Enabling Architectures for QoS Provisioning

Kim-Khoa Nguyen

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy in Electrical Engineering at
Concordia University
Montreal, Quebec, Canada

February 2008

Your file Votre référence
ISBN: 978-0-494-37756-7
Our file Notre référence
ISBN: 978-0-494-37756-7

Abstract

Enabling Architectures for QoS Provisioning

Kim-Khoa Nguyen, Ph.D.
Concordia University, 2008

Nowadays, new multimedia services have been deployed with stringent requirements for Quality of Service (QoS). The QoS provisioning is faced with the heterogeneity of system components. This thesis presents two research: on architectures for QoS management at the application layer, fulfilled mainly by software components; and on distributed software architectures for routing devices providing desired QoS at the underlying communication layer.

At the application layer, the QoS architecture we propose, based on the Quality Driven Delivery (QDD) framework, deals with the increasing amount of QoS information of a distributed system. Based on various QoS information models we define for key actors of a distributed system, a QoS information base is generated using QoS information collecting and analysis tools. To translate QoS information among different components, we propose mechanisms to build QoS mapping rules from statistical data. Experiments demonstrate that efficient QoS decisions can be made effectively regarding the contribution of all system components with the help of the QoS information management system.

At the underlying layer, we investigate distributed and scalable software architectures for QoS-enabled devices. Due to the huge volume of traffic to be switched, the traditional software model used for current generation routers, where the control card of the router performs all the processing tasks, is no longer appropriate in the near future.

We propose a new scalable and distributed architecture to fully exploit the hardware platforms of the next generation routers, and to improve the quality of routers, particularly with respect to scalability and to a lesser extent to resiliency and availability. Our proposal is a distributed software framework where control tasks are shared among the control and line cards of the router. Specific architectures for routing, signaling protocols and routing table management are developed. We investigate the challenges for such distributed architectures and proposed various solutions to overcome them. Based on a general distributed software framework, an efficient scalable distributed architecture for MPLS/LDP and different scalable distributed schemes for the routing table manager (RTM) are developed. We also evaluate the performance of proposed distributed schemes and discuss where to deploy these architectures depending on the type of routers (i.e., their hardware capacity).

Acknowledgements

I would like to acknowledge the help of many people during my doctoral study. First, I would like to thank my supervisor, Professor Brigitte Jaumard, for her invaluable directions and support throughout my study. Her directions with regard to the developments of my academic thinking, problem solving and technical writing were inspiring and will be very helpful in my future work and study.

I also would like to express my warmest thank to my co-supervisor, Professor Brigitte Kerhervé, who kindly allows me to begin my work in her lab. She has helped me greatly in the hardest years and has always been extremely supportive and nice to me. It was only her great help that I got the chance to initiate and complete this thesis.

I am grateful to the Département d'Informatique at Université du Québec à Montréal, Département d'Informatique et de Recherche Opérationelle (IRO) at Université de Montréal, Department of Electrical and Computer Engineering at Concordia University, and Concordia Institute for Information Systems Engineering (CIISE) for their generous support throughout my research.

This work was financially supported by grants from Canadian Institute for Telecommunication Research (CITR), MITACS network (Mathématiques des Technologies de l'Information et des Systèmes Complexes) and the former HyperChip Inc. I would particularly like to thank Marc Lanoue for discussion and guidance in the development of router architectures.

I would like to acknowledge Dr. J. William Atwood and Dr. Pierre Rolin for reviewing this manuscript. Their valuable comments improved the manuscript notably.

I am also thankful to all current and former members of the ORC lab for their support and assistance.

Last but not least, I sincerely thank my parents and my sister, for their love and encouragements during the past years. Without them all I have achieved would not have been possible.

TABLE OF CONTENTS

LIST OF ACRONYMS

ATM	Asynchronous Transfer Mode
BGP	Border Gateway Protocol
CLI	Command Line Interface
CORBA	Common Object Request Broker Architecture
CSPF	Constrained Shortest Path First
eNP	Egress Network Processor
eTM	Egress Traffic Manager
FEC	Forwarding Equivalence Class
FIT	Forwarding Information Table
FTN	FEC-To-NHLFE
GIF	Graphic Interchange Format
ICMP	Internet Control Message Protocol
IDL	Interface Description Language
ILM	Incoming Label Mapping
iNP	Ingress Network Processor
IP	Internet Protocol
IS-IS	Intermediate System-to-Intermediate System
iTM	Ingress Traffic Manager
JMF	Java Media Framework
JPEG	Joint Photographic Experts Group
LAT	Label Allocation Table
LDP	Label Distribution Protocol
LER	Label Edge Router
LIB	Label Information Base
L-ROUTE	Local Route Base
LSP	Label Switch Path

LSR	Label Switch Router
MIB	Management Information Base
MIPS	Millions of Instructions Per Second
MOS	Mean Opinion Score
MPLS	Multi-Protocol Label Switching
NHLFE	Next Hop Label Forwarding Entry
OC	Optical Carrier
OSPF	Open Shortest Path First
PPP	Point-to-Point Protocol
PSNR	Peak Signal Noise Rate
QDD	Quality-Driven Delivery
QoS	Quality of Service
QoSIB	Quality of Service Information Base
RDL	Resource Description Language
RIB	Routing Information Base
RSVP	Resource ReSerVation Protocol
RSVP-TE	RSVP Traffic Engineering
RTCP	Real-Time Control Protocol
RTM	Routing Table Manager
RTP	Real-time Transport Protocol
SNMP	Simple Network Management Protocol
SPF	Shortest Path First
SS	Streaming Server
TCP	Transmission Control Protocol
FIT	Forwarding Information Base
UDP	User Datagram Protocol
VP	Video Provider

LIST OF FIGURES

LIST OF TABLES

Introduction

Due to the evolution of distributed applications, systems and user requirements over the last ten years, we have seen an emergence of QoS architectures to provide end users with services of required quality level. QoS architectures have been implemented on different parts of the systems; sometimes they can cover the whole system from providers to users. Critical parts of the systems where QoS architectures have been mostly deployed are distributed system platforms, operating systems, transport subsystems and networks. End-to-end QoS provisioning has also been already explored in several studies. Although end-to-end QoS is one of the most emergent research trends for distributed systems, no standard model has yet been approved. QoS mechanisms used in commercial products are quite diversified according to the different objectives of providers and users.

QoS architectures are built using QoS-enabled components, which can be software or hardware based. We view the QoS provisioning as consisting of five main activities [Nguy05]: specification, mapping, negotiation, adaptation, and monitoring. The QoS specification aims at configuring the QoS in different layers of the system; it is done using a specification language. The QoS mapping performs the translation of QoS representations into different system layers, in a way transparent to end users. The negotiation activity is an iterative process where one attempts to meet user requirements while taking into account the possibly limited resources of the system. Within a communication session, renegotiations may be invoked when the QoS manager detects a QoS violation or when users want to have a better quality level. A negotiation is only possible if QoS information from user and system is semantically comprehensible by the involved system components. This is done with the help of the mapping activity. The QoS adaptation is required to maintain as long as possible the quality agreement established at the negotiation phase. Finally, monitoring is used to determine the available system offers and to detect the degradation of quality during communication sessions. The monitoring is also often used to update a QoS information base [Nguy05].

