chapters.

6.4.4 Summary of Architectural Properties

In summary, the architecture controls resource usage and provides protection of the system from misuse by (sub) modules. It efficiently uses the OCP interface, since bus requests can be placed while data operations are in progress, hence making good use of request multiplexing. By using a general-purpose interface the details of the resource being accessed are hidden from the application. This simplifies application design and increases re-usability in the face of changing components. It also allows the same interface and overall architecture to manage access to different kinds of resources, such as communication channels, off-chip buses and CPU. The abstraction represented by the interface is nevertheless not so high so as to obscure or mislead the functioning of the resources. It is important to note that an efficient use of the hardware resources presupposes a detailed knowledge and control of their operation. Excessive abstraction usually implies a loss of efficiency and performance in exchange for easier handling. This architecture also delivers good performance and controlled, bounded delay, which is important for most applications, that are not completely delay-insensitive. Last but not least, the proposed design scales sufficiently well with the number of applications to be useful in years to come.

In order to evaluate this architecture, a number of tests have been conducted. It is the task of the next chapter to describe the evaluation criteria set, the methods employed and the tests conducted, as well as to interpret the results and draw the pertinent conclusions.

7 EHM Evaluation

In this chapter, the EHM architecture presented in Chapter 6 will be evaluated. First, the methodology used and the goals and criteria of the evaluation will be enumerated. In order to cover a broad spectrum of possible application scenarios, two extreme case studies have been chosen, which will serve as basis for the tests. Since a meaningful evaluation of a hardware system can only be performed under consideration of the real platform to be used, a short description of the Universal Hardware Platform (UHP), developed at the Institute of Communication Networks and Computer Engineering (IKR) of the University of Stuttgart, will also be presented. Subsequently, the tests themselves will be explained, as well as the results obtained and the conclusions that can be derived thereof. A short summary of results will conclude the chapter.

7.1 Evaluation Criteria, Methods and Goals

As was presented in Section 6.2, the EHM was designed to fulfill a number of criteria. The aim of this chapter is to evaluate, to what extent that goal was achieved. First and foremost, the EHM had to provide independent QoS guarantees, especially bandwidth allocation. Second, since the safety of the code can not be guaranteed with existing mechanisms (s. Section 3.2), service isolation was of paramount importance. Third, the performance of the system has to be sufficient for present and near future communication nodes. Lastly, the EHM had to provide enough scalability in terms of the number of applications managed simultaneously.

A number of methods could have been employed to evaluate the EHM. They all present different advantages and drawbacks, which will be shortly reviewed here.

Analysis. Mathematical analysis is a powerful tool for the characterization and evaluation of communication systems. Nevertheless, because of this system's complexity, analysis could only be applied to selected portions of the overall problem. Furthermore, analysis is realized under a certain degree of abstraction from the real scenario. As will be explained shortly, that eliminated the possibility of obtaining some of the relevant information regarding the design.

Prototyping. This involves as a first step the design of the hardware system with the help of some hardware description language. Subsequently, a functional verification through hardware-close simulation would be done, to check the overall high-level correctness of the design. In order to obtain measurable test results, however, the consequent next step would have been the prototypical realization of the whole system, followed by a set of measurements. This is a very valuable approach, in that it allows to measure long-term reactions of the system and obtain results based on real traffic. There are two sets of drawbacks with this approach however, both deriving from the prototype.

The development of the prototype of a communications node is a very burdensome and complex task. As will be discussed in section 7.2, one of the principal limitations lies on the closeness or else of the prototype to real-world implementations of the architecture. Typically, small-scale prototypes are used to check the validity and feasibility of a design. They can not be used for performance measurements, since their implementation has to make a number of concessions to available technology, development effort and budget. Only a product-close, full-blown prototype, built from scratch for a specific purpose would solve these problems, at the price of higher resource usage in terms of price, time and effort. Such a prototype was out of the scope of this Thesis.

Furthermore, the test of a certain new module, like the EHM, nevertheless needs the support of an important amount of additional prototype infrastructure in order to work, diverting a big portion of the resources into non-central elements of the design. Hence, the realization of a valid prototype presented some strong disadvantages, which might be solved with the help of simulation techniques.

Simulation/Emulation. Simulation techniques rely on good modeling to concentrate on testing the fundamental characteristics of a design, hence saving time and effort. It furthermore presents the possibility of performing tests under a broad set of different scenarios with relatively low burden. The simulation of a complex system can be performed at different levels of abstraction, depending on the simulation model and the degree of detail chosen. For hardware-close works as the one described here, the chosen technology plays a fundamental role in the feasibility and performance of the design, as well as the physical constraints. *Vice versa*, the design is strongly influenced by the existing technology and its limitations. Hence, very abstract simulations, albeit very useful to validate the correctness of the design, are of little help to evaluate a hardware system, and consequently are not used here.

> One of the characteristics of VHDL and other hardware description languages, is their ability to describe digital systems at different degrees of abstraction, allowing simulations to be performed at all levels. In this way, a project can be incrementally refined and evaluated through simulation until it reaches deployment stage. At the lowest level of abstraction, the simulated design is *identical* to the real system. It is then an *emulation*. Most results presented in this Thesis are derived from those emulations.
>
> The one drawback of simulation and emulation techniques is the time-scale. Because they have to reproduce and keep track of every event happening in the model or system, they are very processing-intensive. As a consequence, the more complex the simulated design, the shorter the time slot, that can be simulated in finite time. Hence, emulation is not well suited for the long-time evaluation of systems.

In this Thesis, a *system emulation* in VHDL plus pre-prototyping (*synthesis and place and route*) has been chosen as method to evaluate the EHM. It presents a middle way between pure prototyping and pure simulation, which allows to recreate the technological limitations surrounding the system without incurring in the overhead or lack of real-life relevance of a prototype.

Nevertheless, a number of parameters can not be extracted from simulation, especially those related with the physical limitations of the circuit. The maximum system clock frequency, which influences performance by setting a limit on the cycle duration of the design, is one of them. This derives from the fact, that the maximum clock frequency depends on the number and characteristics of the elements (flip-flops, registers, etc.) present in the critical path of the circuit, as well as from the delay derived from the non-infinite speed of light in the medium. Physical resource usage is another such case: The exact number of gates needed to realize the circuit is dependent on the technology used. The same applies to memory bits, logic cells, etc. In order to obtain this information, the design has to be synthesized and placed and routed for a concrete prototyping platform, taking into account the exact architecture and components (FPGA, memory banks, buses, etc.) present. With that information, the relative information obtained from the emulation can be translated into absolute values (e.g. from the number of clock cycles per operation to the maximum packets per second that can be processed).

The most relevant information for the purpose of evaluating the goodness of the design, however, lies in the relative parameters derived from the emulation: E.g., efficiency, defined as the number of clock cycles employed in data transmission vs total number of cycles (including overhead), is in itself a relative value, and gives a much better image of the quality of the design that the absolute performance. Isolation and QoS, for their part, are more concerned with the bandwidth percentage obtained by every application in the face of overload than

with absolute bits per second. Additionally, absolute values for hardware systems suffer of an inevitable limitation: Technology dependency. While relative values reflect the architectural characteristics of a design, absolute values also have to take into account the limitations imposed by the platform. I.e., the same architecture, transported to a different platform (be it by changing the platform architecture itself or simply by substituting its components for different ones) would deliver completely different results.

Accordingly, the emphasis of this evaluation will lie on the relative results, which reflect the architectural characteristics of the design, much more than on absolute results, which necessarily have to take into account the boundary conditions set by the platform. This is not to deny, that absolute values, precisely because they reflect the limitations imposed by existing technology, are of great importance: Assuming a realistic platform, they allow to judge the usefulness of the design for a certain scenario. By taking also into account the probable evolution of the technology, the results can be extrapolated to the near future. For that reason, absolute values will also be provided. Still, a clear distinction between the effect of the architecture and of the platform in the results will be made throughout this chapter. These absolute values derive from the synthesis and place and route process, as stated before, and not from measurements on an actual prototype.

A second set of limitations in the evaluation of this design derive from the scenario itself. Since the EHM acts as a middleware between a set of applications and a set of common resources, its performance depends on both the former and the latter. As way of example, SDRAM memory performance, which will affect the EHM performance, depends on the data locality and the access pattern. The behavior of the applications themselves (which resources they use, with which frequency, using which distribution) also has an impact in the overall system performance. In order to cope with this variability, a "test case"-based approach has been chosen. As was explained in the previous chapter, two extreme examples of off-chip resources were modeled: SDRAM memory and a Fast Ethernet/Gigabit Ethernet interface. They represent not only two fundamental elements of any communication node nowadays, but also have very different characteristics, especially in terms of access granularity (a few bytes vs thousands of kilobytes) and access delay. By modeling these two extreme examples, most other cases could be represented as points in the space defined between them. Following the same approach, two very different applications were modeled. First, an accounting application, which could be used for AAA purposes in any communication node (details will be given in section 7.3). This application has to keep track of flow activity and is thus very memory intensive. Second, an encryption application as would be found in any VPN-supporting node. Encryption is a very processing and data intensive operation, since every packet flowing through the system has to be ciphered. Again, the behavior of these applications reflects two extremes, that serve to delimit the space of the feasible.

Summarizing, the evaluation of the EHM is based on two case studies, which set the limits within which a very broad palette of applications and resources would fall. The method used for it has been the emulation of the design in VHDL at the lowest level of abstraction, being the exact same design that would go into the prototype. The emphasis of this evaluation lies in showing the *feasibility* of the approach presented in this Thesis, as well as its *adequacy* to solve the problems described in previous chapters.

7.2 Platform Description

The Institute of Communication Networks and Computer Engineering (IKR) of the University of Stuttgart already disposes of a platform for rapid prototyping of network nodes, that has been named the Universal Hardware Platform (UHP, [Jun02]). That platform has served as reference for the design and evaluation of this work. In this section, its main characteristics will be examined.

Before describing the UHP, though, a number of provisos are necessary. First, the design and development of the UHP itself was not part of this Thesis and the author was not involved therein. Second, although the design presented here and used for the evaluation of the EHM was adapted to the characteristics of the UHP, it was never actually tested thereupon. All results presented in this work are based on the system emulation (described in section 7.3) and the synthesis and place and route processes, as stated in the previous section. A description of the UHP is nevertheless necessary to understand the technological and physical constraints, that were built into the system emulation to take into account the characteristics of a real prototype.

The UHP is composed of two main components:

- A set of modular, stackable boards, containing the basic elements of any communication device.
- A central VHDL-library for basic logic designs, that serve to control the hardware building blocks and perform the basic functions of any network node, e.g. retrieving frames from a card and storing them in memory, prior to being processed inside the node.

The main rationale behind the UHP, is that all communication nodes consist basically of the same elements:

- (Programmable) logic
- Processors
- Memory
- (Network) Interfaces

By providing those elements in the form of stackable building blocks, a "hardware construction kit" for network nodes could be realized.

The main system functionality is embodied in the logic design. By providing a central library of basic functions, the UHP allows the designer to concentrate on the specific extensions needed for its system. In this way, a prototype for the EHM could be designed and put together more efficiently than beginning from scratch.

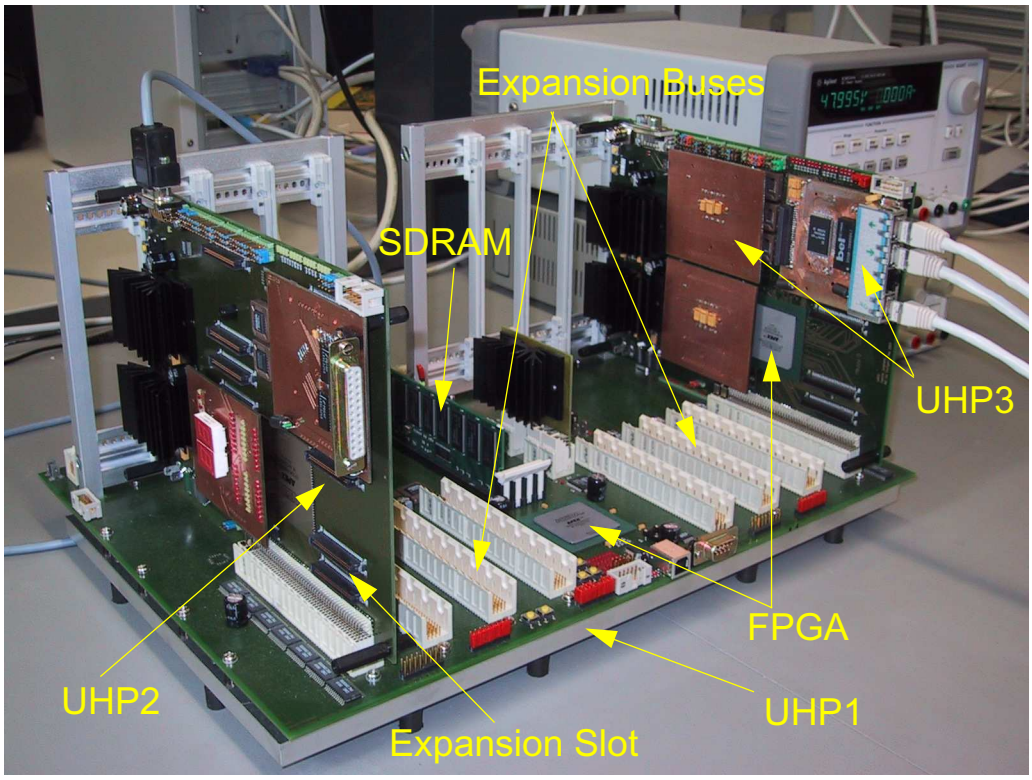


Figure 7.1: The Universal Hardware Platform.

There are basically three kinds of UHP hardware boards (s. Fig. 7.1):

- **UHP1**: Used as a central interconnecting board, it is composed of a central FPGA of Altera's APEX family [Cor], an SDRAM socket and some additional management interfaces (e.g. JTAG and serial). Additionally, it contains two buses connected to the central FPGA, with four expansion slots each, to stack UHP2 cards.
- **UHP2**: Often used as the recipient of most of the system's logic, it also contains an FPGA of the same family, as well as four expansion sockets for smaller UHP3 cards.
- **UHP3**: These smaller cards are mounted on the UHP2s and contain no programmable logic. They were designed to contain the necessary periph-

eral elements. Several of them have been produced containing additional SDRAM or SRAM memory banks, Ethernet or ATM interfaces in different flavors (e.g., optical and copper, Fast and Gigabit Ethernet), DVI interfaces, etc.

The central VHDL logic repository basically contains the modules necessary to control all peripherals (interfaces and memory) located in UHP3s. Additionally, it contains an elementary forwarding engine, that provides the basic functionality of every switching node: It retrieves frames from the interfaces, stores them in an intermediate SDRAM buffer and passes them over to an unspecified application for processing. Alternatively, it retrieves frames from the application and sends them further to the selected network card. On top of that, a number of more specialized applications have been included, which are of no interest for this Thesis.

In order to test the EHM, the emulation testbench assumes the following construction (s. Fig. 7.2): One UHP2 stacked with a UHP3 containing an SDRAM module serves to accommodate the whole EHM logic and the encryption and accounting applications (depending on the test). Additionally, it contains the logic necessary to transport the data over the interconnection bus to the central FPGA on the UHP1. That bus protocol was also developed by the author in the course of this work.

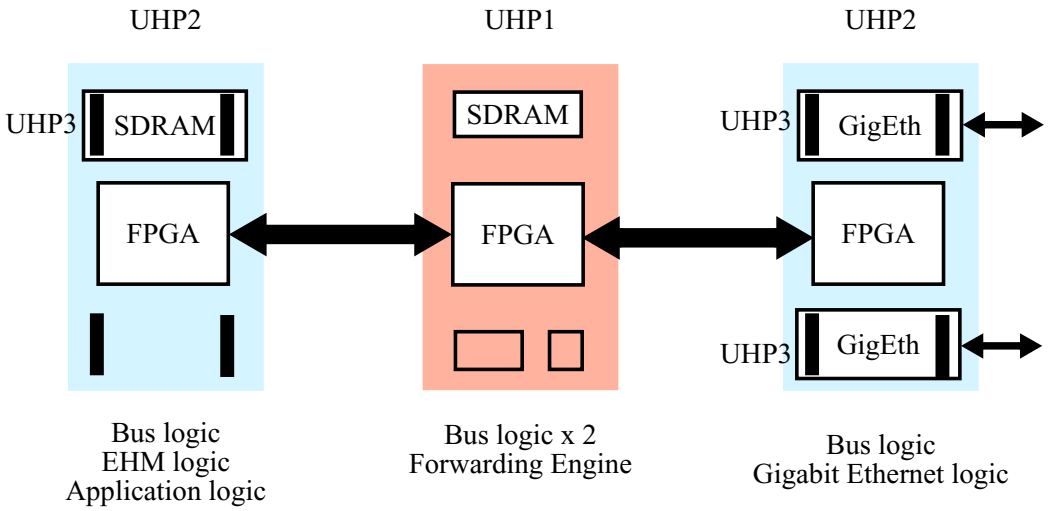


Figure 7.2: UHP Configuration for the EHM Prototype.

The UHP1 contains another set of SDRAMs, which serve as packet buffer upon retrieval from the network interfaces. Additionally, the FPGA contains the logic in charge of interfacing with both buses and the forwarding engine in charge of receiving, forwarding and storing packets to/from the network interfaces and the EHM. On top of that, a softcore processor has also been programmed for management functions.

The second UHP2 has been equipped with two UHP3s to support two Gigabit Ethernet interfaces. The central FPGA only contains the logic necessary to retrieve frames from them and pass them to the UHP1 via the interconnecting bus.

The prototype would work as follows: Packets arrive in one of the Gigabit Ethernet ports and are passed to the UHP1 for buffering in SDRAM. Additionally, a copy of the packet is sent to the EHM-UHP2 as arrival notification. After checking the availability of resources, the EHM-UHP2 requests the packet, which is retrieved from the UHP1 and processed. Packets that are generated by the applications in the UHP2 are directly forwarded by the UHP1 to the Gigabit Ethernet ports without intermediate buffering. The accounting application (s. Section 7.3.2) stores the statistics related to the individual flows in the SDRAM modules directly connected to the EHM-UHP2.

This design, then, provides all the necessary elements to test the EHM. Unfortunately, as was discussed in the previous section, there are also a number of limitations introduced by the physical and logical characteristics of the platform. First, the working of the forwarding engine, as stated, involves the sending of incoming packets to the EHM-UHP2 twice: Once as arrival notification, and a second time as "real" data. Additionally, the engine stores the packets first in SDRAM instead of fetching them directly from the Gigabit interface buffer and passing them directly to the EHM-UHP2. These two effects introduce an additional delay in the system and strongly reduce the overall performance.