The QoS enabled components allow the QoS management activities to be properly conducted. For example, user bandwidth levels can be specified by router configuration. The QoS negotiation can be done with the help of specific protocols. The QoS monitoring can be achieved by server or communication device features. In general, critical services are usually provided with the help of a QoS architecture implemented on top of QoS enabled devices [Wang04]. Thus, regarding the evolution of applications and hardware components, we believe that QoS architectures need to be studied on both sides: at the application layer and at the underlying layer, in order to improve the quality and efficiency of the user-oriented services.

This thesis presents a two-pronged research: on architectures for QoS management at the application layer, fulfilled mainly by software components; and on distributed software architectures for routing devices in order to provide desired QoS at the underlying communication layer. At the application layer, the QoS architecture we propose deals particularly with the increasing amount of QoS information related to the various components of a distributed system. At the underlying communication layer, we propose a new scalable and distributed architecture that is able to fully exploit the hardware platforms of the next generation routers, and to improve the quality of routers, particularly with respect to scalability and to a lesser extent to resiliency and availability.

Part I: QoS Information Management Architectures

The first part of the PhD thesis deals with the increasing amount of QoS information related to the different components of a system, which entails the need of a management approach. The diversity and heterogeneity of QoS information are the main challenges for QoS provisioning. In addition, existing QoS architectures focus principally on network and performance parameters, which are no longer sufficient nor efficient in order to take into account the new requirements from users and providers. For example, current users are longer interested in the data content itself instead of the networking performance. It recently appears that QoS should be considered from a broader perspective [Kerh06, Abde03] so that the contribution of all system components can be taken into account in order to provide the QoS. For example, providers are usually using resource allocation methods (i.e., upgrading their network connection capacity) in order to deal with increasing user quality requirements. This solution is costly and is not practical in some given situations, such as for mobile users whose capacity is rather limited by user devices. When QoS is considered at a larger scale, including user requirements and system capacity, a more appropriate solution can be adopted; for example, video data can be pre-compressed at the server side before the transmission, and then decompressed at the client side when it is received, so that the traffic volume can be smaller. The latter solution is less costly than the first one, however additional information about user devices (i.e., available decompression software and memory buffer for temporary storage), system capacity (i.e., processing resource for performing compression) is needed. There are also alternative solutions such as data prioritization, modality transformation, data transcoding, purpose classification or resource adaptation which can be used efficiently to deal with QoS provisioning. These solutions require information on the clients and servers, as well as on the cooperation between users and providers.

In order to support QoS sensitive applications, system-specific QoS mechanisms such as operating system (OS) scheduling mechanisms and network reservation protocols need to be controlled. This crosscuts the distribution transparencies offered by the middleware layer and reduces the portability and interoperability of applications. In order to deal with this challenge, distributed services should offer abstractions for QoS management and control mechanisms at the system level. A middleware layer seems a natural place for brokering between QoS requirements of applications and the QoS capabilities of operating systems and networks. Interfaces to applications, OS, resources and additional protocols are expected to appear in such a layer in order to deal with a changing run-time environment such as system and network load which influences the QoS capabilities.

Let us consider an example of a video streaming delivery service over the Internet. This service is composed of geographically separated components: servers and clients; the server accepts data input as rough image and audio captured by cameras and microphones. After being processed by acquisition devices, these data are sent to the client over the Internet platform. The client receives, then displays video clips to users [Wu01]. In such a system, the quality perceived by the user is depending on many factors. First, acquiring video by camera and microphone may introduce distortions due to optics, noise, etc. Digitalizing the image and the audio before transmission then may alter the original signal and produce distortion. The transmission channel may also add some noise to data and finally, the display device may introduce distortion, such as low resolution, bad calibration, etc. A QoS management system should therefore provide an interface allowing users to specify their requirements in a simple way. For example, users can express their requirements in terms of four human-comprehensible levels "excellent", "good", "fair" and "bad". This information should be mapped into a corresponding range defined for example by MOS or PSNR metrics [ATIS01]. Corresponding encoding techniques will be selected in order to satisfy these MOS or PSNR values. A negotiation process will be launched to allocate the underlying resources supporting the video encodings (Figure 1-1).

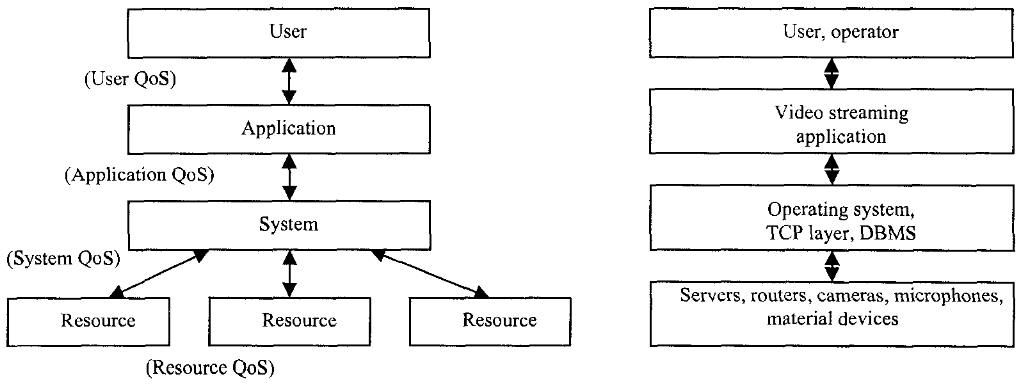

User
(User QoS)
Application
(Application QoS)
System
(System QoS)
Resource
Resource
Resource
(Resource QoS)
User, operator
Video streaming application
Operating system, TCP layer, DBMS
Servers, routers, cameras, microphones, material devices

Figure 1-1: QoS Layers of a Multimedia Distributed System and the Corresponding Components of a Video Delivery Application

The variety and diversity of QoS information lead to the necessity of having a QoS information base (QoSIB) to support the QoS management achieved by a middleware layer. The role of QoS information is decisive because it is essential for user specification and resource management and should be considered in the system architecture and design [Nguy04]. Considering the QDD (*Quality-Driven Delivery*) approach where the end user is positioned at the center of a service delivery model [Gerb03], a QoS information base should be provided with storing, accessing, transferring, producing and analyzing capabilities. The main advantage of a QDD approach is that we can use alternatively different management mechanisms (i.e., resource allocation or content adaptation) in the underlying system to provide QoS support. Located at the heart of the QDD approach, the QoS manager performs a complicated task of matching the user specification to system offer. QoS information management is therefore required to permit the management activities to access independently to a QoS information base, and also to transform different types of QoS information into a format the requiring component can understand.

Therefore, our research aims at proposing an architecture which will be able to improve the QoS provisioning based on QoS information management. Information models, mapping rules and decision making mechanisms are developed in order to provide the QoS according to user requirements.

Part II: Next Generation Scalable and Distributed Router Architectures

A core router is one of the most critical devices which are deployed to provide QoS in distributed systems. In the second part of the PhD work, we aim at proposing a new distributed software architecture that is able to fully exploit the new router hardware platforms and to improve the quality of routers, particularly with respect to scalability and resiliency. In order to achieve this goal, we investigate distributed and modular designs, where the routing software components can run independently on the same or on distinct CPUs and interact with each other regardless of their respective physical location. Such a design leads to a robust router which is not vendor specific and which can use modules developed by different component suppliers.

A router, especially a core router, should scale both in terms of the number of ports which can be connected to the router and in terms of the data forwarding capacity. Scalability means being able to add modules when capacity requirements increase, without impairing the switching performance. A router is resilient if it ensures that a failure of one software non-routing component does not affect the behavior of other independent routing software components. Availability is mainly due to two factors. Firstly, the modularity makes it possible to use redundancy and replication of critical functionalities over multiple modules. Secondly, the modular structure in itself tends to limit the impact of faults in individual modules, and encourages sound engineering design principles.

Basically, these features can be achieved by integrating more powerful processors and memory chipsets in router hardware platforms. Together with the increasing number of processors and memory capacity, router architectures have experienced much evolution and distributed architectures have been recognized as one of the most prominent trends [Chao07]. Such architectures have been investigated by several router providers and will be deployed in the next generation of core routers [Kapl02]. One of the highlighted features of distributed architectures lies in the sharing of some tasks between control cards and line cards, taking into account available processing and memory resources on the line cards. This improvement allows the router to accelerate the processing performance because some tasks can be done in parallel between control cards and line cards, or among different line cards. One example of the task sharing is the data packet processing where line cards achieve the data forwarding with routing information updated by control cards.

Our research deals with the design of distributed software architectures to be implemented on distributed hardware architectures of the new router generation. Basically, a distributed software architecture is required to fully exploit the efficiency of a distributed hardware platform. However, due to legacy techniques or business models, we can observe that, even in the recent router products [Juni07, Cisc05], some of the software components still remain centralized, particularly the routing protocol modules and the Routing Table Manager (RTM). Since the control card of a router is responsible for all routing tasks, it can easily be overloaded by overwhelming traffic in core networks, especially when the routing tables get flopped (i.e., updated/refreshed). In addition, bottlenecks can be experienced in a centralized architecture when the control card is unable to process fast enough the huge number of requests coming from different line cards. The time for route establishment and time to recover are also issues in the centralized architecture because every protocol message must go through the control card, leading to additional delay overhead. These limitations led to the design of a distributed architecture for software implementation, particularly for hyper-speed routers, such as the *petabit* routers [Tse04]. This thesis aims at the enhancement of the software architecture of the first distributed router products [Avic06, Juni07], and in particular the PBR1280 router of former HyperChip Inc. [Dupl05].

Some models for software distribution, such as the ForCES framework [Dori07], have recently been proposed to redesign the current router software. Based on ForCES, a distributed control plane architecture has been introduced for the functions performed by control cards and line cards [Deva03]. For example: i) link-specific functions are performed on line cards, ii) update functions are performed by control cards, and iii) protocol-specific functions still need to be considered on a case-by-case basis for distribution with no standard model. However, the proposed architecture does not consider the specific hardware architecture of next generation routers with a general purpose CPU and extra memory resources on line cards. The ForCES-based architectures can deal effectively with medium scale routers, e.g., routers having some tens of line cards and some hundred interfaces. However, core routers, and especially *petabit* routers, require enhanced distributed architectures in order to run on their high-scale hardware platform (e.g., thousands of line cards). Therefore, in this thesis, we propose enhanced distributed architectures, which enable efficient exploitation of the next generation router hardware architecture.

Contributions of the PhD Thesis

Our approach, dealing with QoS provisioning for distributed systems, is to propose QoS architectures in support of QoS information modeling and management and to design the core routers with high scalability, resiliency and robustness. To this end, we have made the following contributions:

In the first part of our research related to the QoS information modeling and management,

- We define an approach to support the quality-driven delivery based on a QoS information management system. We demonstrate through an experimental video streaming application that the current network-and-performance QoS architectures are not able to deal efficiently with the evolution of distributed system components and user requirements. QoS can be provided more efficiently based on QoS information of all components of a system. An efficient QoS decision is obtained only when all the mappings are taken into account. We describe a framework to reach such goal including the information management and building mapping rules. This work has been presented in [Nguy03], [Nguy04] and [Nguy05].

- We propose steps to build a QoS information management system based on QoS information models. QoS information of each system component is represented by a model with basic operations such as store, access and change. The QoS information models are organized in a hierarchical manner from generic to specific ones. The information management is therefore extensible in the sense that new service components can easily be integrated into the QoS management system. We also explore the different mechanisms to collect QoS information from system and users. This work has been presented in [Kerh06].
- We propose new mechanisms to build the QoS mapping rules in a flexible and dynamical manner. QoS mapping rules are derived from available system working state statistical information using data mining techniques. This work has been presented in [Nguy06].
- We define processes for making the best possible QoS decisions using a QoS information management system built with QoS information models and mapping rules of a distributed system. We also develop an experimental video streaming delivery system which is able to make various QoS decisions with the help of an

independent QoS information manager considering the cost of resource utilization. Such a system suggests alternative QoS management mechanisms, which are not available in current video streaming systems, allowing the system administrator to make cost-effective decisions.

In the second part of our research related to the design of core routers,

- We propose novel distributed and scalable architectures to implement the control plane for next generation routers. This contribution consists of a general framework involving different distributed software architectures for routing and signaling protocols. They can be used to effectively exploit the distributed hardware of the next generation routers. The scalability and resiliency is also improved. This work has been presented in [Nguy07a], [Nguy07b] and [Nguy07f].
- We design a distributed architecture for MPLS/LDP based on the general distributed architecture for signaling protocols. We provide mechanisms for message exchange, table and session management so that the MPLS/LDP architecture can be fully distributed on next generation router platforms. This allows the control functions to be implemented on the line cards of the router and then avoids congestion on the control card. We also evaluate the performance of the proposed distributed architecture in comparison with the traditional centralized architecture. This work has been presented in [Nguy07d].
- We propose distributed architectures for routing table management. We demonstrate that the routing table management can also be distributed on new

router hardware platforms in order to improve the scalability in route processing. Based on different types of routers, we present different distributed mechanisms for routing table management and evaluate them in terms of CPU consumption, memory usage and the number of exchanged messages. We also provide the design of a distributed Routing Table Manager for highly scalable distributed routers. This work has been presented in [Nguy07c], and [Nguy07e].