Second, the presence of the system buses to interconnect the UHP2s and UHP1 introduce, once again, an additional delay. Additionally, due to physical limitations, the buses can only be driven with a maximum frequency of 50 MHz, which sets an upper limit on the overall performance of the prototype, as designed, of 3.2 Gbps half-duplex or 1.6 Gbps full-duplex. Since the bus protocol introduces a small amount of additional overhead, this has to be understood as raw capacity and a theoretical, unreachable maximum.

Third, the format differences between the Gigabit Ethernet interfaces, the forwarding engine at the UHP1 and the EHM involve a series of adaptations, that also consume resources (logic and time). Accordingly, a small degradation of the maximum performance and clock frequency and an increase in gate consumption and delay are unavoidable.

Nevertheless, as stated before, this limitations mostly affect the absolute performance of the system, but not its architectural (relative) qualities. They are related to the physical and logical characteristics of the platform and do not impede the evaluation of the EHM as such. Furthermore, they are necessary to reflect the effect and necessary concessions to technology and physics. A hardware system can not be thoroughly evaluated without taking these factors into account. The prototype, as presented here, has been tested through emulation by respecting all the physical boundary conditions examined in this section. In section 7.4, the results of the tests run will be presented and analyzed.

7.3 Case Studies

7.3.1 Encryption

Encryption is a fundamental element of modern Security Gateways (s. Section 4.2) and other networking devices, like VPN clients. With the increasing relevance of the Multiservice Internet, more and more economic transactions and sensitive information interchange happen on the network. For this reason, concerns with data and user security are mounting, and with them, the relevance of such mechanisms as ciphering.

As a case study, encryption was chosen for being characteristic of a set of applications, like video or audio transcoding, forward error correction, etc., which are very processing intensive as well as data intensive. The former means, that a set of complex operations has to be performed on the data, usually at the bit or block level. The latter stands for the fact, that usually such applications operate flow-oriented, i.e., all packets belonging to one or more flows have to be processed. Typically, important portions of the traffic have to be dealt with in this way.

The encryption standard chosen for the implementation of this case study is the Advanced Encryption Standard (AES, [Nat01]). The main reason for this choice was the high importance that AES will achieve in the future, since it has recently been approved by the National Institute of Standards and Technology (NIST) of the United States of America as the official successor of the Data Encryption Standard (DES). DES was discontinued because of security issues primarily concerning its short maximal key length of 56 bit. The algorithm used in AES is Rijndael [DR98], a symmetric iterative block cipher supporting block and key sizes of 128, 192 and 256 bit, though only a block size of 128 bit is applied when using AES. The basic principle of an iterative block cipher is to repeatedly apply a series of mathematical operations (a single such series is called round transformation or round) to the block that has to be en-/decrypted[1]. All rounds consist of the same types of operations executed in the same order, although their actual parameters may be different in each round.

A completely new implementation of the algorithm was developed for this work [HMS03]. The implementation used here only supports Electronic Code Book Mode, i. e. every single block is en-/decrypted independently of the others. En- and decrypter use a block and key size of 128 bit and thus require 10 rounds to process one block of data [DR98]. They are realized using a structure based on an inner-loop pipeline with two stages. In its present form, the design can provide a maximum performance of roughly 1.2 Gbps.

In the first test scenario (depicted in fig. 7.3), a stream of packets arrives from a packet generator simulating the IP network to the EHM. There, they are passed to the I/O Controller and finally to one of a set of encryption applications for ciphering. After encryption, the packets are sent back to the network inter-

[1] In most such ciphers — including Rijndael — there are slight modifications to this scheme.

face, represented by a packet sink. All elements in the figure were implemented in VHDL. Only the EHM modules were afterwards synthesized and placed and routed to establish their maximum frequency of operation and resource consumption, since the other elements were only required for the tests and do not belong to the original design.

In the envisioned prototype (s. Fig. 7.2), all elements left of the vertical dotted line in fig. 7.3 would lie in the EHM-UHP2. All other testbench elements represent the UHP1 and UHP2 containing the Forwarding Engine and Gigabit Ethernet interfaces, respectively. The UHP Adapter emulates the EHM interface to the UHP bus.

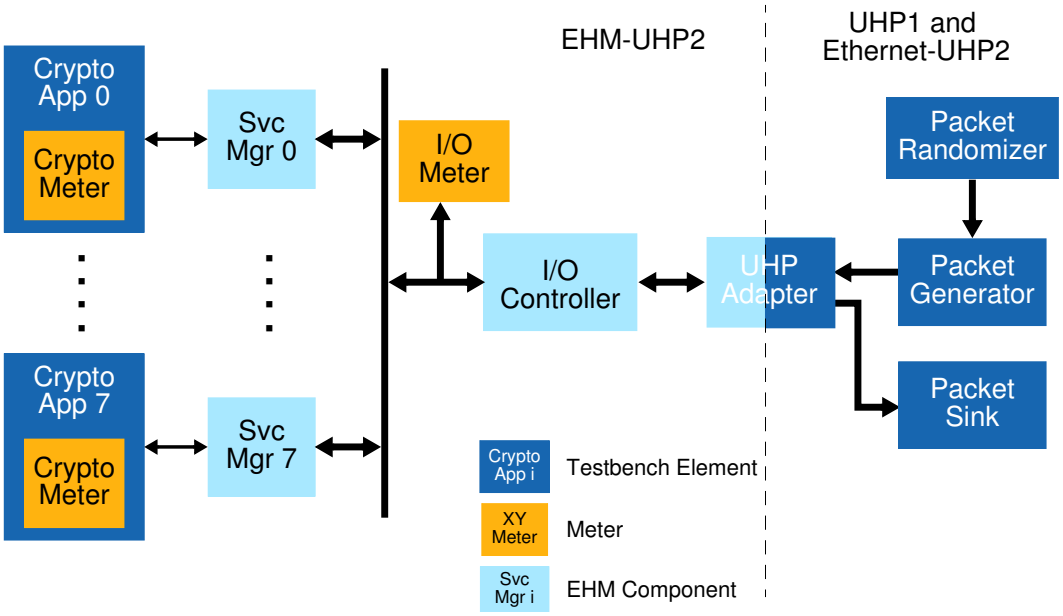


Figure 7.3: Setup of the Encryption Application Testbench.

The Packet Randomizer randomly generates values between 0 and 100 at constant time intervals equivalent to the maximum packet rate (see below). With every value, a destination address corresponding to one of the encryption applications is generated, with a probability equal to the bandwidth percentage accorded to the application. The address is given to the Packet Generator, which puts together a valid 1500-byte IP packet addressed to the corresponding application.

In the prototype (s. Section 7.2), every time a new packet arrives at the network card, it is stored in memory (external to the EHM) and a copy of it passed to the EHM. This triggers an arrival notification by the I/O Controller to the relevant Service Manager. The packet can not be directly sent to the application, because the disposability of tokens has to be checked first, in order to guarantee the QoS shares (s. Section 6.4.2.2). Should enough tokens be available, the Service Manager answers with a read request. The I/O Controller then retrieves

the packet and passes it to the application. Packets are subsequently encrypted. Before they can be sent to the network again, the Service Manager checks anew the disposability of tokens and, if sufficient, sends a write request to the I/O Controller. Upon reception of the corresponding write grant, the packet is sent out to the network card. The process is repeated for every incoming packet. Accordingly, in the test scenario, the Packet Generator first sends a copy of the packet to the I/O Controller, and upon reception of a request, re-sends it. Encrypted packets are received by the Packet Sink and discarded.

This procedure has an obvious drawback: The time penalty incurred because of the handshaking process, plus the fact, that incoming packets are in fact retrieved twice: Once when the card sends them to the EHM (which is used as an "arrival notification") and a second time when they are passed to the application. The reason for this lies in the characteristics of the prototype, as explained earlier, and is not related to the EHM itself. As a consequence, an increase in performance of approximately 50% would be possible, simply by removing this double retrieval. Nevertheless, an arrival notification of some sort is mandatory, since the availability of resources has to be checked prior to sending the packet to the processing application.

As shown in the figure, meters were co-located with the applications to log packet activity, especially arrival and departure time (Crypto Meter). Additionally, another meter (I/O Meter) was implemented and placed next to the I/O Controller, to log the arrival and departure times of every packet, as well as the interval between sending a packet notification and receiving a packet retrieval request (s. Section 6.4.2.2). Since the VHDL simulator keeps track of all events in terms of absolute clock cycles, there is a direct relationship between the simulation time and the real time, based on the clock cycle duration.

Over 10.000 packets were sent and received in all emulation runs. The number of active applications varied from 1 to 8, and the bandwidth share for every application varied from 5% to 95% of total bandwidth, as will be further explained in section 7.4.

To estimate the maximum system performance, packets were sent at increasingly higher rates until loss began to appear. This was repeated under different numbers of applications and bandwidth granularity, in order to estimate the worst-case scenario. That is the value that will be shown in the results section.

7.3.2 Accounting

One of the most critical elements for the further development of the Multiservice Internet is its economic viability. New services are increasingly introduced only, if there is a business model to back them up. Accordingly, architectures that enable the accounting and billing of service usage are mandatory for the future of networked services [SV99]. The accounting application presented here serves

additionally as a representative of memory intensive applications, like graphics, full motion video, etc.

In this implementation, incoming packets are sent to the application. Upon reception, it evaluates if the packet corresponds to an existing flow or to a new one, which is then initialized. Next, a number of information items from the IP header are extracted and stored temporarily in on-chip SRAM. Packets are not sent again to the network, since it is assumed, that the accounting application receives only a copy of it, while the original is directly forwarded to its final destination.

In parallel to that process, the application regularly retrieves the information concerning every existing flow, which is stored in off-chip SDRAM. It then updates the flow statistics with the information collected since the last refresh (and which had been stored in on-chip SRAM) and writes it back into SDRAM. The collected statistics summarize the birth and death time of every flow, the amount of bytes received, the packet size distribution, the update time distribution, the total number of packets, etc. These items are considered exemplary of the information needed to evaluate service usage, but they do not pretend to be accurate or exhaustive.

In the tests, only this second process was measured, since the goal was to evaluate the performance and behavior of the memory, hence the scenario depicted in fig. 7.4. Every application (Account App 0..7) is responsible for a small number of flows. A fixed address range has been reserved to store the information corresponding to every flow. Every accounting application retrieves the information corresponding to one of its flows from off-chip SDRAM at periodic intervals in round-robin fashion. The physical SDRAM element has been substituted by an SDRAM Emulator furnished by the manufacturer [Mic]. Next, the updated flow information is rewritten into memory. In the prototype, all modules would reside in the EHM-UHP2, except the SDRAM module, which would lie in a UHP3 (s. Fig. 7.2).

As in the previous example, meters have been placed within the applications and next to the SDRAM Controller (Acc Meter, RAM Meter). They are in charge of logging arrival and departure times of read and write requests, as well as of the corresponding data, to help calculate the system's throughput and delay characteristics.

The overall update rate was higher than the memory access rate, so that the system was tested under permanent overload. Loss could not occur, since an application could only perform an update with permission of its Service Manager, which would otherwise keep it in a waiting state (s. Section 6.4.2.1). The Service Managers logically allow for a maximum of 100% memory usage only. Since the applications could perform their updates at a higher rate than allowed, the measurements reflect the bottleneck effect by the EHM and especially the Memory Controller (SDRAM Controller in the figure).

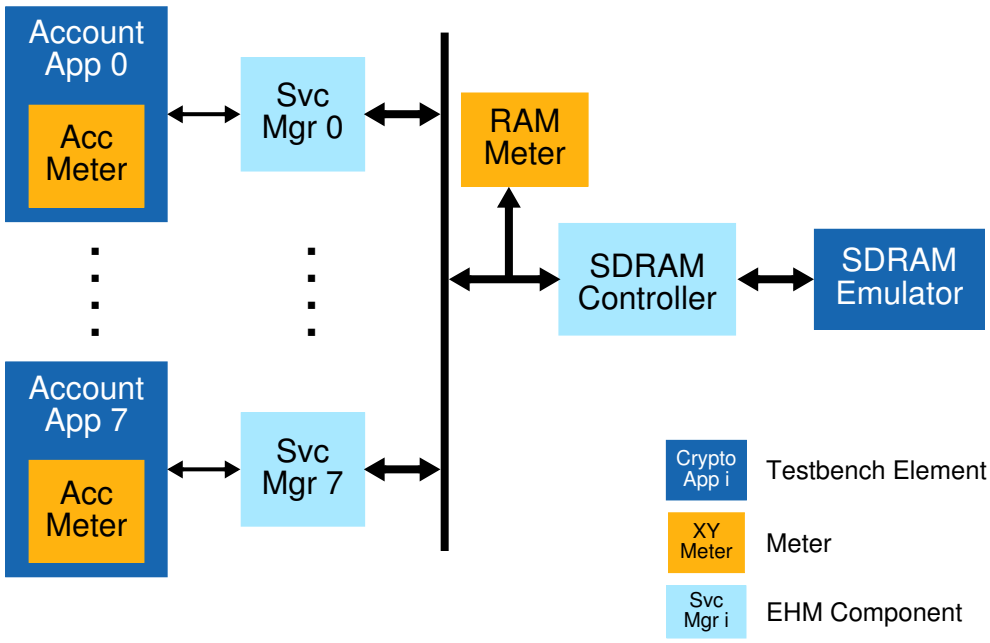


Figure 7.4: Setup of the Accounting Application Testbench.

All updates included a 16 word burst to read the information from memory, first, and an equally long burst to write it back again, second. Arguably, this deterministic access pattern has an effect in the overall efficiency attainable. Although this is true, the fact that requests from several applications are multiplexed and that their bandwidth shares are very different, together with the use of request re-arrangement within every epoch minimizes this determinism. Furthermore, since virtual memory is used to prevent interference among applications, the access locality to information stored in memory is very low. Consequently, the access pattern presented here is quite unfavorable to efficient memory usage and can thus be used as a conservative estimate.

In all emulation runs, around 125.000 updates were performed. The number of active applications varied from 1 to 8 and the allocated memory bandwidth for every application ranged from 5% to 95%.

7.4 Evaluation of the Test Results

As was stated at the beginning of this chapter, the EHM will be evaluated in terms of its efficacy maintaining QoS, its efficiency and overall performance, its ability to guarantee isolation among applications and its scalability. The following sections present the results of the tests based on the case studies explained in previous sections.

7.4.1 QoS Evaluation

7.4.1.1 Bandwidth Allocation

A first battery of tests is represented in Figs. 7.5 and 7.6. In those tests, an increasing number of applications were active simultaneously, either from the accounting type or from the cipher type. The offered load was always enough to saturate the system (overload condition). The EHM ensured a controlled sharing of the resources among the applications according to the policy set by the administrator. In these experiments, an increasingly large amount of bandwidth was reserved for a reference application, to test the reaction of the system to the number of applications and the granularity of allocated bandwidth.

Fig. 7.5 shows the results for the network interface case. Ideally, the obtained bandwidth during the measurement (y axis) should perfectly coincide with the requested bandwidth (x axis). This ideal case is represented by the 45 degree line plotted in all figures as reference. As can be seen, independently of the number of applications and the amount of bandwidth allocated, which ranges from 5% to 95%, the measurements fall within 1.5% of the reference line. The measured values are represented by the triangles with the measured bandwidth obtained presented next to them. The best linear fit for those measurements has also been plotted. As can be seen, it corresponds to an almost perfect fit of the 45 degree ideal case for all bandwidth shares. The obtained bandwidth is consistent with the requested one for any number of applications, with no noticeable variations in scale. This represents an almost ideal scenario concerning the isolation capacity of the EHM: Independently of the number of applications and the amount of bandwidth reserved by them, any reference application will always get its share undisturbed. This was one of the most critical goals of the EHM.

Fig. 7.6 shows the results for the memory interface case. Analogous data to the previous case was plotted in these diagrams. With only one noticeable exception, all values follow the pattern discussed before: They almost perfectly fit the ideal case for bandwidth allocation. One important conclusion of these diagrams regards the reaction of the EHM to different time scales. While SDRAM works with 16 word bursts, the network interface reserves bandwidth for 375 word bursts (1500 bytes, the classical length for Ethernet frames). In spite of the 20 times longer allocation time, the bandwidth is fairly allocated in all cases. Accordingly, it is safe to imply, that most application types within these two extreme cases would present the same behavior. The EHM is thus fit to serve a wide variety of different applications and off-chip resources.

It deserves to be mentioned, that all these tests were run with an inhomogeneous mixture of bandwidth shares allocated to the different applications. I.e., the available bandwidth was distributed very "unfairly" among the applications in every run. The rationale behind it was to test, if the EHM was able to keep the QoS guarantees in the face of very inhomogeneous mixtures.

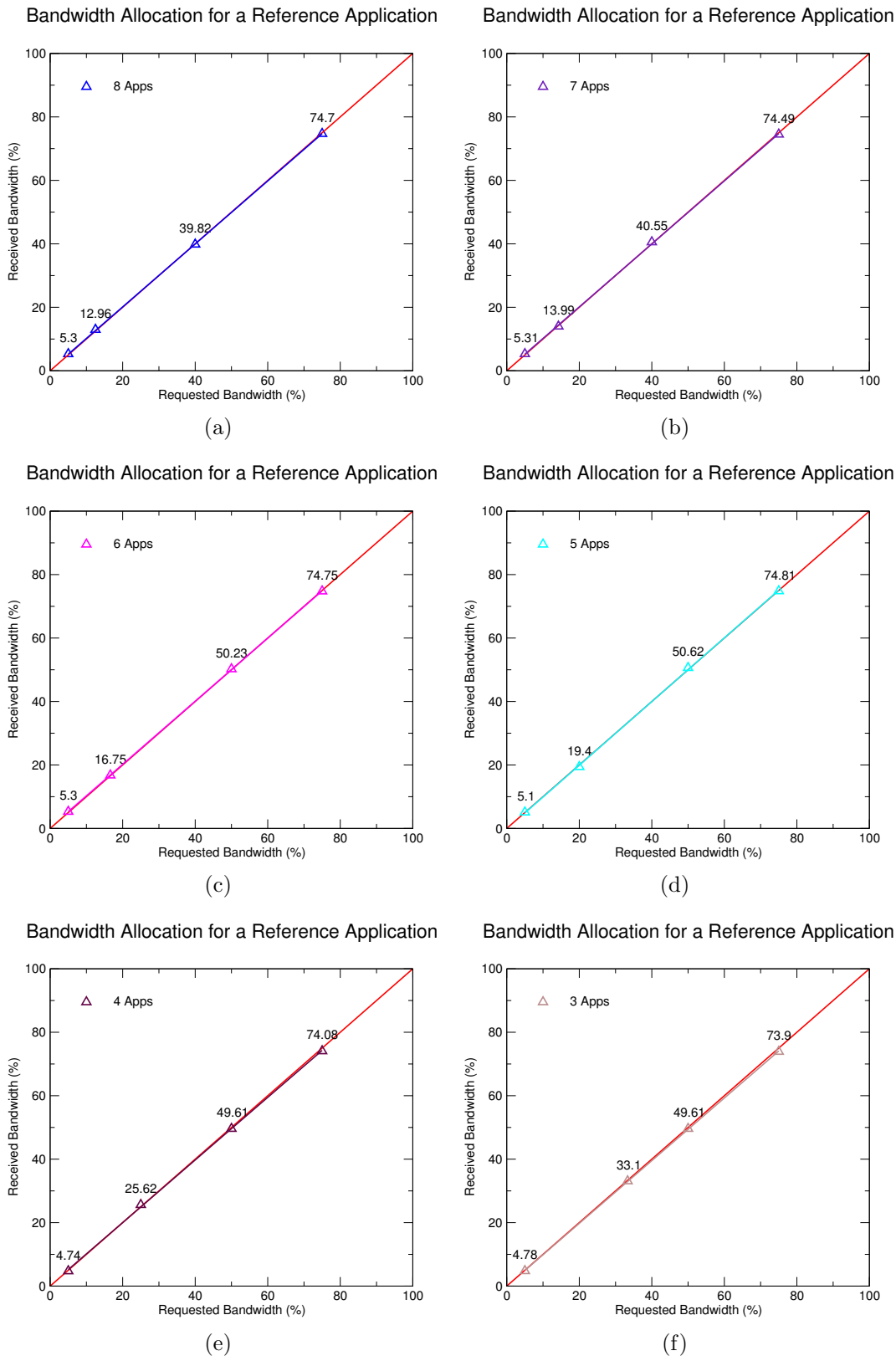


Figure 7.5: I/O Bandwidth Allocation for Different Number of Applications.

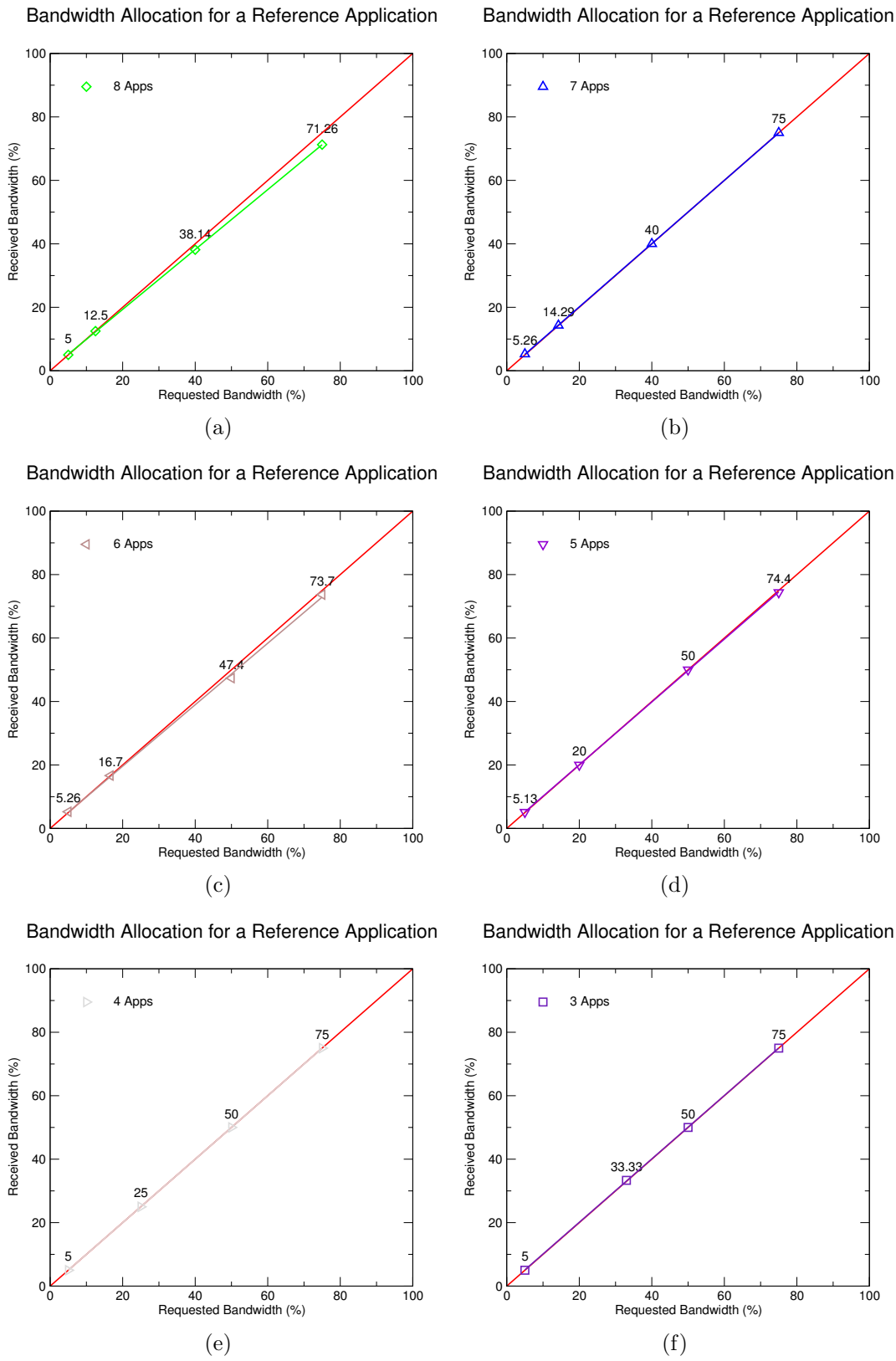


Figure 7.6: SDRAM Bandwidth Allocation for Different Number of Applications.

Tables 7.1 and 7.2 present a representative excerpt of the results for the I/O test case. As can be observed, in spite of the very unequal bandwidth allocation shares among applications (1% for an application, 75% for another[2]), the EHM maintained the QoS allocated to each with an accuracy within 1.1% in all cases.

App Nr	req (%)	obt (%)
App0	4	4.90
App1	1	1.22
App2	4	3.90
App3	4	4.17
App4	75	74.70
App5	4	4.30
App6	4	3.88
App7	4	2.93

Table 7.1: Inhomogeneous Bandwidth Distribution Among Applications: I/O, 8 Apps

App Nr	req (%)	obt (%)
App0	5	6.15
App1	15	14.97
App2	5	4.80
App3	75	74.08

Table 7.2: Inhomogeneous Bandwidth Distribution Among Applications: I/O, 4 Apps

Tables 7.3 and 7.4 show analogous results for the SDRAM case. In this case, the maximum deviation from the expected allocation falls within 0.3% of the expected value for the cases chosen. In general, all values fall within 1.5% of their target.

7.4.1.2 Bandwidth Utilization

Fig. 7.7 presents the utilization of both the I/O and SDRAM interfaces. Utilization is defined as the number of cycles used for data transmission over the total number of emulated cycles. It must be noticed, that strictly those

[2]As will be seen in section 7.4.3, 1% of the total available I/O bandwidth corresponds to roughly 4–12 Mbps, depending on the FPGA chosen and the number of active applications. Analogously, 1% of the SDRAM bandwidth corresponds to 4–20 Mbps. The scenario laid down in this Thesis foresees active applications serving large traffic aggregates (service operator model). Hence, allocating such small bandwidth fractions per application would be a contradiction. It has been used in the emulation only to test the theoretical limits on system behaviour.

App Nr	req (%)	obt (%)
App0	4	4.82
App1	1	1.20
App2	4	4.82
App3	75	74.70
App4	4	4.82
App5	4	4.82
App6	4	4.82
App7	4	4.82

Table 7.3: Inhomogeneous Bandwidth Distribution Among Applications: SDRAM, 8 Apps

App Nr	req (%)	obt (%)
App0	5	5.00
App1	15	15.00
App2	5	5.00
App3	75	74.99

Table 7.4: Inhomogeneous Bandwidth Distribution Among Applications: SDRAM, 4 Apps

cycles in which data was transported over the interface were considered in the first category. The overhead associated with the interface itself (handshaking, address transmission, etc.) has been explicitly removed from the "useful" cycles, although it is immanent to any interface. The goal was to provide an absolute lower utilization threshold, that could serve as a safe estimate of the goodness of the system.

As can be seen, the SDRAM utilization index increases with the number of applications, showing a certain multiplexing gain until it reaches its maximum for 3 applications. This effect derives from the memory scheduler itself. It was originally designed and optimized for 3 applications, hence its optimal behavior for that case. Due to the overhead incurred in the request selection process to optimize memory access, for the access pattern presented in the experiments the maximum utilization value can not be exceeded. That access pattern presents an equal share of write and read operations with very low locality, due to the use of virtual memory for the applications. Since the memory blocks allocated to every application fall very far from one another, there is no locality between requests.

Nevertheless, the maximum value (and constant for more than 2 applications) of 76.2% is a very good result for a SDRAM controller. A trivial controller, which would serve requests on a first-come, first-served basis, would reach an efficiency of a mere 40% (see reference [DL01] for a detailed discussion on the topic). This derives from the high overhead associated with SDRAM memory

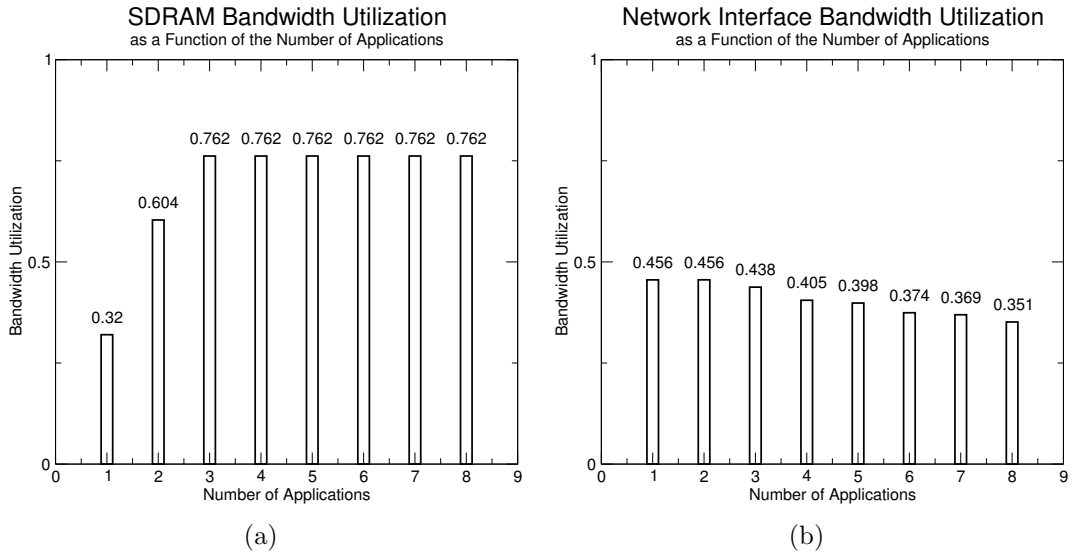


Figure 7.7: Network Interface and SDRAM Bandwidth Utilization for Different Number of Applications

preparation for access. By introducing the epoch concept and performing request rearranging within an epoch, the best possible operation order is achieved, even in the presence of low locality. Other known schedulers would reach this efficiency only in the presence of extremely high locality [DL01]. The price to pay is an additional jitter, since a request scheduled to be served can be delayed up to $(N-1) \times MBS$ cycles in the worst case, where MBS represents the Maximum Burst Size measured in cycles and N is the number of requests present in this epoch. The effect of delay will be discussed shortly.

In the case of the I/O interface, the utilization presents a much worse behavior. In the first place, it decreases with the number of applications, albeit very slowly. Secondly, it peaks at 45.6% for only two applications. This is due mainly to three factors, all associated directly with the prototype. In the first place, as explained before, there is a very costly handshake process: For every incoming packet, a notification is first sent to the service manager, so that the token disponibility will be checked. Then, a read request is generated, followed by a grant by the I/O controller. Only then is the packet retrieved from an external buffer and passed to the application. This process introduces an overhead, which increases with the number of applications active. Second, because of the construction of the forwarding engine, a packet has to be completely retrieved twice before it can be passed to the application. The main reason lies in the lack of enough on-chip memory in the UHP FPGAs to accomodate several full packets while scheduling their delivery to the applications. It follows, that by simply using more modern FPGAs and without touching the architecture, an increase of roughly 50% in the utilization would be attainable. Additionally, a less burdensome handshaking

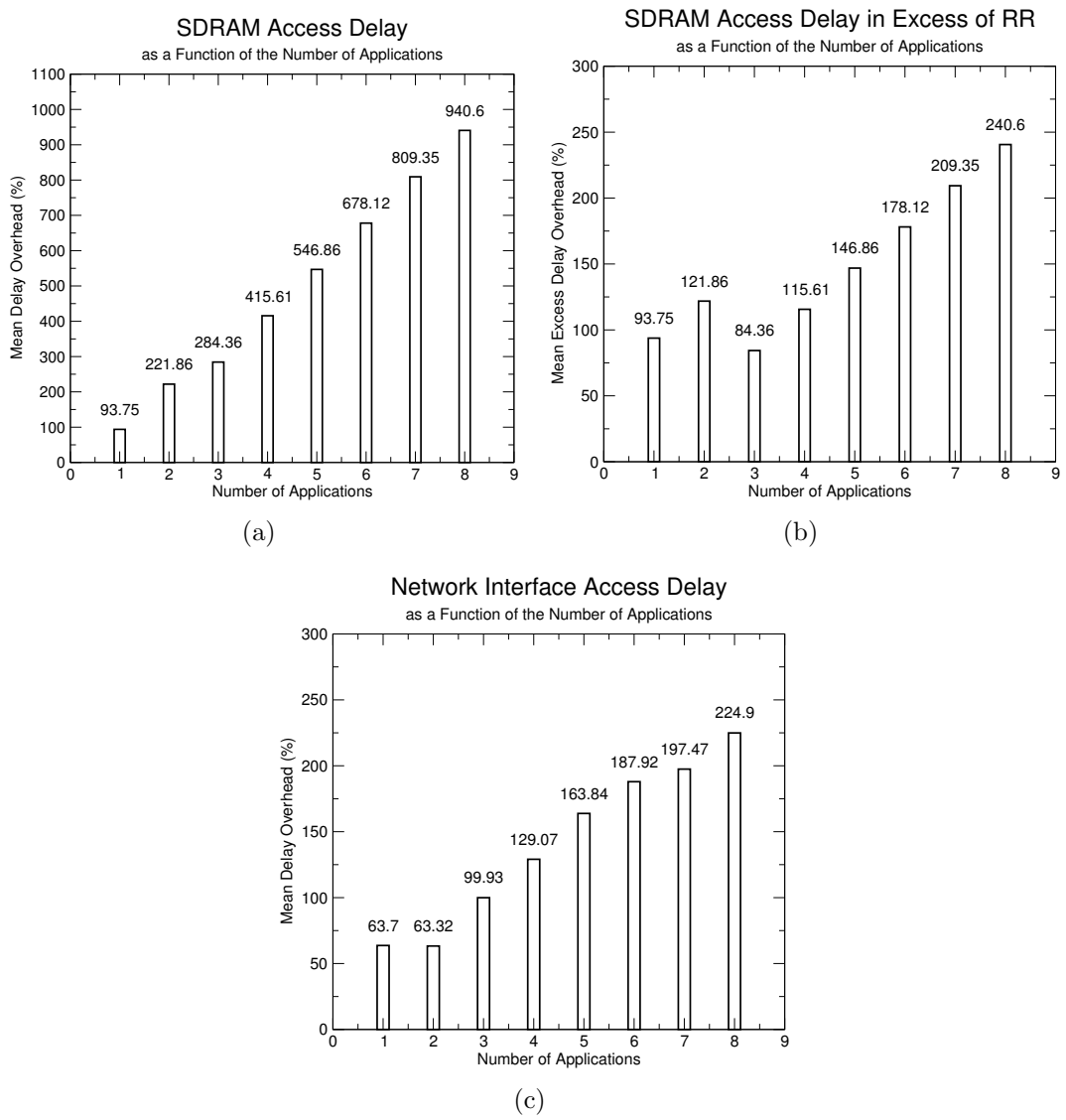


Figure 7.8: Network Interface and SDRAM Mean Access Delay for Different Number of Applications

process, based on a request-response procedure between service manager and I/O controller would allow for further efficiency increases. As a consequence, it can be expected, that with the here proposed changes, utilization would reach values closer to 70-75%. None the less, it must also be noticed, that even such a low utilization value is enough to reach a performance in the Gbps area for multiple encryption applications running simultaneously, even considering all the limitations of a prototype designed as proof-of-concept and not optimized for speed. Software solutions commercially available are still far behind.

7.4.1.3 Delay

As was stated before, in the presence of overload, delay is the price to pay in exchange for guaranteed bandwidth. Fig. 7.8 presents the results concerning SDRAM access time and network interface access time for the emulation. Plotted are the average delay overhead vs the number of multiplexed applications. Overhead is calculated as the number of cycles that it took to perform a read or a write, minus the ideal duration of the operation (which is equal to the number of words, if a word could be written/read every cycle), divided by the total number of cycles (see equation 7.1). Again, this comparison is done against the theoretical, in practice unreachable, ideal case, in order to provide a safe lower bound of the system's performance.

$$Total\ Overhead(\%) = \frac{\sum_{i=0}^{N} \frac{Cycles_i - PktSize_i}{PktSize_i}}{N} \times 100 \tag{7.1}$$

At first sight, the overhead introduced by the EHM in memory access seems huge. Furthermore, it grows above linearly with the number of applications. This must be interpreted with care, however. Delay in the system derives from two different sources: On the one side, the overhead introduced by the memory controller itself in order to schedule, rearrange and serve requests coming from the applications. On the other, from the way in which the requests are passed to the controller. As explained in section 6.3, requests are presented to the memory and I/O controller firstly in a weighted round-robin fashion, the weights being set by the state of the Token Buckets associated with every Service Manager. In the case of the SDRAM tests, greedy applications were used, which always had information to retrieve or write to memory. Under these circumstances, the minimum average waiting time for an application will be the same as the maximum and equal to $(N-1) \times MBS$, where N is the number of active applications and MBS the Maximum Burst Size. This follows from an application having to wait for every other application being served, before obtaining access to memory again. Hence, in order to test the overhead introduced by the architecture itself and not by the access pattern, Fig. 7.8 (b) presents the delay plot discounting the round-robin effect. I.e., $(N-1) \times MBS$ cycles are subtracted from the overhead calculation, to account for the "greedy source" effect.