Outline of the PhD Thesis

The remainder of this thesis is organized in two parts.

The first part of the thesis, dedicated to the QoS architectures at the application layer, includes chapters 1, 2 and 3. In Chapter 1 we present the background and motivation of research, where we focus on the QoS provisioning at the application layer and its issues, particularly for distributed multimedia systems. We also review and analyze some emergent QoS architectures for distributed multimedia systems. In Chapter 2, we present our approach for QoS information management based on the Quality-Driven Delivery framework. QoS information models and QoS mapping mechanisms are provided. Chapter 3 describes the QoS decisions and applications where we demonstrate that better QoS decisions can be made using our QoS information management system.

The second part of the thesis, dedicated to new software architectures for the next generation routers, includes chapters 4, 5, 6 and 7. Chapter 4 presents the background and motivation of the research. We discuss the evolution of routers, their hardware and software components and especially the main features of the next generation routers. We also review the previous work which has been published on distributed software architectures. In Chapter 5, we propose a new generic distributed scalable framework for software implementation for the next generation routers. We also present new distributed architectures for routing and signaling protocols and discuss the possibility of using these architectures to implement the OSPF and MPLS modules. Chapter 6 describes the details of the proposed distributed architecture for MPLS/LDP with further developments and a performance evaluation. Chapter 7 provides different possible distributed architectures for RTM and gives details of the selected architecture. We also evaluate the performance achieved by the proposed RTM distributed architectures and discuss the ability of using them for different types of routers

Finally, we conclude the thesis with the lessons learned and some thoughts about future work.

PART I. QOS INFORMATION MANAGEMENT ARCHITECTURES

Chapter 1 QoS Provision at the Application Level

In this chapter, we review the QoS provisioning and QoS issues for the distributed systems. We describe the principles for QoS provisioning at the application level and we focus on the QoS mechanisms such as resource allocation, adaptation and content optimization. We also investigate the current QoS architectures and analyze their insufficiency regarding the evolution of the current distributed systems and user requirements. Finally, we motivate and provide the main objectives of our research.

1.1 Distributed Systems and Applications

Contemporary distributed multimedia systems are widely spread across various platforms and infrastructures, such as telephone, dedicated optical or DSL lines. The development of inexpensive high speed networking technology enables a new computing environment where applications can be connected and share multimedia documents. The distributed multimedia systems usually handle a large amount of multimedia content, which is distributed across networks. This shared computing environment hold several advantages over the traditional computing environment where each application has its own dedicated purpose computing hardware. The emergent features of the new distributed multimedia applications include: higher resource utilization, better manageability and lower cost [Nahr00].

One of the main characteristics of the new distributed multimedia systems is the heterogeneity. In addition to the various platforms, multimedia documents and technologies, these systems typically go through independent upgrade and procurement cycles which lead to heterogeneity over time.

Distributed multimedia applications can be classified along four dimensions: the task they perform, the type of media they involve, the location of the operations (e.g., geographical dispersion of users) and the behavioral characteristics of users (e.g., user expectations, skills) [Mira02]. The system architecture supporting such applications is usually heterogeneous, consisting of a large number of client machines, database servers, video servers or other specific servers, all interconnected through communication networks. These complex environments require the integration of system management mechanisms providing system scalability, application adaptation and quality of service (QoS) support.

In legacy systems, access to the services such as network bandwidth, processing time and data, follows a best-effort policy. The adoption of such a policy results in unpredictable behavior in service provisioning. Current distributed multimedia applications cannot tolerate uncertainty in relation to access to data and computational resources. Such applications are said to have quality of service (QoS) requirements where users demand that the availability of the resources used by them be predictable. QoS support was initially introduced in the field of telecommunication networks and multimedia systems and led to proposals for management strategies aimed at deciding whether and how multimedia streams can be delivered to the user with some constraints. In the context of distributed systems, QoS is considered as the ability to adapt system offers to user requirements, taking into account several aspects: communication channel, client applications and the nature of the service itself. QoS used in the system references to the architectures ensuring the delivery of service from end to end or from application to application.

Traditionally, mechanisms for resource management are added to computational systems in order to make access to computational resources in a predictable fashion. The term "QoS architecture" is used to describe middleware which provides applications with mechanisms for QoS specification and enforcement. These architectures organize the resources provided by the system with the intent of fulfilling the QoS requirements imposed by the application.

For example, the video delivery service presented in the Introduction can be enhanced with QoS management middleware in order to provide the video streaming service with higher quality constraints. The QoS architecture provides an interface allowing users to specify their requirements and then interacts with device managers, or uses OS functions, in order to allocate resources satisfying user requirements. Various types of hardware, operating systems and network infrastructures coexist in distributed systems, and multiple resource reservation protocols can be deployed in this environment. Consequently, allowing applications to manage resources via a middleware layer implies that the differences between resource reservation protocols have to be handled by the middleware. Therefore, the traditional QoS architectures strongly depend on computing platforms and suit a specific application area.

Generally, a QoS architecture is achieved based on the cooperation of system components, including communication network, applications and management software, aiming at providing the service of quality to users. This leads to the fact that QoS can be studied from different perspectives: resource management [Nahr99, Abde03], database management [Ye03] and agent-based and cooperative management [Phan03]. The common objective of QoS architectures for distributed systems consists in the provision of high quality communication from end-to-end transparently to the users. Service layering, resource allocation, and optimization are processed inside the system. Currently, there is not a specific approach for QoS in distributed systems, but some researchers have proposed to use layering [Wang00, Gu05, Yuan06] or object oriented [Gill05, Duza04] architectures.

We next present the issues for QoS provisioning in such architectures.

1.2 QoS Issues

QoS is widely used in various fields of data processing. The term QoS has been introduced originally in the telecommunications area to describe the operational requirements for the quality of communication such as bandwidth, delay, error rate, reliability or availability. The QoS is delivered based on contracts between providers and users. A typical contract, so called SLA (*Service Agreement Level*), specifies, usually in measurable terms, the quality levels the provider has to furnish. Based on the SLA, the provider allocates the resources (e.g., server or routers) and configures the links to satisfy the required number of users or volume of traffic [ICS00]. With a layer-based communication system such as the OSI model, the higher layers uses the services provided by the lower layers in order to achieve user communications. Due to the QoS constraints, these layers need to exchange functional information, such as bit rate, throughput, etc., and non-functional information, such as availability, reliability, etc., out of the data and control messages. This leads to some challenges because the legacy systems and protocols have not been designed to support this information. Some extensions or complementary protocols, such as DiffServ or IntServ, may be used to deal partially with these issues. However, they mainly focus on the functional information and only a few parameters are taken into account, as shown in Table 1-1.