With these provisos, the overhead introduced by the EHM through the signaling and scheduling effects responds much more precisely to the expectations. As can be seen, it grows sub-linearly with the number of applications, which is a very convenient scalability property. The *overhead* peaks at 240.6% for 8 applications. This implies, that when memory is multiplexed among 8 clients, every request takes roughly 3.5 times more cycles to be served than when only one

(ideal) application is present[3]. The relevance of this fact must be compared with the architecture of modern networking applications. On the one side, the most critical parameter in networking is throughput, not delay. On the other, however, applications frequently require data stored in memory to continue data processing. Long delays can thus have a big impact in overall performance. Nevertheless, the point has been reached, in which most networking applications would need more memory bandwidth than is physically available, even with exclusive, non-multiplexed access. In order to overcome that limitation, modern applications present a multi-threading behavior: Several strands of processing are intertwined and share a number of off-chip resources. In this way, when one thread is waiting for data coming from memory, another thread typically uses the processing resources to advance in a parallel task (s. Section 2.5.3). Under this assumption, the effect of delay on overall performance can be minimized. Still, a trade-off exists between the multiplexing effect and the delay penalty introduced, which has to be balanced for any given scenario.

An interesting effect is the delay minimum being reached for three applications. As explained before, this derives from the memory controller architecture, which was originally optimized for three applications.

The results for the network interface are summarized in Fig. 7.8 (c). In this case, the bottleneck lies in the application as well as in the network interface itself. As was explained in previous sections, packets are received twice before being passed to the applications. These, in turn, need almost 1000 cycles to process a 1500 byte packet. Since the time needed for a packet to be transmitted over the 32 bit wide network interface spans only 375 cycles in the ideal case, loss could occur depending on the traffic arrival pattern. Hence, in order to avoid loss, back-to-back packet arrivals were prevented by introducing an inter-packet gap. The maximum performance derives then from this inter-packet gap and from the double arrival effect. Under this conditions, though, it can be appreciated that the delay behavior of the system is similar to the SDRAM case. There is no round-robin effect, since the sources are not greedy, meaning that a packet has to arrive before it can be processed. The EHM-provoked overhead is strongly sub-linear with the number of applications and peaks at 224.9% for 8 applications[4].

[3]Using the performance results of section 7.4.3, with a packet size of 1500 byte and a clock frequency of 55.34 MHz (corresponding to a Stratix FPGA), the following exemplary results can be given: The minimum time needed for a 16-word burst write operation (overhead equal to 0) under such circumstances is $16/55.34\ MHz = 289.1\ ns$. Since there are 8 active applications being served in round-robin fashion, the total delay would comprise $289.1 \times 8 = 2312.8\ ns$ in the ideal case. The EHM incurs in a total delay of $(940.6/100) \times (16/55.34\ MHz) = 2720\ ns$. Hence, the delay difference between the EHM and the optimum for this example yields a mere 407.20 ns or 17.60%.

[4]Under the same conditions as in the previous example, the ideal delay incurred in sending a 1500 byte packet yields $375/55.34\ MHz = 6.77\ \mu s$. The EHM needs $(224.9/100) \times (375/55.34\ MHz) = 15.24\ \mu s$. The difference comprises $8.47\ \mu s$ per packet. In an interchange between Stuttgart and Berlin, with an overall estimated delay of some 10 ms and 50 active

This is somewhat smaller than for the SDRAM case. This effect derives from the fact, that the signalling overhead is basically constant per transaction. Being transactions in this case of longer duration, its effect is milder.

7.4.1.4 Fairness

Regarding the ability of the EHM to guarantee isolation and fairness, it was important to ensure, that the relative position of an application in the scheduling process (i.e. first, last, somewhere in between) would not affect its QoS. Tables 7.5, 7.6, 7.7 and 7.8 summarize the results in this respect. The tables are exemplary for the whole battery of tests, independently of the bandwidth distribution chosen among the applications and the number of applications. Analogous results were also obtained for the delay distributions. For clarity, only two sets of tables are presented here: For 8 and 4 applications, respectively, and only for bandwidth distribution.

App Nr	req (%)	obt (%)
App0	12.50	12.50
App1	12.50	12.50
App2	12.50	12.50
App3	12.50	12.50
App4	12.50	12.50
App5	12.50	12.50
App6	12.50	12.50
App7	12.50	12.50

Table 7.5: Effect of the Position in the Bandwidth Distribution: SDRAM, 8 Apps

App Nr	req (%)	obt (%)
App0	25.00	25.00
App1	25.00	24.99
App2	25.00	24.99
App3	25.00	24.99

Table 7.6: Effect of the Position in the Bandwidth Distribution: SDRAM, 4 Apps

In table 7.5, eight accounting applications were simultaneously active. An equal share of the overall bandwidth was allocated to each, in order to test the effect of the position within the scheduling process. As always with memory tests,

routers in the path, this additional delay would imply an overhead of $50 \times 8.47\ \mu s = 423.5\ \mu s$, which amounts to only 4.2% of total delay.

the system was running in overload. The table presents the required bandwidth (the theoretically allocated share) and the obtained bandwidth according to the measurements. As can be seen, a perfect match was obtained for all applications. The test with four applications also sharing equal bandwidth portions, reflected in table 7.6, presents the same pattern.

App Nr	req (%)	obt (%)
App0	12.50	13.00
App1	12.50	12.96
App2	12.50	11.27
App3	12.50	13.17
App4	12.50	12.22
App5	12.50	13.51
App6	12.50	11.68
App7	12.50	12.20

Table 7.7: Effect of the Position in the Bandwidth Distribution: I/O, 8 Apps

App Nr	req (%)	obt (%)
App0	25.00	26.00
App1	25.00	24.32
App2	25.00	25.62
App3	25.00	24.06

Table 7.8: Effect of the Position in the Bandwidth Distribution: I/O, 4 Apps

Table 7.7 summarizes the results for eight encryption applications sharing access to the network interface. The system was running at the maximum performance without loss. As can be observed in the table, the values are somewhat further from the ideal than in the previous example. This is due to the coarser granularity of allocation, since packets can not be interrupted or segmented when in transmission, even if tokens have been exhausted. Larger deviations are then possible. Nevertheless, no deviation exceeds 1.3% of the theoretically allocated bandwidth, no matter which number of applications and bandwidth distribution pattern chosen.

A second interesting observation is, that in spite of this slight variations in bandwidth, the relative position within the scheduler plays no definite role. In other words, there is no recognizable bias e.g. toward the first or the last applications. Fairness is thus maintained.

These conclusions are corroborated by the results of table 7.8.

7.4.2 Resource Consumption and Scalability

One key concern in the design of the EHM was its scalability. Conversely, in order to prove useful, the EHM had to show a moderate consumption in terms of FPGA resources (logic elements, pins, RAM bits), so that a sufficient number of complex applications could fit onto the chip. The results of the synthesis and place and route processes are summarized in tables 7.9 and 7.10.

Device	APEX 20K1000EPC652-1	STRATIX EP1S80B956C6
Logic Elements	8169/38400 (21%)	6741/79040 (8%)
RAM Bits	121224/327680 (36%)	108936/7427520 (1%)
I/O Pins	155/488 (31%)	155/691 (22%)
f_{max}	37.39 MHz	52.68 MHz

Table 7.9: Resource Consumption for the APEX and STRATIX FPGAs (3 Apps).

Table 7.9 shows the EHM resource usage for an Altera APEX FPGA [Cor], which powers the UHP1. However, since the inception of the UHP, two further FPGA generations have emerged, which shows how fast these technologies are evolving. The next version of the UHP will very probably be based on the newer Stratix family, or in the recently commercialized Stratix II [Cor]. In order to give a glimpse of the EHM's performance in a more up-to-date prototype, resource usage for the Stratix family is also included. This comparison also serves to highlight the effect, that technology has on performance, for exactly the same architecture.

Two different EHM configurations were synthesized. The minimum meaningful EHM configuration comprises the two resource managers (I/O and SDRAM), plus three service managers (i.e., support for three applications). For less than three applications, the EHM represents an excessively complex solution. These results are summarized in table 7.9. Additionally, table 7.10 summarizes the resource consumption for the maximum configuration deemed appropriate: Both resource managers plus support for eight applications.

Device	STRATIX EP1S80B956C6
Logic Elements	12721/79040 (16%)
RAM Bits	282376/7427520 (3%)
I/O Pins	155/691 (22%)
f_{max}	55.34 MHz

Table 7.10: Resource Consumption for the STRATIX FPGA (8 Apps).

The number of necessary I/O pins are set by the nature of the off-chip resource interfaces, and is therefore constant and independent of the EHM as such. The number of used on-chip RAM bits, on the other hand, were a cause of major

concern. They are used as intermediate buffers in the service managers and in the resource controllers. Accordingly, they increase strongly with the number of supported applications. The APEX family contains a reduced number of embedded memory blocks, which represented an important limitation for the prototype design. Hence, only the minimum EHM version was fitted into it, leaving the more demanding one for the Stratix device.

As can be observed, logic cell consumption and RAM bit usage are negligible in the Stratix case, even for the maximum EHM configuration. Well over 80% of logic resources and over 97% of memory resources can still be allocated to applications. Even for the older APEX device, roughly 80% is still free for other designs.

As way of example, the resource consumption of the encryption application used in the tests has been summarized in table 7.11. As it is, seven such applications would easily fit in the Stratix device together with a full-blown EHM design.

Device	STRATIX EP1S80B956C6
Logic Elements	8770/79040 (11%)
RAM Bits	25008/7427520 (0.33%)
I/O Pins	185/691 (27%)
f_{max}	103.95 MHz

Table 7.11: Resource Consumption of the Exemplary Encryption Application for the STRATIX FPGA.

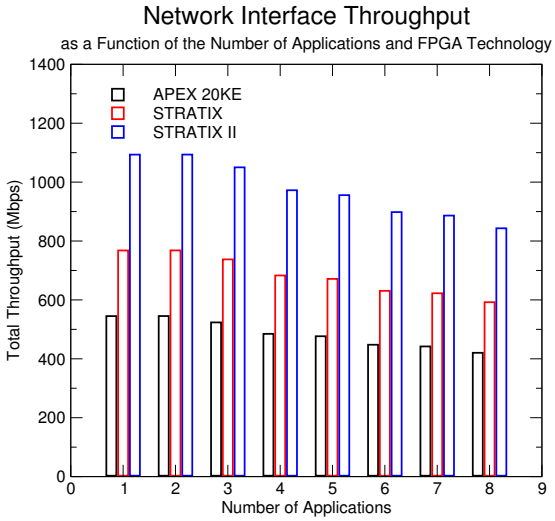
Hence, it can be derived, that resource consumption is not a limiting factor for EHM scalability, at least up to (and even beyond) eight simultaneously active applications sharing the chip.

7.4.3 Performance Evaluation

Overall system performance is determined by three factors: Circuit efficiency (utilization, ρ), maximum clock frequency and internal bus width (see equation 7.2). These factors, for their part, are determined in part by the design itself. However, the maximum system frequency is strongly dependent on the technology used. As was explained in the previous section, two further FPGA families have appeared since the inception of the UHP. In order to provide a more accurate picture of the performance possibilities of the EHM, data has also been included for both Stratix and Stratix II families, on top of the APEX device. All results are based on the emulation tests run, plus the information concerning maximum frequency obtained in the synthesis, place and route process.

$$Throughput = \rho \times Word\,Size \times f_{max} \tag{7.2}$$

The performance gain related strictly to technological evolution can be observed by comparing the maximum frequencies obtained for the APEX and Stratix devices (see tables 7.9 and 7.10): An almost 50% frequency increase. At the moment of writing, the IKR does not dispose of the libraries necessary to synthesize the design for a Stratix II device. Hence, for completeness, a further 50% increase in frequency between Stratix and Stratix II has been extrapolated, following the previous observation and the information given by the manufacturer [Cor]. As a consequence, the performance results for the Stratix II given in this section assume a maximum frequency of 83.1 MHz, 50% more than the 55.34 MHz obtained in table 7.10.



(a)

# Apps	APEX (Mbps)	STRATIX (Mbps)	STRATIX II (Mbps)
1	545.15	807.52	1212.59
2	545.24	807.52	1212.59
3	523.52	775.64	1164.73
4	484.75	717.21	1076.98
5	476.54	704.81	1058.36
6	447.74	662.31	994.54
7	441.90	653.45	981.24
8	420.37	621.58	933.38

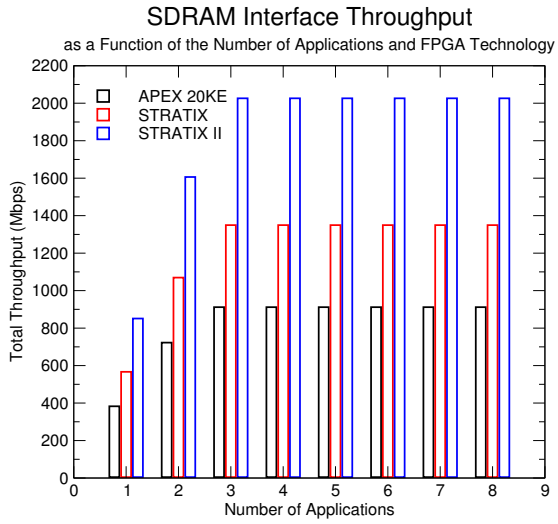
(b)

Figure 7.9: Overall System Performance: The I/O Intensive Case.

Performance results, expressed in absolute bps, are summarized in figs. 7.9 and 7.10. The former represents the tests for the encryption applications accessing the network interface, while the latter gathers the information of the accounting applications interacting with off-chip SDRAM. Both figures present a graphical

comparison of the three FPGA families, as well as the exact numerical values obtained for all tests.

As could be expected, in both cases the performance as a function of the number of applications follows the value of ρ, presented in section 7.4.1, s. Fig. 7.7. In the I/O case, throughput peaks at roughly 1.2 Gbps for two applications and the Stratix II family, and almost 1 Gbps for eight applications in the same case. Results for the APEX device are logically more modest, with only 420 Mbps for eight applications. As explained in section 7.4.1, a number of optimizations are possible, which would draw utilization into the 70-75% range without affecting the overall architecture. In that case, the APEX results would increase to some 600 Mbps in the eight application case, while the Stratix II results would top at 1.5 Gbps in the same configuration. This represents almost an order of magnitude more than existing software solutions, even for this proof-of-concept prototype[5].



(a)

# Apps	APEX (Mbps)	STRATIX (Mbps)	STRATIX II (Mbps)
1	382.87	566.68	850.94
2	722.67	1069.61	1606.16
3–8	911.72	1349.41	2026.31

(b)

Figure 7.10: Overall System Performance: The Memory Intensive Case.

In fig. 7.10 the results for SDRAM access have been plotted. As can be observed, up to 2 Gbps can be achieved, thanks to the better utilization of this

[5]For comparison, the results from recent Rijndael performance tests realized on an otherwise completely idle 1.8 GHz Pentium 4 running *one* instance of the GnuPG crypto engine (version 1.2.2) and the OpenSSL crypto library (version 0.9.7b), both widely used in the Linux environment, yield 241.83 Mbps and 375.43 Mbps, respectively [Dev].

resource controller. It must be noticed, however, that a 32 bit interface controller was used internally. Would a 64 bit memory interface have been used, as is commonplace nowadays, all results would simply double.

Summarizing, the results presented here are quite modest for the APEX device. This derives, on the one hand from the design itself, which was envisioned as feasibility study and was therefore not optimized for speed. On the other, though, a very strong technological effect can be identified. By comparing the performance reached with different FPGA families available on the market, the potential of the EHM is unveiled, showing much better results, which lie an order of magnitude over existing active networking software solutions. Such results, in the Gbps area, allow to contemplate the use of the EHM at the network edges and even in the MAN core.

7.4.4 Security Evaluation

As was explained in section 3.2, the opening of network nodes to hardware programmability brings a forgotten set of hardware threats to new relevance, which traditional security infrastructures are not prepared to cope with. One of the aims of the present work was to explore the possibilities to control and prevent such threats by implementing resource management directly a the hardware level. The emphasis was set on the threats that Mobile Code implies for an otherwise well-behaved node and on which measures could be taken to guarantee, that its safety will not be compromised. The safety of the code itself inserted in an active node can not be guaranteed by existing means. Basically, the additional mechanisms introduced at the hardware level are access control techniques and scheduling.

In the first category fall such techniques as virtual memory. Every application uses a number of bits to code the addresses, where its information shall be stored. The physical memory present in the system, however, is much larger than the sight that the application has. The Service Manager prepends an offset to that address, so that reading or overwriting information concerning another application is not possible.

For the network interface case, it would be in principle thinkable, that an application request incoming packets addressed to another one, e.g. in order to perform a denial of service attack. That would amount to an identity theft. This possibility has been excluded, since the I/O controller receives information from the packet classifier, as to which application has the right to request a certain packet. That information is stored together with a packet identification. Upon request, an identity check is performed before the packet is passed. An application can furthermore not pretend to be another one, since the Service Manager has a hard-coded identifier, which is used indirectly to identify the application. Since the application has no interface to that information, that form of identity theft is excluded.

Other forms of application interference are also prevented. Through the use of point to point interfaces between the applications and the EHM, it is physically impossible to disturb the communication of other applications. Within the EHM, the token buckets together with the round-robin scheduler ensure a guaranteed bandwidth allocation with known and bounded delay.

Still, it would be possible for an application to request access to a certain resource, for which it has the right and enough tokens, and once granted, try to overuse it by no relinquishing it in due time. That option is also eliminated through the introduction of Maximum Burst Sizes (MBS). For every operation, a Maximum Burst Size, measured in clock cycles, is given. A timer starts to count as soon as a grant was given to the application. Independently of the application's behavior, if the timer reaches the value MBS, the transaction is immediately aborted and the interface to the resource is freed. Additionally, MBS tokens are subtracted from the application's bucket, in order to reflect the real usage. Overuse is thus not possible beyond the MBS limit, and real usage instead of announced usage is accounted for.

Additionally, the architecture foresees the surveillance of all chip I/O pins by the EHM. This means, that no application would have the right to directly connect to an I/O pin, *de facto* bypassing EHM's control in access to off-chip resources. This can be easily prevented, since the integration of new applications in the overall chip design happens in the NAN, under direct control of the network operator. Such surveillance prevents a broad set of attacks at the electrical signal level, as explained in section 3.2.

Summarizing, the mechanisms introduced in the EHM allow for an efficient and complete isolation among applications. This protects applications from each other and from the system. However, the system also has to be protected from the applications. Guaranteed enforcement of resource usage agreements is also provided, due to its importance not only for isolation but also for accounting. To a degree, unexpected service behavior is also accounted for, in that resource overuse is immediately penalized. However, no system is bug-free and no claim to absolute security can be taken seriously.

7.4.5 Summary of Results

This chapter has described the tests performed on the EHM design, as well as the rationale behind them. It has checked those results against the design goals presented in previous chapters. One of them was the desire to provide service isolation, among applications but also between the applications and the platform itself, for security reasons. As has been discussed, QoS guarantees, but also resource usage limitation are enforced by the EHM. Analogously, the system itself quickly reacts against unauthorized or unexpected behavior, cutting off access to the shared resources by the misbehaving application.

The design is in a position to deliver a throughput level, which active networking solutions had not reached before in such a shared and open environment. In spite of this, delay has shown to be a problem for such applications or resources, which have very stringent delay requirements on top of high throughput requirements.

Technology, as well as the prototyping platform itself, has a very strong impact on the overall absolute system performance. In order to separate the technological effect from the intrinsic characteristics of the design, a comparison for different FPGA families was performed. This comparison showed the capacity of the design to perform in the Gbps area and thus keep up with the necessities of modern networking.

Resource usage was very moderate in all cases, falling even to negligible values in more modern, bigger FPGAs. As an overall conclusion, this chapter has shown the feasibility and the adequacy of the proposed solution to deal with the goals set at the beginning of this Thesis.

8 Conclusions and Outlook

This Thesis started with the observation, that the Multiservice Internet is suffering under its own success. The increasing diversity of services being offered over it, and the sheer number of service instances simultaneously active are a double scalability problem. As a result, the management of modern communication networks is increasingly complex and costly. A trade-off exists between the desire for network flexibility, performance needs to cope with traffic demands and management complexity.

Several proposals have emerged to overcome this trade-off. The one that serves as inspiration for this work is the research in the area of Active and Programmable Networks (A&PN). The principle behind A&PNs is to transform the network from a mere bit-transporting pipe into a service-aware open programmable platform. The main assumption behind it is, that some services are better realized in, or at least supported by, the network itself. In spite of considerable effort, however, these approaches have faced severe problems: There seems to be a trade-off between security and openness and between flexibility and performance. The goal of this Thesis was to sketch a novel network model that would overcome those trade-offs, providing performance, flexibility, openness, security and QoS simultaneously, with acceptable complexity.

This work begins with a novel taxonomy and characterization of router tasks, mechanisms and architectures in chapter 2. This new analysis of the role of routers in modern networks was made necessary by the expansion in router functionality in the last years. Especially the introduction of traffic management tasks (basically, resource management and security) has had an tremendous impact on router design and architectures. The increased router complexity has also driven to specialization, with the appearance of a wide variety of router-like networking devices like security gateways, firewalls, SOHO routers, etc. In parallel, a strong technological evolution has taken place, with an increasing reliance in hardware support to achieve sufficient performance. Programmable hardware is playing an important role in overcoming the trade-off between flexibility and performance.

Chapter 3 explored the possibilities that A&PN opened for the future of router design, as well as its main limitations. One of the most critical problems derives from the emphasis on openness and management by third-parties. This complicates the achievement of sufficient security levels. The introduction of

programmable hardware presents new risks, for which traditional proposals, based mainly on pure software solutions, are not prepared to cope with. QoS, on top of that, is a necessary element of any modern router, and its main principles and approaches were also reviewed in this chapter.

Chapter 4 presented the author's vision of an overlay network model able to overcome the limitations and trade-offs mentioned above. It begins by acknowledging the necessity of strong security checks in the form of access control, AAA, encryption, sandboxes, etc. However, in order not to endanger performance and flexibility in exchange for security, it clearly separates service admission and introduction, which is subject to strong security checks, from service management and operation, which is not. Once a service has been deployed, only run-time checks are applied, which critically improves performance with respect to other proposals. Furthermore, performance is also underpinned by the introduction of not only programmable software, but also programmable hardware environments in the network nodes. Flexibility is sustained by the programmability of the platform, which extends to the control plane: Every service operator is allowed to implement its own management interfaces and protocols. The key to achieve flexibility without compromising security at the system level lies in a strong isolation among services and between every service and the node platform itself. In software, that isolation is reached through well-known methods: Sandboxes, Execution Environments and the Operating System. In hardware, those mechanisms are not adequate, for their view of, and hence their control over embedded applications is too coarse. In order to prevent threats coming from the hardware part of the applications, resource management has to happen at the hardware level, too.

As a consequence, chapter 6 presents the Embedded Hardware Manager. the EHM provides isolation by acting as intermediate element between the applications and the off-chip resources, as well as among the applications themselves. It acts as scheduler, QoS manager and security agent for native hardware applications multiplexed on a common platform. The EHM can work with a wide range of applications and off-chip resources, as long as they have no critical delay requirements in the ns domain. The EHM has been tested by means of two extreme case studies, which serve as boundaries for a broad set of scenarios. The test cases involve a processing and data intensive application, on the one hand (encryption) and a memory intensive application, on the other (accounting). In order for the tests to be complete, the corresponding resource managers for off-chip SDRAM and a Fast Ethernet/Gigabit Ethernet network interface have been implemented. These two case studies present very different characteristics in terms of operation time scale and granularity of operation, besides representing applications and resources ubiquitous in modern communication systems.

The EHM has been evaluated in chapter 7 by means of a system emulation. Additionally, a prototype has been designed, in order to consider the physical constraints set on the architecture by a real platform. In this way, relative as

well as absolute results could be obtained. Isolation, scalability, QoS allocation in terms of bandwidth and delay, resilience to security threats and overall efficiency were measured, as well as overall performance (processed packets per second). As documented in chapter 7, the results are very satisfactory in all areas, proving the feasibility and adequacy of the approach presented in this Thesis. Nevertheless, a number of restrictions also apply. First, delay is the price to pay in exchange for isolation and QoS guarantees. This disqualifies the EHM for use with extremely demanding resources in terms of maximum delay, like SRAM. Second, the platform presented a number of inadequacies, which also curtailed the performance achievable by the prototype. Third, the goal of the architecture being to serve as proof-of-concept, no attempt was made to optimize its implementation for performance. Undoubtedly, further optimizations in the implementation would be possible, without noticeably altering the overall architecture. Still, the performance lies in the Gbps range, which is much better than previous software proposals. Resource usage is moderate and even small for the latest FPGAs.

And yet, there is still a long way to go to realize the vision of the Octopus Network Model. Only an sketch of its principal elements has been attempted here. The full realization of the OOG is still pending, comprising both its software and hardware parts. Both exist as separate instances: The EHM and associated hardware platform has been presented in this work. Previous proposals, like Darwin, could be adapted to serve as the software part. The integration is the missing link.

The completion of the NAN comprises two different strands of work: On the one side, the realization of the interface to the code repositories, including authentication, authorization, etc., seems more a matter of implementing existing solutions than of research. On the other side, however, the automation of the service inclusion in existing designs implies an important amount of research in tool support for system design.

The negotiation protocol between the user and the network is also a pending issue, although existing signaling protocols offer a very good starting point. A true explosion of signaling in IP networks for all kinds of uses and services can be observed recently, with SIP and RTCP being only two examples.

As a last word, the dangers of openness remain a critical issue. In spite of hardware resource management and software sandboxes and other techniques, a large amount of real-life experience has to be summoned, before the fears of the network operators can be safely put aside.

This Thesis tries to lie the groundwork for the successful realization of an open, secure programmable overlay network for the future Multiservice Internet.

A Review of Some Relevant Active Network Proposals

A.1 TU Braunschweig / Universität Karlsruhe: FHiPPs and AMNet

The AMNet project was initiated at the TU Braunschweig under the direction of Prof. Dr. Martina Zitterbart. When she was called to the University of Karlsruhe, the project moved along and was continued there, where it was further developed. AMNet was initially an Active Networking approach to better support multicast in IP networks [WZ98]. However, the project quickly evolved into a Programmable Network infrastructure for general functionality enhancement [FHSZ02]. In its early version, it also comprised a programmable hardware board for the support of processing-intensive tasks [HMP00], but in later publications it seems to have lost relevance, converging to an all-software proposal. In fact, the authors clearly state their goal to develop nodes, which will be placed at the network edges or even outside the data path. The idea in this second case would be to redirect packets, that need special treatment to those nodes by influencing the routing decisions at the edges. The marginalization of the hardware support, on the one side, and the decision to move away from the main data path, on the other, show a renounce to achieve high performance, one of the critical success criteria for Active Networking presented in previous sections. In fact, their own numerical results [FHK+03], present a maximum achievable throughput of 68 Mbps for a simple exemplary application.

AMNet relies on a DNS-like structure of code repositories for the distribution of applications. Authorization, authentication and signature mechanisms are also foreseen for increased security and integrity. Contrary to other approaches, this proposal renounces explicitly the use of virtual machines for performance reasons, relying on native code instead. The compiled application, thus, resides in the repository. To account for OS and hardware diversity, several versions of the code might be developed and stored in the servers.

To account for the security issues associated with downloading native code, two main elements are introduced. On the one hand, a resource negotiation takes place between the node and the repository prior to download. In this phase the

resource needs of the service and the resource disposability at the node are compared, as well as the rights associated with the new application. If the code is admitted, a runtime access control infrastructure surveils the application's resource usage [HSS+02]. Basically, this infrastructure is placed as an intermediate layer between the application and the OS. All system calls issued by the service are intercepted and compared against a resource monitoring database. Should an application exceed its associated resource limits, the call is rejected. The three resources surveilled are bandwidth, processing power and memory space. For all other checks, like access to memory space associated to other applications, AMNet falls back on the UNIX access control infrastructure, which is used as the underlying NodeOS.

Arguably, the most novel elements of their architecture are related with their hardware platform, known as the FHiPPs (Flexible High Performance Platform, [Har02]). It is composed of four main elements, depicted in Fig. A.1: A DSP board, an ATM interface, an embedded CPU and a set of FPGAs interconnected by a switch fabric. These elements are located in two boards attached to a host via the PCI bus. Because of the diversity of reconfigurable equipment, the FHiPPs is suitable for the efficient support of very different applications. None the less, this same heterogeneity makes the platform complex to program and manage, thus presenting a trade-off between flexibility and usability.

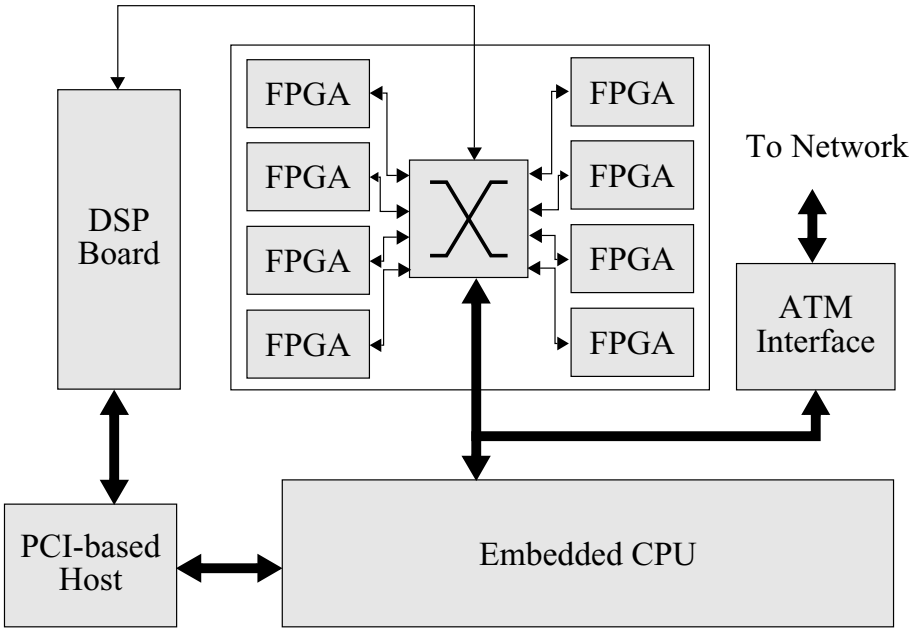


Figure A.1: The Flexible High Performance Platform (FHiPPs).

A very interesting component of the architecture is related to the interconnection between the hardware platform and software running on the host PC. In principle, the reprogramming of the FHiPPs should present a new API (realized

as a device driver) to the software on the host, in order to efficiently use its new capabilities. In order to support such "dynamic device drivers", the Happlet concept was developed, which stands for "Hardware Applet". The introduction of new functionality in the hardware platform (including the corresponding driver) is done by using this mechanism.

A pseudo device interface is permanently installed in the host machine, and serves as multiplexing point to access the hardware. Associated with it, there is a hardware manager, which provides basic functionality to integrate new happlets, initialize them and remove them, if needed. A Happlet consists of two main parts: A host code segment, which consists of the driver itself, plus indications for the introduction and configuration of new code into the hardware platform. The second part is called the hardware code segment, which contains the DSP, FPGA or CPU code itself (or a mixture thereof for more complex applications, s. Fig. A.2). In this way, the introduction of new functionality and the introduction of the communication mechanisms between that hardware functionality and the control software is realized in one coordinated step. Happlets also reside in the code repositories and are downloaded from there exactly like software plugins. In order for other (software) services to communicate with the hardware in the AMNet context, a pseudo-application is instantiated in the host, which serves as "software-representative" of the hardware platform. Other services can thus communicate with this application through normal channels (e.g., sockets) for the purpose of interchanging information or packets. The pseudo-application, then, sends the relevant data to the corresponding Happlet via the pseudo-device interface.

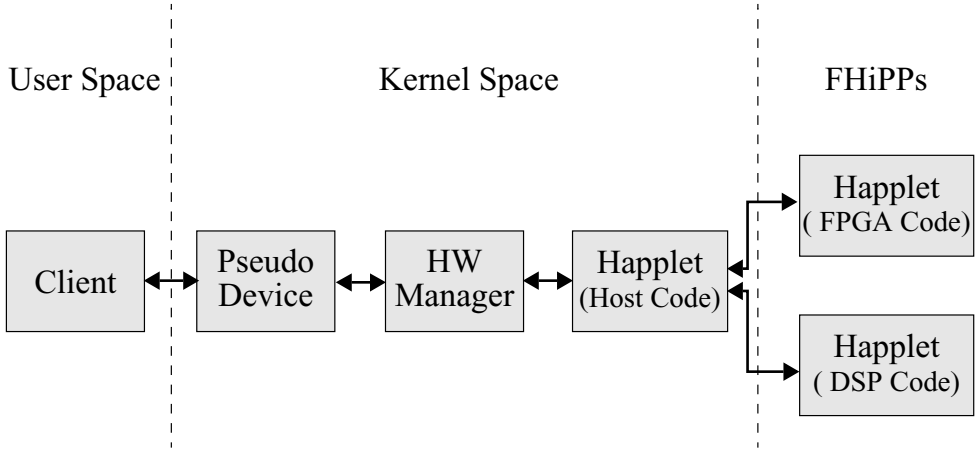


Figure A.2: The Hardware Applet Concept.

In spite of its undoubtful merits, the AMNet initiative presents a number of weaknesses. The first affects the use of native code. Although an access control infrastructure is in place, it is restricted to checking the total amount of resources used by every application, with the goal of avoiding overload situations.

A number of other attacks are possible, as stated above, against which only the default UNIX mechanisms are used. Furthermore, since applications run outside a virtual machine, code bugs can still drive *all* active services down, even if they do not make the node crash. But the main drawback of using native code is associated with the Happlets. Since device drivers are dynamically installed in the OS kernel, a much more critical threat exists: A bug in this code could indeed bring the whole node to a halt by provoking an OS crash. Against this possibility no protection has been devised.

The second set of threats are directly associated with the FHiPPs platform. Since AMNet assumes a centralized node management and the correctness of the code installed therein, no runtime checks are performed at the hardware level. As explained in Section 3.1, a number of attacks are therefore possible. These threats are indeed mentioned in [Har02], but no access control or resource management architecture to prevent service interaction at the hardware level is deemed necessary.

Summarizing, the AMNet approach has developed a very flexible software and hardware platform for Active Networking. Nevertheless, the resource management and security parts are somewhat insufficient, especially at the hardware level.

A.2 Work at the Washington University in St. Louis and the ETH Zürich

At the Washington University in St. Louis, a number of mutually related projects have been conducted in the last ten years. Their main common goal was to investigate network node architectures, that would allow simultaneously high performance (in the multi-Gbps domain) and functional flexibility and extensibility. The project most closely related with active networking was developed in collaboration with the Eidgenössische Technische Hochschule Zürich, which provided the input for the software architecture of the Distributed Code Caching for Active Networks (DAN) [DDPP98], [DPP+99]. The latest stage of this ambitious work is the Dynamically Extensible Router (DER) [KDK+02], which in a way reunites all previous (and sometimes parallel) efforts.

The DER consists of four main modules (s. Fig. A.3): A Control Processor, an ATM Switch Core interconnecting a number of Port Processors and their respective Line Cards. The Control Processor runs the management and routing software and it will be explained shortly. The Switch Fabric is in reality a multi-Gbps ATM switch entirely designed and developed at Washington University, known as WUGS [CFFT97]. However, in the DER it is used simply as a high performance backplane to interconnect the Port Processors, which realize the data plane functionality of the node. The Line Cards provide conversion and encoding for the target physical layer.