In order to provide the QoS, the network administrator can use various QoS control mechanisms. Most legacy Internet applications, such as FTP or email, may accept the *best-effort* QoS where providers promise to deliver the best service they can, but do not guarantee a high-quality connection. Contemporary applications, such as video streaming, require a bandwidth larger than a predefined minimal threshold to be working properly. Networking resources, such as the router ports, are therefore reserved for the demand, based on the *network management policies* and *access control* mechanisms. Most current networking devices allow users to configure these policies. However, due to the heterogeneity of network equipment and management software, the exchanges among devices and networks remain a challenge in order to establish a QoS communication from end-to-end. The model widely used to specify the QoS parameters of the current networking devices is CIM, created by DMTF (*Distributed Management Task Force*) [DMTF07]. However, this model is not yet completed for all networking services.

Architecture / Characteristic	Internet /IETF IntServ and DiffServ	ITU/ATM Forum	OSI
Throughput	X	PCR[1]/ SCR[2]	X
Delay	X	CTD[3]	X
Jitter		CDV[4]	
Error ratio		CLR[5]	X
Availability		X	X
Priority	X		
Connection Mode		X	
Reliability			X
Cost	X	X	

Table 1-1: QoS Characteristics Considered by Different Organizations

The QoS problem is also met in the field of software engineering to describe the non-functional characteristics of a system. In general, a software system is composed of several components with many depending interactions among them. Setting the QoS characteristics of a component may have a direct impact on the architecture of the whole system. The QoS should therefore be taken into account from the design phase of the software development process. In [Frol98], authors proposed to characterize the QoS along *dimensions*. A set of dimensions, sharing common criteria, can be gathered together into a *category*. A dimension consists of a name and a domain of values. There are three types of domains of values: *numeric*, *enumerated* and *set*. The QoS specification is then based on the QoS contracts, which define the values of the dimensions. A profile associates a contract to users, interface, operations, etc.

[1] PCR : Peak Cell Rate [2] SCR: Sustained Cell Rate
[3] CTD: Cell Transfer Delay [4] CDV: Cell Delay Variation [5] CLR: Cell Loss Ratio

The QoS support in distributed systems involves issues addressed from both the telecommunication networks and the software engineering points of view. Research considers the QoS as the ability to adapt the system offers to the user requirements, taking into account various factors such as the communication systems, the client applications, the service architectures, etc. In particular, one of the issues for distributed systems is that QoS is usually specified and interpreted by users and providers in different ways. Users are concerned with the characteristics of the applications while providers are more interested in the system parameters and resources. This leads to the need of mapping mechanisms in order to unify different information.

We next describe mechanisms for QoS provisioning and then discuss the requirements for QoS architectures.

1.3 QoS Provisioning

As presented in the previous section, distributed service delivery demands QoS support, especially for QoS sensitive applications such as multimedia and e-commerce services. QoS provisioning is enforced by SLA contracts and QoS violation is usually related with financial penalty.

Traditionally, the most important concern of distributed multimedia systems is bandwidth allocation to user. Since the multimedia transmission requires a lot of networking resource while networking equipment is often costly. In order to deal with such a challenge, QoS mechanisms can be deployed, such as:

a) Resource allocation. Allocation and re-allocation are fundamental methods to address QoS violation problems. One of the most used models for resource allocation is

Integrated Services, based on RSVP protocol (*Resource Reservation Protocol*) that sets up paths and reserves resources in the network. RSVP provides end-to-end QoS services on per-flow basis. Thus, when bandwidth is decreasing, RSVP can be invoked to reserve more network resources. In the context of multimedia networks with multiple components, resource allocation is a multi-dimensional problem taking into account resource information profiles, the application requirements and utility functions.

b) Adaptation. QoS adaptation is used to maintain as long as possible the service level agreement built at the negotiation phase and can be achieved at the client side or server side. In case of QoS violation, QoS adaptation is performed transparently, in such a way that the system transition takes place from one state to another state to provide the requested level of service. QoS adaptation differs from conventional management function due to its real-time characteristics. Thus, when bandwidth to users is decreasing, available adaptation mechanisms can be: changing the network path or changing the server providing the service. In general, the majority of adaptation mechanisms can fit into three main classes [Nguy05]: resource control, reconfiguration and change-of-service. The first class includes the mechanisms making fine adjustments to individual resources in the distributed system. The second class performs the adaptation by altering the topology of the end-to-end processing. The third class allows users to prioritize services and adjust as necessary.

c) Content optimization: Content optimization can also be used as well as adaptation or allocation to deal with QoS violation problems. When bandwidth is decreasing, possible content optimization techniques consists in compressing data content, or changing the codec. For example, [Abde99] proposes a technique replacing GIF images by JPEG images, which may reduce transmission overload more than eight times. [Shah01] presents a classification of content optimization techniques including: i) information abstraction for reducing bandwidth requirement, ii) data prioritization for providing different QoS levels, iii) modality transformation for transforming content adaptively to a particular device, iv) data transcoding for enabling universal access using pervasive computing device, and v) purpose classification for eliminating redundant information.

To the best of our knowledge, a good QoS provisioning mechanism should have the following characteristics:

- *User satisfaction.* The service provision has to meet user requirements, in terms of desired quality levels (e.g., service time and loss ratio). With the distributed multimedia systems, there are usually multi-dimensional requirements at the same time. For example, a user can have different requirements on the image, audio and text. The system consists of different platforms and infrastructures needed to be considered together. A user requirement can also be achieved by different components, which can proceed in parallel.
- *Efficient resource utilization.* In addition to satisfying user QoS requirements, we also want to achieve efficient resource consumption. This problem has different implications in different environments. For example, in the pervasive computing environment, we need to overcome resource constraints of mobile devices such as limited memory capacity. However, when we look at the peer-to-peer computing environment, the problem becomes how to efficiently distribute the load among different peers to achieve optimal load sharing.

- *Flexibility*. The service provision should be able to adapt to the change of environment and user requirements as fast as possible. Besides, most current QoS provisioning mechanisms consider only a limited number of QoS dimension or user requirements. As a result, the flexibility and efficiency is limited when the system is growing. The multimedia distributed system requires QoS provisioning mechanisms that are extensive, to support the heterogeneity of its components.

It recently appears that resource allocation and adaptation can be combined to provide QoS in distributed systems [Fost01, Kerh06], but we believe that an optimal decision will not be easy to obtain if all QoS dimensions are not considered together. The user requirements should be positioned at the center of the QoS framework as they are the ultimate objective and QoS information of all components of the system should be taken into account. We will review the current QoS architectures and then come back on this discussion in the next section.