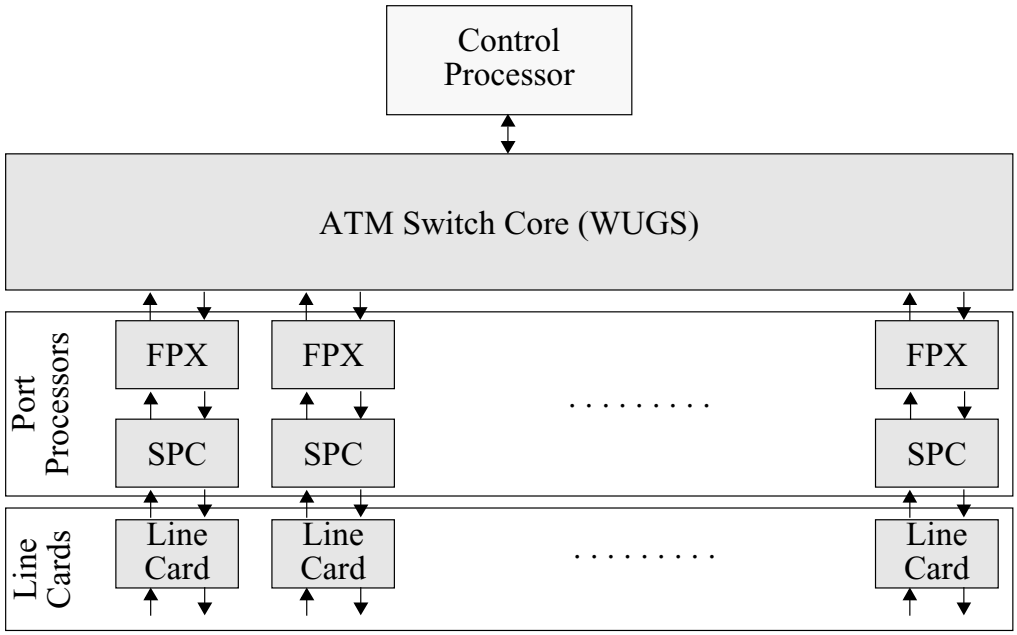


Figure A.3: Dynamically Extensible Router.

The Port Processors contain a Field Programmable Port Extender (FPX) board [LNTT01], a Smart Port Card (SPC) board [KDK+02], or both (s. Fig. A.4). The FPX presents a platform for the support of data plane functionality realized in programmable hardware (so called "Dynamic Hardware Plugins (DHP)" [TTL01]), while the SPC includes an embedded processor for software-based applications ("Software Plugins"), also on the data plane. In spite of this duality, the authors envision, that most high-level packet processing will be realized in the SPC. The FPX will take charge of low-level, processing-intensive tasks like packet classification, pattern matching, route lookup and the like.

The SPC basically consists of an embedded Intel Pentium processor, some DRAM, an FPGA that provides south bridge functionality and an ATM host-network interface. The SPC runs a version of the NetBSD OS modified for active networking.

The FPX is composed of two FPGAs, five memory banks and two network interfaces. Functionally, it is divided in two main subsystems: The Network Interface Device (NID) and the Reprogrammable Application Device (RAD), s. Fig. A.4. The RAD contains the synthesized VHDL code modules, that implement the desired functionality, like encryption or route lookup. The current FPX version supports two RAD-modules in the same FPGA. Each module has dedicated interfaces to two memory banks for its own use. The NID controls how packets are routed to and from the RAD modules. It also provides mechanisms to dynamically load hardware modules over the network. Partial reconfiguration is possible, thus enabling the update of one RAD module without interrupting the

packet processing on the other. The NID, then, provides the basic infrastructure for the introduction and management of hardware modules. The RAD presents a set of standardized interfaces to the application programmer, equivalent to a set of APIs for software applications. This facilitates code development.

The introduction of new applications into the router is managed by the DAN architecture. Basically, it follows a Programmable Network approach, in which plugins (software and hardware) are stored in a hierarchy of code repositories, analogous to the DNS. The Control Processor, besides running the usual routing and management protocols (OSPF, SNMP, etc), is also in charge of plugin management. Capsules are used for management purposes, but in order to improve performance, they do not carry code themselves. Capsules carry an indication (a "pointer") of the plugin to which they should be directed. Several such pointers will be carried, if a chain of plugins should process the packet. Normal packets do not carry such indications: The relevant plugin for their processing is detected through packet classification at the router interface. Should a plugin be needed, which is not currently installed in the router, it will be downloaded on-demand from a repository. Obviously, plugins can also be downloaded by an administrator off-line. To protect the integrity and validity of the code, repositories have to go through an authentication and authorization process, and plugins are digitally signed. Nevertheless, and again in order to improve performance, the DER project renounces to the idea of virtual machines for their inherent overhead. Plugins are written in a common high-level language (like C), compiled at the repository and downloaded as an executable by the node. There, it runs as native code, just as any other application. The DAN architecture foresaw a limited resource management entity for software applications. Basically, that entity would monitor the CPU cycle and memory consumption by the plugins. Although referred only as a monitor, it is easy to see, that that entity could be expanded to actually perform some form of resource control.

The performance of the DER components is, without any doubt, the best in the whole Active Networking community. Their node design is, at the same time, very flexible. However, a number of concerns related to the security and openness assets arise. They can be summarized by saying, that they do not perform any kind of runtime security checks. In the first place, the fact of bypassing the use of virtual machines (or equivalent constructs) for software plugins severely impacts the safety of their design. Even if the plugins are signed, by neither performing checks on their correctness, nor any kind of resource management, code bugs or malicious programmers can easily compromise the node integrity. This feature alone would make their design unsuitable for real-life deployment. The same applies for their hardware plugins, which do not undergo any kind of safety, security or resource control. Obviously, no isolation among plugins (software or hardware) exists. In the second place, their architecture, although foreseeing dynamic functionality extension, is ill-suited for true sharing. The lack of runtime checks is complemented by a centralized management architecture (the

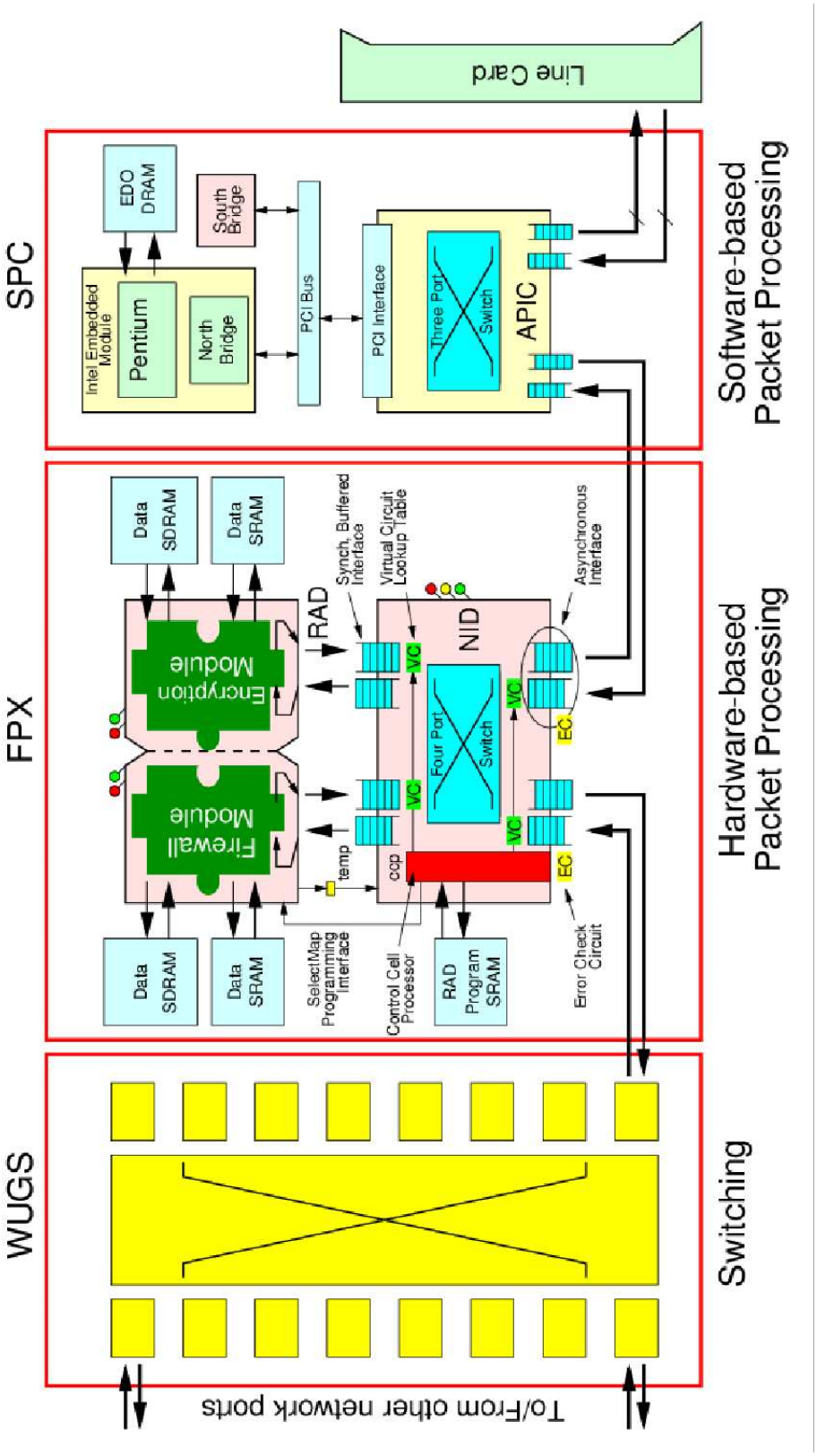


Figure A.4: Architecture of a Port Processor.

Control Processor), which controls the introduction and removal of all plugins. No provision for service operator-specific access to the plugins is foreseen.

The DER is then truly high-performance and flexible, albeit neither secure nor open. Nevertheless, in a closed environment, like present networks, it provides a network operator with an incremental step to increase the flexibility of their nodes, as long as they keep strict controls on their software and protect them from external access.

A.3 University of Pennsylvania: Switchware

The origins of the Active Networking activities at the University of Pennsylvania can be traced back to the Protocol Boosters project [FMS+98]. Its main goal was to develop the possibility of dynamically installing protocol extensions in network nodes without having to replace the existing software. In a way, the idea was similar to linking LINUX modules to the kernel on-demand. From that first "active" initiative, two main strands emerged: On the one side, a platform for Booster support in hardware was developed, and on the other, a secure Active Network architecture was designed.

The SwitchWare project [AAH+98] developed an architecture, which supported active capsules and active extensions (i.e., bigger applications stored in repositories, also known as "switchlets") simultaneously. The main emphasis of SwitchWare, though, was on developing a secure infrastructure for Active Networking [AMK+01]: Code and node integrity, authentication and authorization were its main concerns. It should be pointed out, however, that they were not concerned with safety: The goal was to devise an infrastructure, that would retrieve unaltered the right code from the right place and allow it to do only authorized operations. If, on the other hand, the code itself performed semantically illegal operations, or was otherwise buggy or malicious, its consequences remained obscure. Equally, resource management beyond traditional OS mechanisms was not considered [AAK+00].

The approach was to try to achieve security without penalizing performance. To that end, the use of virtual machines was avoided. Additionally, heavyweight security checks were introduced at the boot or compile time, leaving only more lightweight operations for the runtime environment. In order to achieve this goal, a new programming language was developed (Programming Language for Active Networks, PLAN [AAKS98a]) and a second one was adapted (Caml [AAKS98b]). PLAN was a very simple, very restrictive language, to be used in active capsules. Its functionality was minimal, so as to be safe. As way of example, PLAN does not allow to change state information in a node, it can only operate on the content of the capsule itself. Obviously, this approach was unusable for bigger, more complex applications. To that purpose Caml was used. To ensure a secure environment in which to run Switchlets, the Secure Active Network Environment (SANE) architecture was developed (s. Fig. A.5).

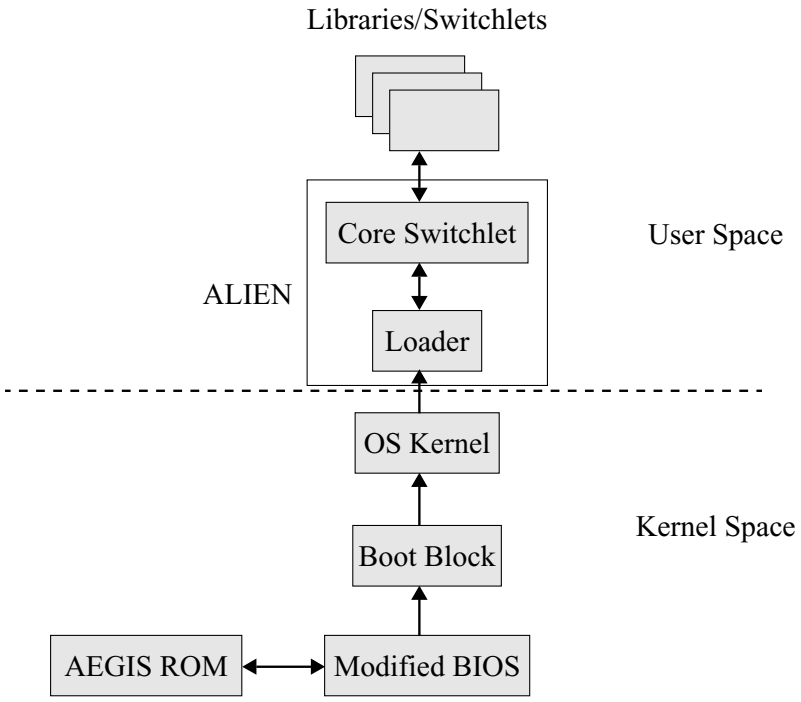


Figure A.5: The SANE Layering Structure.

At the basis of the SANE architecture [AAKS98b] lies a secure bootstrapping procedure, called AEGIS. The main idea is to check the integrity of every piece of code involved in the bootstrapping process prior to allowing it to take control of the node. With the help of a modified BIOS and a minimum set of axiomatically safe code, every layer is checked with the help of digital signatures, until the node is up and running.

On top of the OS, two further elements, jointly called ALIEN, manage the Switchlets. The Caml loader is responsible to interpret Caml bytecode and serves as an interface to the OS (LINUX). Additionally, it dynamically loads switchlets on-demand. The Core Switchlet represents the API for user-programmed Switchlets. It also implements access control by means of module thinning (see below).

The choice of Caml as a programming language derived from its security characteristics: It supports strong typing, module thinning, garbage collection and dynamic module loading. Strong typing prevents uncontrolled casts, that might provoke errors at runtime. Module thinning adapts the interface seen by the applications depending on their access rights. In this way, the language itself allows to restrict the functionality seen by an application. Garbage collection prevents the deallocation of memory, its subsequent reallocation to another application and the peeking into it by a re-claiming previous module, which would equate with "spying". Furthermore, pointing to deallocated memory can provoke sys-

tem crashes. Garbage collection prevents such effects. Dynamic module loading was a pre-requisite of Programmable Networks. The SwitchWare approach, then, relied heavily in a programming language choice, that limited the expressiveness of programs to a secure subset.

As mentioned before, this impressive security construct nevertheless neglected an evaluation of Switchlet safety. Alternatively, it could have introduced strong resource management mechanisms and virtual machines to prevent service interference and the endangering of node stability. Under these circumstances, buggy code could still provoke system misbehavior.

A second weakness lies in its performance limitations. Although providing good performance for basic applications in a software environment, SwitchWare is not apt for deployment even in modern MANs. To solve this problem, the use of the Protocol Booster hardware platform in the context of SwitchWare was proposed. The Programmable Protocol Processing Pipeline (P4, s. Fig. A.6 and [Had99], [HS97]) was basically a pipeline, in which every element was an FPGA. The idea was to distribute processing-intensive Boosters (like a FEC module) among a set of hardware processing elements in a pipelined fashion. A Switching Array served to manage the pipeline ordering, thus also allowing the dynamic introduction and removal of functionality from individual FPGAs. An external host controlled the P4 configuration, triggered the update of FPGA code and implemented a signaling protocol for the coordination of Boosters in a network-wide scenario. The P4 was designed for ATM networks running at OC-3 speeds (155 Mbps).

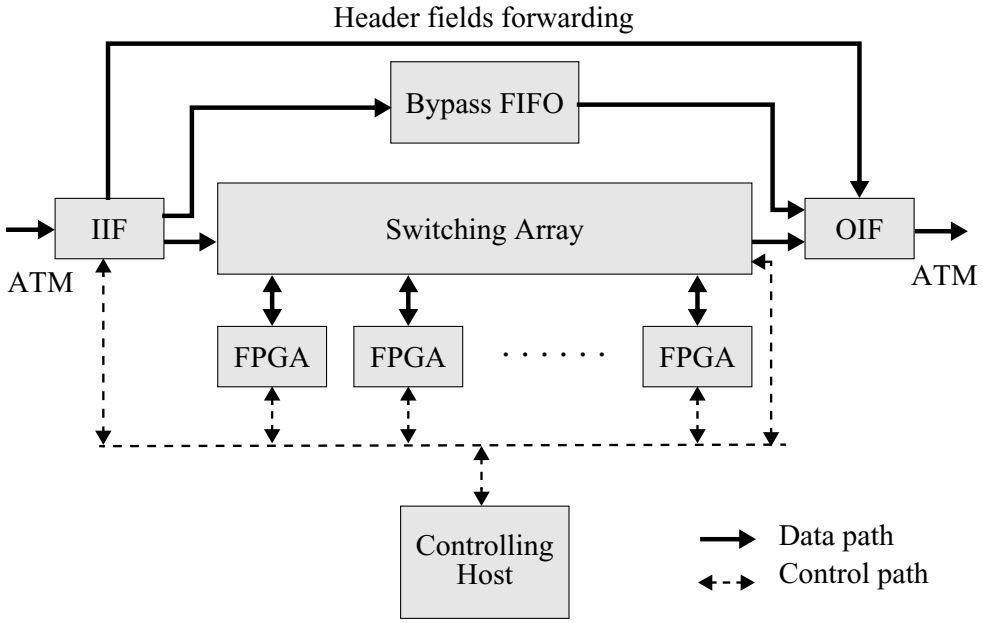


Figure A.6: Programmable Protocol Processing Pipeline.

This design represents a first try at developing a hardware platform for Active Networking that could overcome its performance limitations. Under that perspective, and as proof-of-concept, it was a success. Nevertheless, a number of limitations should be mentioned: Due probably to technological limitations, the platform does not foresee the access to external resources by the FPGAs (e.g. DRAM banks), thus limiting their processing possibilities. The platform was designed as a pipeline, and although it could also support several services in parallel, adequate management mechanisms to prevent module interference are missing. Furthermore, the platform does not perform any kind of resource management or safety checks either, which makes sense, assuming that it is controlled by a single, trusted party. In an Active Networking environment, though, such controls are indispensable. In due fairness, however, the dangers of hardware attacks were analyzed and discussed in [Had99]. As already mentioned, the technological limitations of FPGA technology in 1997 did not allow for a more ambitious performance goal than those 155 Mbps, which nowadays seem insufficient.

Summarizing, SwitchWare presents the most ambitious try at a secure Active Networking architecture to date. Together with the P4, it presents a promising mixture of secure software and high-performance hardware (at that time), which could lead to feasible Programmable Networks. The restriction of programming languages might reduce the flexibility and usability of the proposal to some degree.

A.4 Carnegie Mellon University: Darwin

The Darwin project at Carnegie Mellon University [CCF+01], [GSTF00], [GS01] wanted to design all necessary elements for the establishment, management and support of overlay networks (s. Fig. A.7). The present vision of overlays brings to mind approaches, that are independent of the physical network, like Peer-to-Peer applications. The goal of Darwin was different: The overlays running on top of a certain infrastructure had to be able to access, configure and share the physical network's resources. Every service provider would have control on the allocation of its share of network resources, in isolation from all others. The resources provided by the network, as stated in section 3.1, were abstracted to be bandwidth, memory and processing capacity in the network nodes.

In order to accomplish its goal, Darwin envisions four different elements in their architecture:

- A high-level resource broker called **Xena**.
- Runtime resource management instances, called **Delegates**.
- A hierarchical scheduling algorithm **(H-FSC)**.
- A management protocol called **Beagle**.

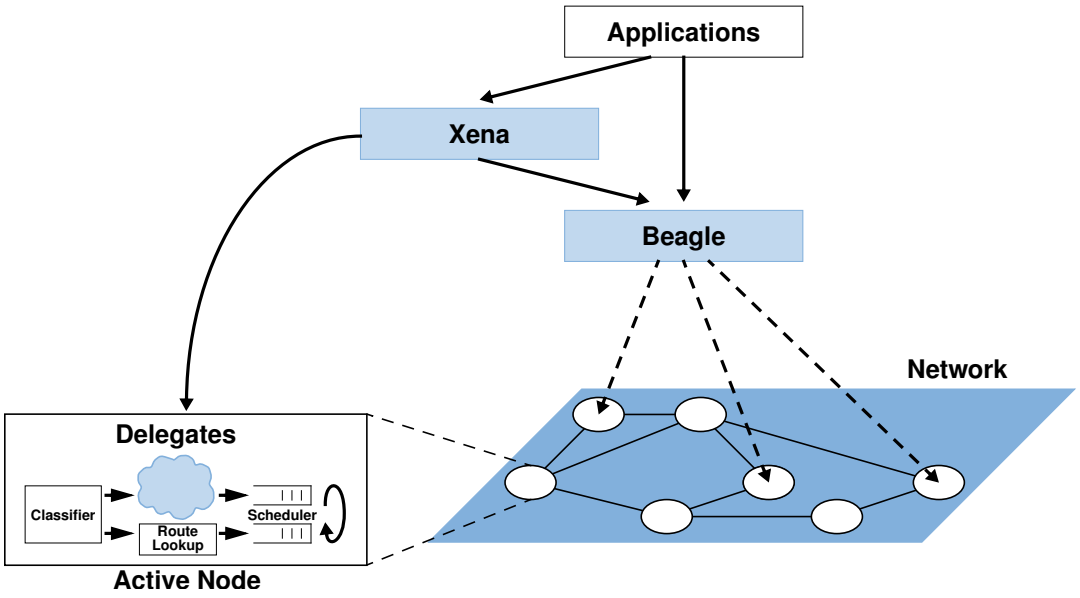


Figure A.7: The Darwin Network Model.

Xena's mission is to perform admission control on new overlay establishment requests, map the description of resources needed by the overlay to network equivalents, keep track of available network resources, optimize resource usage at the network level (for that purpose, semantic-keeping transformations of overlay descriptions can be performed by Xena) and allocate those resources in the network nodes. It is realized in an independent node and can also serve as proxy for such applications, that are not able to communicate their resource requirements themselves.

Delegates are instantiated by Xena in the corresponding nodes upon creation of a new overlay (a "Virtual Mesh" in Darwin terminology). Two kinds of delegates exist: Control plane delegates and data plane delegates (s. Fig. A.8). Together, they are roughly equivalent to the Execution Environment and the Active Applications described in section 3.1. The control plane delegate is characterized by its resource requirements, its runtime environment and the flows and state, which it is allowed to manipulate. It basically receives the instructions relative to resource allocation sent by Xena and accordingly configures the node in which it runs. This configuration implies reserving certain resources and also implementing a security policy. To that end, the delegates use Access Control Lists (ACL), which summarize the rights of programs associated to this delegate over node resources and packets. Additionally, delegates run on a Java virtual machine, which presents a safety architecture of its own. The data plane delegate (the Active Application) represents the functionality to be performed on the traffic: Compression, encryption, etc. The control plane delegate adds filters to the packet classifier to direct the relevant packets to the data plane delegate.

It can also alter the routing table and related information to set tunnels, choose alternate routes, etc. for its managed traffic.

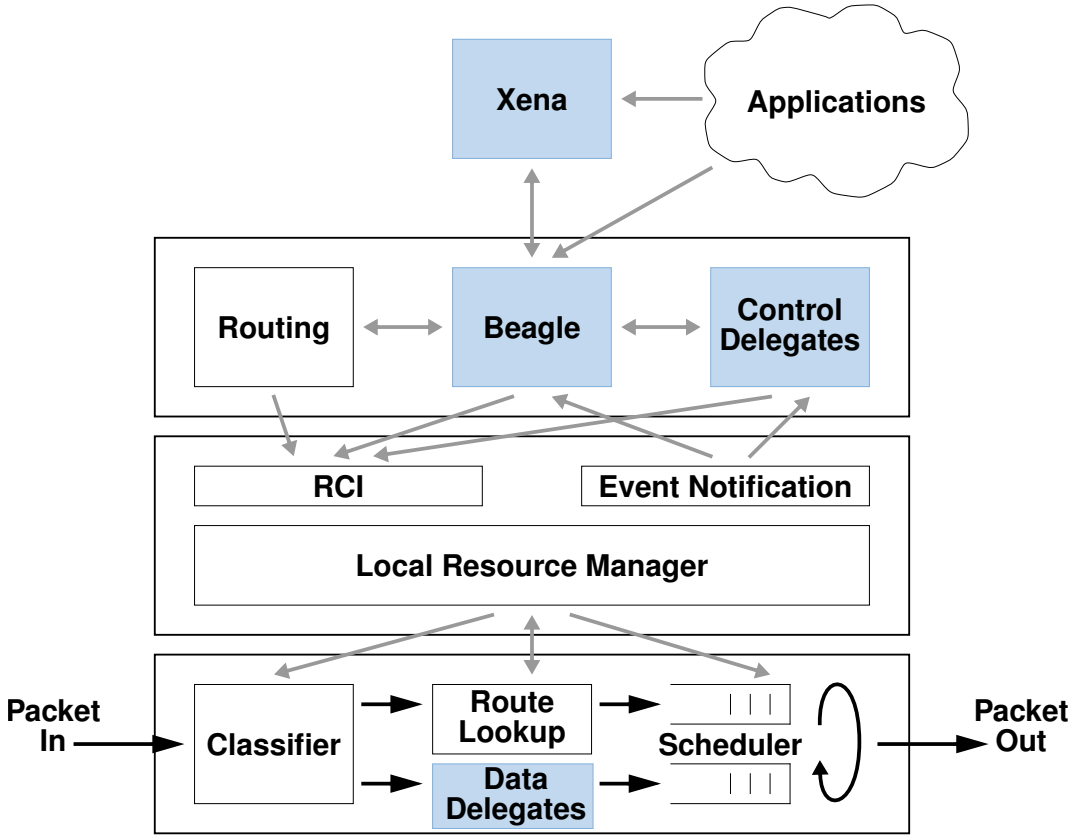


Figure A.8: The Darwin Network Node Architecture.

To prevent interference among delegates, besides the ACLs, a hierarchical scheduling algorithm is used. As other such algorithms, it has the property of dividing a resource among entities in a tree fashion (s. Fig. A.9). Every branch falls under the control of a certain entity (in this case, a control plane delegate), which can further divide it as it pleases. In this way, a control plane delegate can share a number of resources among several data plane delegates under its surveillance. Since delegates can only operate on their branch of the resource tree, interference among them is prevented.

The Beagle signaling protocol transports the messages between Xena and the delegates. At delegate establishment, it first transports the resource limitations for the new service to the NodeOS, which will enforce them by using its own set of ACLs. Afterwards, the code instantiating the delegates (control and data plane) is transported to the node and installed. This includes authentication and authorization between Xena and the node. From that moment on, Beagle serves as a signaling protocol for any changes, that might occur on the network state. In a way, Beagle realizes a similar function to RSVP, but for whole overlays instead

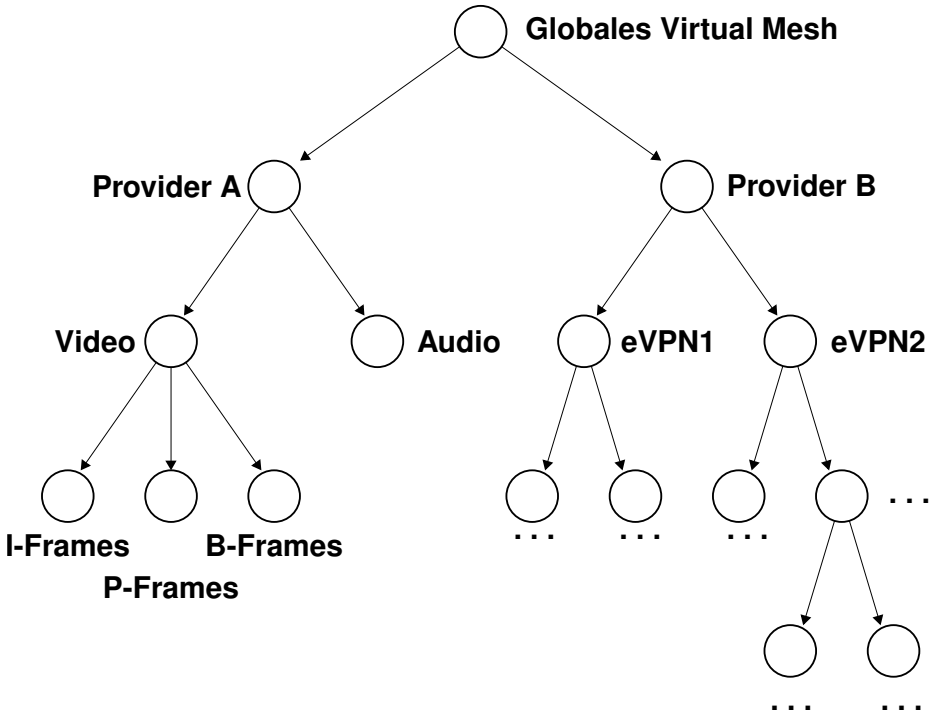


Figure A.9: The Grouping Tree Concept for Hierarchical Scheduling.

of individual flows. It also presents an idealized view of the node resources to Xena, so that optimization procedures will be performed there.

As can be seen, every one of these elements has a different scope and represents a different resource aggregation level as well as a different time scale: Xena performs the optimization and allocation of resources at the network level. The time-frame for such activities would be in the order of minutes to hours. The delegates transform these instructions into a node configuration, which would take place in the sub-second domain. Once configured, the scheduling algorithm ensures QoS enforcement at the packet level (microseconds). Beagle is the protocol in charge of transporting the instructions from Xena to the Delegates and *vice versa*.

Darwin presents a comprehensive architecture to introduce and manage overlay networks integrated with the underlying infrastructure. Nevertheless, a number of issues remain unsolved. First, the architecture stops right underneath the "service layer": The mechanisms described refer only to raw resources. A service overlay would need similar solutions to find, index, reserve and organize *service instances or components* to form a coherent, usable whole. This is an area, which has been addressed elsewhere [BS99].

Secondly, the introduction of resource brokers presents a number of well-known, unsolved problems: Scalability for big networks is only one of them. Additionally, the interaction between brokers pertaining to (administratively)

different networks is a critical point for end-to-end coherence. Furthermore, the optimization problem to be solved is NP-hard, which forces the use of heuristics. The quality of such heuristics, as well as how easy it might be to find them, is an open point.

Thirdly, Darwin allows delegates to directly configure the basic features of a network node, like the routing tables or the classification rules. This is arguably a dangerous move, for those features affect the whole of the traffic, not only packets belonging to a certain overlay. As was explained in chapter 4, strict separation of fundamental router tasks from value-added services (like overlays) seems to be a more secure and equally flexible solution.

Lastly, the introduction of programmability at the hardware level is not considered, neither the risks associated with hardware attacks to the system. This drawback severely penalizes the performance achievable with Darwin.

As a conclusion, Darwin is a thorough, comprehensive architecture, which has addressed two aspects deemed critical for an Active Network proposal: Usability and security, albeit with some weaknesses. Performance, on the other hand, was sacrificed from the beginning.

In this last section, the A&PN projects closest to this Thesis' own work have been reviewed and their strengths and weaknesses highlighted. Their choices in the performance vs security vs openness trade-off have been exposed. They serve as a reference to compare the choices made in the Octopus Network Model to overcome that trade-off.

Bibliography

[AAH+98] D. S. Alexander, W.A. Arbaugh, M.W. Hicks, P. Kakkar, A.D. Keromytis, J.T. Moore, C.A. Gunter, S.M. Nettles, and J.M. Smith. The SwitchWare Active Network Architecture. *IEEE Network Magazine*, May/June 1998.

[AAK+00] D. S. Alexander, W.A. Arbaugh, A.D. Keromytis, S. Muir, and J.M. Smith. Secure Quality of Service Handling: SQoSH. *IEEE Communications Magazine*, April 2000.

[AAKS98a] D. S. Alexander, W. A. Arbaugh, A. D. Keromytis, and J. M. Smith. PLAN: A Packet Language for Active Networks. *IEEE Network Special Issue on Active and Controllable Networks*, 12(3):37 – 45, 1998.

[AAKS98b] D. S. Alexander, W.A. Arbaugh, A.D. Keromytis, and J.M. Smith. A Secure Active Network Environment Architecture: Realization in SwitchWare. *IEEE Network Magazine*, May/June 1998.

[ABC+03] F. Arts, P. Barri, I. Clemminck, A. Niemegeers, B. Pauwels, G. Taildeman, and M. Vrana. Network processor requirements and benchmarking. *Computer Networks*, 41, 2003.

[AFB98] W. Almesberger, T. Ferrari, and J.Y. Le Boudec. SRP: A scalable resource reservation protocol for the Internet. *Computer Communications*, 21, 1998.

[AJ02] D.O. Awduche and B. Jabbari. Internet Traffic Engineering Using Multi-Protocol Label Switching (MPLS). *Computer Networks*, 40(1):111–129, 2002.

[Alta] Altera Corp. *Excalibur Device Overview Data Sheet*. Available at www.altera.com.

[Altb] Altera Corp. *NIOS 3.0 CPU Datasheet*. Available at www.altera.com.

[Altc] Altera Corp. *Stratix II Device Handbook*. Available at www.altera.com.

[AMA+99] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus. *RFC2702: Requirements for traffic engineering over MPLS*. Internet Engineering Task Force (IETF), September 1999.

[AMK+01] D.S. Alexander, P.B. Menage, A.D. Keromytis, W.A. Arbaugh, K.G. Anagnostakis, and J.M. Smith. The Price of Safety in an Active Network. *Journal of Communications and Networks*, 2001.

[ANS] ANSI. *Available at www.atis.org/tg2k/.*

[ARM99] ARM. AMBA Specification, Rev 2.0, 1999.

[Awe01] J. Aweya. IP Router Architectures: An Overview. *International Journal of Communication Systems*, 14:447–475, 2001.

[BBC+98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. *RFC2475: An Architecture for Differentiated Service* . Internet Engineering Task Force (IETF), December 1998.

[BCS94] R. Braden, D. Clark, and S. Shenker. *RFC1633: Integrated Services in the Internet Architecture: an Overview* . Internet Engineering Task Force (IETF), June 1994.

[BDH+03] L. Burgstahler, K. Dolzer, C. Hauser, J. Jähnert, S. Junghans, C. Macián, and W. Payer. Beyond Technology: The Missing Pieces for QoS Success. In *Proceedings of the ACM SIGCOMM Conference*, Karlsruhe, Germany, August 2003.

[BS99] M. Brunner and R. Stadler. The Impact of Active Networking Technology on Service Management in a Telecom Environment. In *Proceedings of the 6th International Symposium on Integrated Network Management*, 1999.

[BZB+97] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. *RFC2205: Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification* . Internet Engineering Task Force (IETF), September 1997.

[CAB99] J.A. Copeland, R. Abler, and K.L. Bernhardt. IP flow identification for IP traffic carried over switched networks. *Computer Networks*, 31, 1999.

[Cal99] K. Calvert. *Architectural Framework for Active Networks Version 1.0.* Active Network Working Group, July 1999.

[CCF+01] P. Chandra, Y. Chu, A. Fisher, J. Gao, C. Kosak, T.S.E. Ng, P. Steenkiste, E. Takahashi, and H. Zhang. Darwin: Customizable Resource Management for Value-Added Network Services. *IEEE Network*, 2001.

[CF98] D.D. Clark and W. Fang. Explicit Allocation of Best-Effort Packet Delivery Service. *IEEE/ACM Transactions on Networking*, 6(4), August 1998.

[CFFT97] T. Chaney, J.A. Fingerhut, M. Flucke, and J.S. Turner. Design of a Gigabit ATM Switch. In *Proceedings of Infocom*, 1997.

[CGMP99] S. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching Output Queuing with a Combined Input/Output-Queued Switch. *IEEE Journal on Selected Areas on Communications*, 17(6), June 1999.

[Cisa] Cisco Systems Inc. *Available at business.cisco.com/glossary/.*

[Cisb] Cisco Systems Inc. Cisco 12816 Router Data Sheet. Available at www.cisco.com.

[Cisc] Cisco Systems Inc. Designing Service Provider Core Networks to Deliver Real-Time Services. Available at www.cisco.com.

[Cisd] Cisco Systems Inc. The Evolution of High-End Router Architectures. Available at www.cisco.com.

[CLO01] H.J. Chao, C.H. Lam, and E. Oki. *Broadband Packet Switching Technologies*. John Wiley & Sons, Inc., 2001.

[CLS+04] C. Clark, W. Lee, D. Schimmel, D. Contis, M. Koné, and A. Thomas. A Hardware Platform for Network Intrusion Detection and Prevention. In *Proceedings of the Third Workshop on Network Processors and Applications (NP-3)*, Madrid, Spain, February 2004.