1.4 Review of Current QoS Management Frameworks

The first QoS architectures for distributed systems have been introduced in the 90s [Aurr98]. Most of them have been developed in the context of research projects and dealt only with some specific QoS problems. Until very recently, there have been many attempts to generalize and standardize QoS architectures in order to build a system which covers all QoS aspects of distributed systems but no commercial product has been introduced. Some notable architectures are:

- The QoS framework of the MONET research group. It is a set of QoS architectures providing services and protocols for end-to-end QoS service guarantee for distributed multimedia applications. They defined different projects for QoS resource management, QoS middleware and QoS routing. Most interesting architectures include 2KQ [Nahr00], QoSTalk [Gu01, Gu05] and GRACE [Yuan06]. Some script languages for QoS information specification have been developed to describe user requirements and functionality of applications, systems and resources. The translation of user requirements, considered as generic information, into specific resources parameters is done via composite system information. The main contributions of MONET include:

- A XML-based QoS enabling language. This is an extended version of XML, so called HQML, used to specify QoS at user, application and resource levels. Since XML is an interactive Web language, HQML can reuse this feature to report QoS violations. A mechanism to map user requirements to a predefined set of resource parameters is also developed.
- A mechanism, so called 2KQ, which partitions the end-to-end QoS setup process into distributed QoS compilation and run-time QoS instantiation phases for different types of applications.
- A service composition program, so called SpiderNet, which combines different components in order to achieve a requirement.
- A cross-layer design, which allows the resources to adapt to application requirements based on the coordination of the operating system.

Basically, the researches of the MONET group have covered many aspects of the end-to-end QoS provisioning problem. However, the number of QoS parameters and the system components are limited. In addition, they do not take into account different types of mapping.

- The QoS architecture for multimedia application developed by Hafid and Bochmann [Hafi99, Serh05]. This architecture focuses principally on QoS negotiation and re-negotiation. It takes into account the static information identifying the service and the dynamic information characterizing the communication sessions. QoS information is defined by profiles for users, applications and resources. The mapping consists of transformations among different profiles. The negotiation process tries to determine a functional configuration of the system that can support user QoS requirements. This architecture has a QoS manager, which maintains a set of functional configurations of the system. For each user requirement, it selects the optimal configuration considering the cost of service. The maximal bit rate and minimal bit rate are the principal parameters of the system. Some optimal algorithms for QoS decision making are also taken into account.
- QuO [Zink97, Wang04]. It is an object-oriented architecture with certain QoS capabilities. It defines three specification languages, called CDL (*Contract Definition Language*), RDL (*Resource Definition Language*) and SDL (*Structure Definition Language*), used to describe the QoS of application, resource and system respectively. QoS specifications are defined by developers for specific objects when the application is being built. Mapping from object specification to the CORBA platform is done when the application is compiled. The features of QuO object design include:

  - Integrating knowledge of the system properties to the objects. This allows the object to be aware of the environmental conditions.
  - Reducing the variance in system properties. This allows the QoS to be provided in some predefined levels.
  - Using design patterns. This allows the objects to be programmed in a systematic way.
  - Supporting code reuse and generation. This eases the application development.

  Applications may use QuO object services through the contracts. A contract consists of:

  - A set of functional states. Each state corresponds to a QoS level. There is an activating condition for each state.
  - Transitions between the different states.
  - A reference to the object representing the environmental condition, through which the applications are aware of the system QoS.
  - A notification mechanism to inform the application about the transitions of the states.

  The contract does not contain information about the cost of service. Thus it is impossible to compute the best QoS decisions in terms of the cost.

- TAO [Schm99, Gill05]. It is also an object-oriented architecture based on the CORBA distributed middleware, as QuO. Unlike QuO, TAO focuses rather on the processing performance (in terms of execution time) and event scheduling than on the QoS specification. TAO added some QoS features to the standard CORBA bus such as: i) the ability of QoS specification and validation, ii) real-time characteristics, and iii) the performance optimization. QoS information is defined by the IDL interfaces CORBA provides. The QoS of the transport layer and the operating system can be specified, based on the following parameters.

- Transport layer: sending buffer size, receiving buffer size, keep alive time, routing permission, delay permission, network priority enabling.
- Operating system: worst-case execution time, cached execution time, period, criticality, importance, quantum operation, dependency information.

TAO provides predefined functions allowing the applications to specify QoS levels based on the above set of parameters. Some additional mappings have been added so that QoS specifications can be translated to the CORBA platform parameters. However, in some cases, TAO requires users to define the mapping rules in *ad-hoc* manner, for example when DiffServ is used.

Table 1-2 summarizes the main features of the five architectures described above. The aspects taken into account include: implementation of the architecture, specification levels, QoS information, transmission support, mapping mechanisms and the monitoring methods. As we can see, the current QoS architectures provide intermediate QoS controllable layers for distributed applications. There are two models for these architectures: layer-based and object-based. The layer based architectures connect the applications to the resources by building a QoS enabled middleware layer. Only a limited number of logical resources is considered in these architectures, where they mainly focus on the communication network. User requirements are performance related parameters.

The object based architectures define the software objects with QoS capability. Each object can achieve a limited set of QoS functions. Applications may use the services provided by these components to satisfy user requirements. The physical resources are not presented and these architectures are based on an abstract resource management layer (i.e., CORBA). They mainly deal with the execution time and task sharing for heavy resource consuming applications.

QoS architecture	**Implementation**	**Transmission support**	**Specification levels**	**QoS information**	**Mapping**	**Monitoring method**
MONET framework (1996-2007)	layered	High speed (ATM network)	- User - System - Resource	3 types: - generic - composite - specific	Compilation of generic information in terms of specific information	Specific methods
Hafid & Bochmann (1999-2007)	layered	Best effort (Internet), but often use high quality network for multimedia applications	N/A (some simple user interfaces)	2 types: - static - dynamic Represented by profiles	Mapping user specifications to system configurations	Tools available from infrastructure or resource reservation protocols
QuO (1997-2006)	object-oriented	CORBA middleware	- User - System (object) - Resource Using QDLs languages	2 types of information : - negotiable - real	N/A (defined by CORBA)	Available CORBA methods
TAO (1999-2007)	object-oriented	CORBA middleware	pre-defined IDLs	system execution time	Mapping CORBA object to specific platforms	Available CORBA methods

Table 1-2: Comparison of QoS Architectures

Both types of the current QoS architectures are not extensible to integrate new QoS parameters, especially from underlying devices. They manage QoS information in a *hard-coded* manner. The implementation of mapping rules between layers or components is not presented. Only a QoS solution is considered within the set of possible QoS solutions regarding the different components of the system.

Through this review, we recognize that the evolution in the fields of telecommunications, network management, software and distributed systems has created new QoS information for specific devices, architectures and software systems. It is therefore essential to have a systematic approach allowing the management of this information. However, in the QoS architectures presented in this section, the information management is achieved in an *ad-hoc* manner and is integrated directly into the code, leading to the following disadvantages:

- The modification of a QoS parameter, if needed, requires modification of the source code of applications or even modification of system architecture.
- The extension and reengineering of system become difficult because the definition of a new QoS parameter will require changing the system design and implementation.
- It is impossible to modify a QoS parameter when the application is running.
- There is no flexible mechanism to express the relationships between different QoS parameters.
- The maintenance of the functions providing QoS can be costly.

The QDD framework we propose in Chapter 2 is a novel approach for QoS information management, focusing on QoS information modeling and mapping, which is able to deal with such issues.

The outline of our approach will be provided in the next section where we motivate the research problem.

1.5 Motivation

The trend recognized through our literature review is that the evolution of applications in different domains (e.g., telecommunication, network management, software engineering, distributed system) requires QoS information for describing user specifications and system offers. The volume of QoS information is increasing as new devices and applications with specific features are developed. In addition, QoS information becomes more diversified and heterogeneous due to various system architectures and services. We believe therefore that a QoS information management system should be provided, allowing QoS management activities to obtain appropriate QoS information with accurate formats and contents. The QoS information management should be achieved by a QoS information manager, located at the center of the QoS management approach, providing the required QoS information to users, providers and all components contributing to the quality delivery process. However, such a QoS information manager has not been taken into account in the existing QoS architectures where the QoS information management is hard-coded instead. The approach we propose in this thesis, focusing on modeling and transformation, aims at separating the QoS information management tasks from the QoS management architectures so that the QoS information management can be achieved more efficiently and flexibility.

The ultimate goal of our research is to propose a new approach for QoS information management for multimedia applications in the context of distributed systems. More specifically, our objectives are to design QoS information models for multimedia applications; design the methods to create these models, and to generate and store QoS information; define the QoS mapping activities for different QoS architectures; develop the mechanisms for generating the mapping rules; and validate our approach through the possible QoS decisions made for an experimental distributed application.

1.6 Chapter Conclusions

In this chapter, we have presented the fundamentals of QoS provisioning for the distributed multimedia systems. The main concept of the QoS is the QoS characteristics, so called dimensions, which are used to describe the non-functional parameters of a system. The QoS is provided to end-users by the cooperation of the components contributing to the service. It is guaranteed by a contract between users and providers. Essential mechanisms used to provide the QoS include: resource allocation, adaptation and content optimization. In order to select the appropriate QoS mechanisms, the contribution of all the service components should be taken into account.

The current QoS architectures are dealing with specific QoS problems, e.g., for a given service or system. They can be based on layering or object oriented architectures. Only some specific layers or service components are considered and the QoS decision usually concerns resource allocation. We also pointed out that the current QoS architectures are not designed in an extensible way, so the integration of new service components is not possible. The QoS information management is not considered as a separate module in the system, raising the issues of QoS management and QoS decision making.

Our proposal is a QoS management approach that is able to select the optimized QoS decisions based on the available QoS information of all service components of the system. To this end, we will investigate the QoS information management mechanisms in the next chapter where we focus particularly on the information modeling and mapping.

Chapter 2 QoS Information Management

This chapter presents QoS information management and our approach to manage the QoS information efficiently. We begin by investigating the QoS management activities and the need of QoS information for QoS provisioning. We then position our approach in the context of the QDD framework. QoS information modeling is presented to support the QDD framework, and we explain the mechanisms we propose to collect and manage QoS information. We also focus on the QoS mapping activity and then we propose an approach to build the mapping rules to translate QoS information among different service components.

2.1 QoS Management Activities

In a distributed service model, the service must be delivered transparently from provider to users. The difference between provider and users leads to two different points of view on the service: the one of the provider is directly related to the distributed system supporting the service; the one of the user concerns the application and human perception. While users expect that QoS information can be described in the simplest possible way, the provider usually tries to detail as much as possible this information in terms of physical resource parameters. This paradox results in the classification of QoS information into two categories: qualitative and quantitative. Qualitative QoS information is used primary by users to specify their requirements. Qualitative information is not directly measurable and can be expressed in terms of quantitative information in the context of a given service. Within a simple specification, qualitative information can be expressed by users in terms of very abstract notions such as “Excellent”, “Good”, “Fair” or “Bad”. The provider, on the other hand, describes and quantifies the QoS he offers in terms of measurable information, such as “transmission rate 50 frames per second” or “processor speed 10 MIPS”.

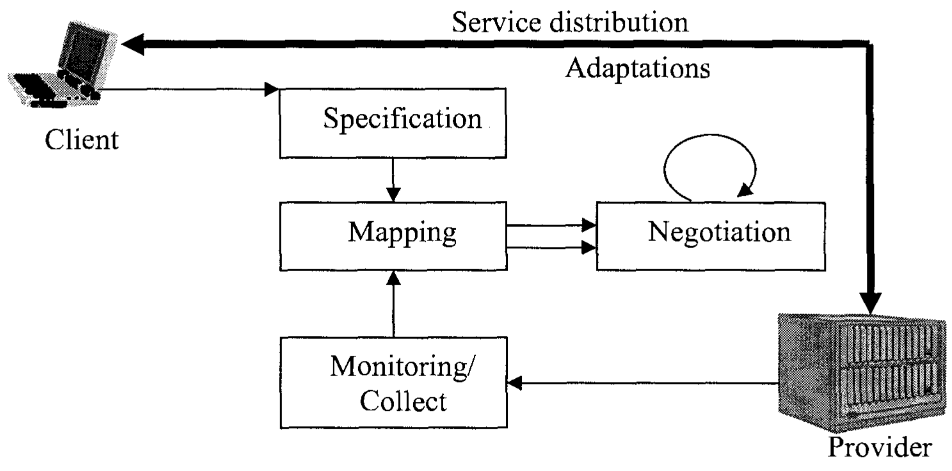


Figure 2-1: QoS Activities within a Service Distribution Session

One of the main activities of QoS provisioning is the information processing, which is achieved by different system components. For example, a video streaming application [Wu01] composed of geographically separate servers and clients has to take care about the quality of raw images and audio data, communication channel and software applications (Figure 2-2).

In order to provide the video streaming with the quality satisfying the user requirements, such a system is built with a QoS Controller component. Since the server and the client are geographically separated, the QoS Controller should be implemented on both the server and the client sides. On the server side, the QoS Controller is responsible for combining the encoded video image and audio, and reducing the noise.

On the client side, the QoS Controller separates the video and audio in order to display them on separate channels. User QoS requirements are transmitted to the streaming server through the communication between the two QoS Controller modules. From the user point of view, the quality of a video sequence is traditionally expressed in terms of the perceptive performance such as image resolution or frame rate. The ultimate goal of QoS provisioning is to allocate and control all the physical and logical resources contributing to the multimedia transmission session in order to deliver the service satisfying the user requirements.