[CMK+99] A.T. Campbell, H.G. De Meer, M.E. Kounavis, K. Miki, J.B. Vicente, and D. Villela. A Survey of Programmable Networks. *ACM Sigcomm Computer Communication Review*, 29(2):7–24, April 1999.

[CNRS98] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. *RFC 2386: A framework for QoS-based routing in the Internet*. Internet Engineering Task Force (IETF), August 1998.

[Com00] D.E. Comer. *Internetworking with TCP/IP*, volume 1: Principles, Protocols, and Architectures. Prentice-Hall, 4th edition, 2000.

[Cor] Altera Corp. Homepage. www.altera.com.

[CSZL01] G. Carle, H. Sanneck, S. Zander, and L. Le. Deploying an Active Voice Application on a Three-Level Active Network Node Architecture. In *Proceedings of the 3rd International Working Conference on Active Networks*. IFIP-TC6, Springer Verlag, September/October 2001.

[DBCP97] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink. Small Forwarding Tables for Fast Routing Lookups. In *Proceedings of the ACM SIGCOMM Conference*, 1997.

[DDPP98] D. Decasper, Z. Dittia, G. Parulkar, and B. Plattner. Router Plugins: A Software Architecture for Next Generation Routers. In *Proceedings of the ACM SIGCOMM Conference*, 1998.

[Dev] C. Devine. AES Encryption Benchmark. Available at www.cr0.net:8040/code/crypto/aesbench/.

[DH95] S. Deering and R. Hinden. *RFC1883: Internet Protocol, Version 6 (IPv6) Specification* . Internet Engineering Task Force (IETF), December 1995.

[DH98] S. Deering and R. Hinden. *RFC2460: Internet Protocol, Version 6 (IPv6) Specification* . Internet Engineering Task Force (IETF), December 1998.

[DL01] S. Dharmapurikar and J. Lockwood. Synthesizable Design of a Multi-Module Memory Controller. Technical Report WUCS-01-26, Applied Research Lab, Washington University in St. Louis, September 2001.

[DP00] K. Dolzer and W. Payer. On Aggregation Strategies for Multimedia Traffic. In *Proceedings of the 1st Polish-German Teletraffic Symposium (PGTS2000)*, Dresden, Germany, September 2000.

[DPP+99] D.S. Decasper, B. Plattner, G.M. Parulkar, S. Choi, J.D. DeHart, and T. Wolf. A Scalable High-Performance Active Network Node. *IEEE Network Magazine*, January/February 1999.

[DR98] J. Daemen and V. Rijmen. AES Proposal: Rijndael. *NIST AES Proposal*, 1998.

[Eck01] C. Eckert. *IT-Sicherheit*. Oldenbourg Verlag, 1st edition, 2001.

[Eng03] T. Engbersen. Guest editorial: Network processors. *Computer Networks*, 41, 2003.

[FHK+03] T. Fuhrmann, T. Harbaum, P. Kassianidis, M. Schöller, and M. Zitterbart. Results on the Practical Feasibility of Programmable Network Services. In *Proceedings of the 2nd International Workshop on Active Network Technologies and Applications (ANTA 2003)*, Osaka, Japan, May 2003.

[FHSZ02] T. Fuhrmann, T. Harbaum, M. Schöller, and M. Zitterbart. AMnet 2.0: An Improved Architecture for Programmable Networks. In *IFIP-TC6 4th International Working Conference on Active Networks (IWAN)*, Zurich, Switzerland, December 2002.

[FJ93] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, August 1993.

[FJ95] S. Floyd and V. Jacobson. Link-sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking*, 3(4), August 1995.

[FKLS98] K.W. Fendick, V.P. Kumar, T.V. Lakshman, and D. Stiliadis. The PacketStar 6400 IP Switch - An IP Switch for the Converged Network. *Bell Labs Technical Journal*, October-December 1998.

[FLYV93] V. Fuller, T. Li, J. Yu, and K. Varadhan. *RFC1519: Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy.* Internet Engineering Task Force (IETF), September 1993.

[FMS+98] D.C. Feldmeier, A.J. McAuley, J.M. Smith, D.S. Bakin, W.S. Marcus, and T.M. Raleigh. Protocol Boosters. *IEEE Journal on Selected Areas on Communications*, 16(3), April 1998.

[For94] W. Ford. *Computer Communications Security.* Prentice Hall, 1994.

[GEH+98] C. Gbaguidi, H. Einsiedler, P. Hurley, W. Almesberger, and J. P. Hubaux. A Survey of Differentiated Services Architectures for the Internet. Technical report, EPFL, May 1998.

[GIGK95] L. Georgiadis, I.Cidon, R. Guérin, and A. Khamisy. Optimal Buffer Sharing. *IEEE Journal on Selected Areas on Communications*, September 1995.

[GPS+00] Alex Galis, Bernhard Plattner, Jonathan M. Smith, Spyros Denazis, Eckhard Moeller, Hui Guo, Cornel Klein, Joan Serrat, Jan Laarhuis, George T. Karetsos, and Chris Todd. A Flexible IP Active Networks Architecture. In *Proceedings of the 2nd International Working Conference on Active Networks*, 2000.

[GS01] J. Gao and P. Steenkiste. An Access Control Architecture for Programmable Routers. In *Proceedings of the Openarch*, 2001.

[GSTF00] J. Gao, P. Steenkiste, E. Takahashi, and A. Fisher. A Programmable Router Architecture Supporting Control Plane Extensibility. *IEEE Communications Magazine*, March 2000.

[Had99] I. Hadžić. *Applying Reconfigurable Computing to Reconfigurable Networks*. Ph. D. Thesis, University of Pennsylvania, 1999.

[Har02] T. Harbaum. *Rekonfigurierbare Routerhardware für adaptive Dienstplattformen*. Ph.D. Thesis, Universität Karlsruhe, 2002.

[Her00] S. Herzog. *RFC2750: RSVP Extensions for Policy Control* . Internet Engineering Task Force (IETF), January 2000.

[Her01] D. Herity. Network Processor Programming. *Embedded.com*, 2001.

[HMP00] T. Harbaum, D. Meier, and M. Prinke. Hardware support for RSVP capable routing. In *Proceedings of the 3rd ATM Workshop*, June 2000.

[HMS03] O. Horvath, C. Macián, and S. Stanchina. A Simple Autonomous Reconfigurable Cryptographic Node. In *Proceedings of the 9th Open European EUNICE Summer School and IFIP Workshop on Next Generation Networks*, Balatonfüred, Hungary, September 2003.

[HS97] I. Hadžić and J. M. Smith. P4: A platform for FPGA implementation of protocol boosters. In *Proceedings of the 7th International Workshop on Field Programmable Logic and Applications (FPL97)*, September 1997.

[HSS+02] A. Hess, M. Schöller, G. Schäfer, A. Wolisz, and M. Zitterbart. A dynamic and flexible access control and resource monitoring mechanism for active nodes. In *Proceedings of the 5th International Conference on Open Architectures and Network Programming (OPENARCH) (Short Paper Session)*, 2002.

[Int95] International Standardization Organization. *ISO/IEC 8652: Ada Reference Manual*, 1995.

[ITU93] ITU-T. *Vocabulary of Terms for ISDN*, March 1993.

[ITU94a] ITU-T. *Recommendation E-800: Terms and definitions related to quality of service and network performance including dependability*, August 1994.

[ITU94b] ITU-T. *Recommendation X.200: Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*, June 1994.

[ITU02] ITU-T. *Traffic control and congestion control in IP based networks*, March 2002.

[Juna] Juniper Networks. T-Series Routing Platforms Datasheet. Available at www.junipernetworks.com.

[Junb] Juniper Networks. The Essential Core: Juniper Networks T640 Internet Routing Node with Matrix Technology. Available at www.junipernetworks.com.

[Jun02] S. Junghans. A Universal Hardware Platform (UHP) for rapid prototyping of network nodes. Internal Presentation, Institute of Communication Networks and Computer Engineering, University of Stuttgart, September 2002.

[Kar01] S. Karnouskos. Security implications of implementing active network infrastructures using agent technology. *Computer Networks*, 36:87–100, 2001.

[KCY+03] I. Keslassy, S.T. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard, and N. McKeown. Scaling Internet Routers Using Optics. In *Proceedings of the ACM SIGCOMM Conference*, Karlsruhe, Germany, August 2003.

[KDK+02] F. Kuhns, J. DeHart, A. Kantawala, R. Keller, J. Lockwood, P. Pappu, D. Richard, D. Taylor, J. Parwatikar, E. Spitznagel, J. Turner, and K. Wong. Design and Evaluation of a High-Performance Dynamically Extensible Router. In *Proceedings of the DARPA Active Networks Conference and Exposition*, May 2002.

[Kec02] D.O. Keck. *Erkennung von Wechselwirkungen zwischen Mehrwertdiensten durch Analyse ihrer Konfigurationen und Protokollabläufe*. Ph.D. Thesis, Universität Stuttgart, 2002.

[Kel00] F. P. Kelly. *Stochastic Networks: Theory and Applications*, chapter Notes on effective bandwidths, pages 141–168. Oxford University Press, September 2000.

[KKR97] H. Kröner, P.J. Kühn, and T. Renger. Management von ATM-Netzen. *Informationstechnik und Technische Informatik*, 39(1), 1997.

[KLS98] V.P. Kumar, T.V. Lakshman, and D. Stiliadis. Beyond Best Effort: Router Architectures for the Differentiated Services of Tomorrow's Internet. *IEEE Communications Magazine*, May 1998.

[KP01] S. Karlin and L. Peterson. VERA: An Extensible Router Architecture. In *Proceedings of the Openarch*, 2001.

[KS98] S. Keshav and R. Sharma. Issues and Trends in Router Design. *IEEE Communications Magazine*, May 1998.

[LM97] S. Lin and N. McKeown. A simulation study of IP switching. In *Proceedings of the ACM SIGCOMM Conference*, 1997.

[LMK+03] J.W. Lockwood, J. Moscola, M. Kulig, D. Reddick, and T. Brooks. Internet Worm and Virus Protection in Dynamically Reconfigurable Hardware. In *Proceedings of the Military and Aerospace Programmable Logic Device (MAPLD)*, September 2003.

[LNTT01] J.W. Lockwood, N. Naufel, J.S. Turner, and D.E. Taylor. Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX). In *Proceedings of the FPGA*, February 2001.

[LSV99] B. Lampson, V. Srinivasan, and G. Varghese. IP Lookups Using Multiway and Multicolumn Search. *IEEE/ACM Transactions on Networking*, 7(3), June 1999.

[LYR02] J. Li, M. Yarvis, and P. Reiher. Securing distributed adaptation. *Computer Networks*, 38:347–371, 2002.

[MA98] N. McKeown and T.E. Anderson. A quantitative comparison of iterative scheduling algorithms for input-queued switches. *Computer Networks and ISDN Systems*, 30, 1998.

[Man03] E. Mannie. *Generalized Multi-Protocol Label Switching Architecture, Internet Draft*. IETF, May 2003. Work in Progress.

[McK95] N. McKeown. *Scheduling Algorithms for Input-Queued Cell Switches*. PhD thesis, University of California at Berkeley, May 1995.

[McK04] N. McKeown. Network Processors and their Memory. Keynote Address at the Third Workshop on Network Processors and Applications (NP-3), February 2004.

[MF01] C. Macián and R. Finthammer. An Evaluation of the Key Design Criteria to Achieve High Update Rates in Packet Classifiers. *IEEE Network*, 15(6), November/December 2001.

[Mic] Micron Inc. Available at www.micron.com.

[Mic99] Micron Inc. Small-outline SDRAM module MT4LSDT464(L)H, MT8LSDT864(L)H data sheet, 1999.

[MLP+01] S. Murphy, E. Lewis, R. Puga, R. Watson, and R. Yee. Strong Security for Active Networks. In *Proceedings of the IEEE Openarch Conference*, 2001.

[MRLC98] J.E. van der Merwe, S. Rooney, I. Leslie, and S. Crosby. The Tempest – A Practical Framework for Network Programmability. *IEEE Network Magazine*, May/June 1998.

[Nat01] National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES). *Federal Information Processing Standards Publication (FIPS PUB)*, 197, November 2001.

[NBBB98] K. Nichols, S. Blake, F. Baker, and D. Black. *RFC2474: Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers* . Internet Engineering Task Force (IETF), December 1998.

[NJZ99] K. Nichols, V. Jacobson, and L. Zhang. *RFC2638: A Two-bit Differentiated Services Architecture for the Internet*. Internet Engineering Task Force (IETF), 1999.

[NL98] George C. Necula and Peter Lee. Safe, Untrusted Agents Using Proof-Carrying Code. In Giovanni Vigna, editor, *Mobile Agent Security*, Lecture Notes in Computer Science No. 1419, pages 61–91. Springer-Verlag: Heidelberg, Germany, 1998.

[OCP01] OCP International Partnership. Open Core Protocol Specification, Release 1.0, 2001.

[Par98] C. Partridge. Designing and Building Gigabit and Terabit Internet Routers. Tutorial, ACM SIGCOMM Conference, Vancouver, Canada, September 1998.

[PC03] M. Peyravian and J. Calvignac. Fundamental architectural considerations for network processors. *Computer Networks*, 41, 2003.

[PCB+98] C. Partridge, P.P. Carvey, E. Burgess, I. Castineyra, T. Clarke, L. Graham, M. Hathaway, P. Herman, A. King, S. Kohalmi, T. Ma, J. Mcallen, T. Mendez, W.C. Milliken, R. Pettyjohn, J. Rokosz, J. Seeger, M. Sollins, S. Storch, B. Tober, G.D. Troxel, D. Waitzman, and S. Winterble. A 50 Gb/s IP Router. *IEEE/ACM Transactions on Networking*, 6(3), June 1998.

[PG93] Abhay K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, 1993.

[PG94] Abhay K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the multiple node case. *IEEE/ACM Transactions on Networking*, 2(2), April 1994.

[Pos81] J. Postel. *RFC791: Internet Protocol*. Internet Engineering Task Force (IETF), September 1981.

[PS99] Ping Pan and Henning Schulzrinne. Yessir: a simple reservation mechanism for the internet. *SIGCOMM Comput. Commun. Rev.*, 29(2):89–101, 1999.

[Pso99] K. Psounis. Active Networks: Applications, Security, Safety, and Architectures. *IEEE Communications Surveys*, First Quarter, 1999.

[RLG98] Q. Razouqi, T. Lee, and S. Ghosh. A guaranteed-no-cells-dropped buffer management scheme with selective blocking for cell-switching networks. *Computer Communications*, 21:930–946, 1998.

[RVC01] E. Rosen, A. Viswanathan, and R. Callon. *RFC3031: Multiprotocol Label Switching Architecture*. Internet Engineering Task Force (IETF), January 2001.

[SB95] S. Shenker and L. Breslau. Two Issues in Reservation Establishment. In *Proceedings of the ACM SIGCOMM Conference*, Cambridge, MA, August 1995.

[Sed97] R. Sedgewick. *Algorithms in C, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching*. Addison-Wesley, 3rd edition, 1997.

[Sik01] A. Sikora. Der PLD-Report. *Elektronik*, 5, 2001.

[Sil01] Silicore Corp. Specification for the WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores, Revision B.2, 2001.

[SLSC98] B. Suter, T. Lakshman, D. Stiliadis, and A. Choudhury. Efficient active queue management for internet routers. In Proceedings of Interop Engineers Conferece, Las Vegas, NV, May 1998.

[Sta98a] W. Stallings. SNMP and SNMPv2: The Infrastructure for Network Management. *IEEE Communications Magazine*, March 1998.

[Sta98b] W. Stallings. SNMPv3: A Security Enhancement for SNMP. *IEEE Communications Surveys*, 1(1), 1998.

[SV99] C. Shapiro and H. R. Varian. *Information Rules: A Strategic Guide to the Network Economy*. Harvard Business School Press, 1999.

[Tan00] A.S. Tanenbaum. *Computer Networks*. Prentice-Hall International, Inc., 3rd edition, 2000.

[Tel] Deutsche Telekom. http://www.telekom.de/t-pay.

[TMW97] K. Thompson, G.J. Miller, and R. Wilder. Wide-area traffic patterns and characteristics. *IEEE Network*, 11(6):10–23, 1997.

[TP99] H. H. Tzeng and T. Przygienda. On Fast Address-Lookup Algorithms. *IEEE Journal on Selected Areas on Communications*, 17(6), June 1999.

[TSS+97] D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall, and G.J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine*, January 1997.

[TTL01] D.E. Taylor, J.S. Turner, and J.W. Lockwood. Dynamic Hardware Plugins (DHP): Exploiting Reconfigurable Hardware for High-Performance Programmable Routers. In *Proceedings of the IEEE Openarch*, 2001.

[TY97] J. Turner and N. Yamanaka. Architectural Choices in Large Scale ATM Switches. Technical Report WUCS 97-21, Washington University in St. Louis, May 1997.

[VCY03] M. Venkatachalam, P. Chandra, and R. Yavatkar. A highly flexible, distributed multiprocessor architecture for network processing. *Computer Networks*, 41, 2003.

[Wan98] M. Wannemacher. *Das FPGA-Kochbuch*. International Thomson Publishing, 1st edition, 1998.

[Whi97] P. P. White. RSVP and Integrated Services in the Internet: A Tutorial. *IEEE Communications Magazine*, May 1997.

[WLG98] D. Wetherall, U. Legedza, and J. Guttag. Introducing New Internet Services: Why and How. *IEEE Network Magazine*, May/June 1998.

[WPF03] T. Wolf, P. Pappu, and M.A. Franklin. Predictive scheduling of network processors. *Computer Networks*, 41, 2003.

[Wro97] J. Wroclawski. *RFC2210: The Use of RSVP with IETF Integrated Services* . Internet Engineering Task Force (IETF), September 1997.

[WSS+01] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, and S. Waldbusser. *RFC3198: Terminology for Policy-Based Management*. Internet Engineeering Task Force (IETF), November 2001.

[WVTP97] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner. Scalable High Speed IP Routing Lookups. In *Proceedings of the ACM SIGCOMM Conference*, 1997.

[WZ98] R. Wittman and M. Zitterbart. AMNet: Active Multicasting Network. In *Proceedings of the International Conference on Communications (ICC'98)*, 1998.

[Xil] Xilinx Corp. *Virtex-II Pro X Platform FPGAs Complete Data Sheet*. Available at http://www.xilinx.com.

[XN99] X. Xiao and L.M. Ni. Internet QoS: A Big Picture. *IEEE Network*, 13(2):8–18, 1999.

[Zha95] H. Zhang. Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks. *Proceedings of the IEEE*, 83(10), October 1995.