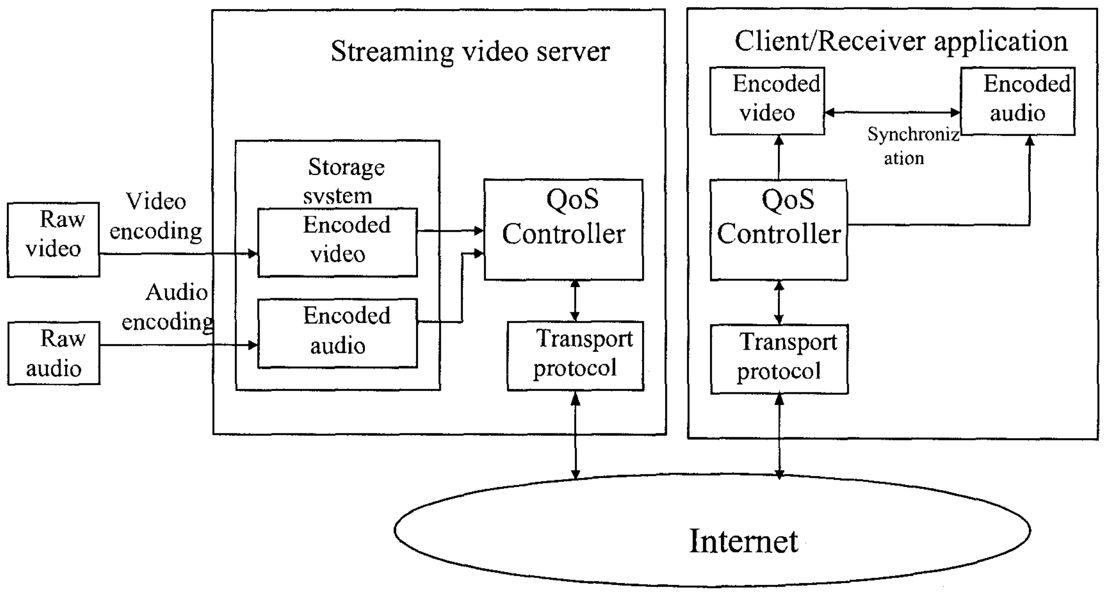


Figure 2-2: Architecture of a Video Streaming System

Therefore, the challenges for QoS provisioning include:

- The mapping between different levels of the system. The qualitative information at the user level must be mapped into qualitative and/or quantitative parameters at the network and system levels.
- The access control according to the quality levels.

- The validation of QoS in cases of dynamic changes of system and network. The QoS provisioning is expected to adapt dynamically to the changing condition along the communication session.

In the context of multimedia applications, qualitative information describes the quality of multimedia objects the user receives, including voice, video and data. Samples of multimedia applications include video conferencing, video on demand or distant learning. The provision of QoS for distributed multimedia systems must take into account different operations performed on multimedia objects, namely visualizing, editing, converting, processing, retrieving and analyzing.

Basically, QoS specification starts with collecting information from users and system components. Most current systems provide users with a graphical interface allowing them to declare their required quality level and the acceptable cost. On the other hand, QoS information about the system offer is collected by monitoring tools. Sometimes, data collected by monitoring tools consist of a set of values, while users need only certain particular ones (i.e., medium, minimal or maximal values). In such cases, output data should be pre-processed to get the desired values. The data analysis is also needed to filter the valid values when there is a negative influence of the environment on the results.

Since QoS parameters of each system component have different meanings, mapping is required. This operation aims at transforming QoS information. The mapping process helps to allocate system resources according to the user's demand. As current systems consist of many different layers or components, the mapping activity is required to make them cooperate in such a way that output information from one component can be understandable by other components. For example, in order to allocate router bandwidth, QoS information of TCP flows provided by the transport layer should be translated into QoS information of IP traffic at the network layer using some mapping rules. Mapping rules can be used also to convert QoS information from one system to other, for example from IP to ATM so that an application can run on different systems without having to modify its QoS requirements. Mapping is generally based on the relationships between different parameters. This relation is traditionally expressed in terms of mathematical formulas. For example the loss rate at the application layer can be calculated from the loss rate at the network layer using the following formula [Huar97]:

$$L(a) = \frac{L(n) \times A(a)}{A(n)}$$

where L(a), L(n) are respectively the loss rates at the application and network layers, and A(a) and A(n) are respectively the PDU average sizes at the application and network layers.

When QoS information has been collected and processed, it needs to be stored. In large systems, QoS information is usually stored in a database. The access to the database is granted by a system manager and should be performed using queries. Basically, QoS information is often stored using a management information base (MIB) [Hafi99]. A QoSIB (*QoS information base*) can be remotely accessible by management protocols such as STMP. In the current QoS architectures, the QoSIB is usually built using an *ad-hoc* manner, meaning that information is collected then passed directly to the QoSIB with no treatment (e.g., pre-analysis). In order to access the QoSIB, distributed components should have extra knowledge about required information stored in the QoSIB, such as information definition or relative position of required information inside the QoSIB.

In summary, QoS provisioning starts with the specification activity and an intermediate component, such as QoS Controller, should be inserted into the system in order to guarantee the expected QoS level. This component is based on available QoS information about the system, which can be obtained by collection, analysis, mapping and storage operations. We next present our framework to build such a QoS information management system and to define a QoS information base.

2.2 QDD Framework

As stated in [Nguy04], QoS information plays a decisive role in the QoS provisioning process. Due to the evolution of distributed applications, the amount of QoS information increases, leading to the need for management approaches. The diversity and heterogeneity of QoS information are the main challenges for QoS information management. In addition, existing QoS architectures focus principally on network and performance parameters, which appear sometimes no longer adequate for new requirements from users and providers. In [Kerh06], we discussed that QoS should be considered from a broader perspective so that the contributions of all system components can be taken into account in order to provide QoS. For example, instead of using resource allocation methods (i.e., upgrading network connection bandwidth) as current providers are doing, we can deploy alternative solutions such as data prioritization, modality transformation, data transcoding, purpose classification or resource adaptation in order to efficiently deal with the QoS problem. These solutions require full information about clients and servers. Some intermediate layers should also be implemented on clients and servers.

We believe therefore that a QoS information manager is essential for QoS provisioning in the context of distributed multimedia applications, which is unfortunately not yet defined in current QoS architectures. Such a QoS information manager will be responsible for offering QoS information management services, such as store, access, share, transfer or produce, to all components of the system. In order to design and implement the QoS information manager, we position our work in the context of the Quality-Driven Delivery approach, proposed by Kerhervé et al. [Gerb03, Kerh06].

The QDD framework includes QoS information modeling and transformation. QoS information models are used during different QoS activities for translating requirements to system constraints, for exchanging QoS information, for checking compatibility between QoS information and more generally for making QoS decisions. Transformation is used for supporting some QoS activities such as QoS mapping. The following activities are involved in the QDD framework:

- *Modeling QoS information.* This aims at designing QoS information models with basic operations such as store, access, derivation, generation, etc., for system components. In the context of QoS information modeling, we are also interested in the collecting operation, used to obtain QoS information from system components and to analyze and synthesize such information.
- *Mapping QoS information.* This aims at representing different types of mappings used in a distributed system. One of most important tasks involved in QoS information mapping research is building QoS mapping rules, which consists in proposing a flexible mechanism considering possible QoS mappings between components or layers in a system.

- *Making QoS decisions*. This aims at selecting mapping schemes allowing the system components to provide the QoS in an optimal way in terms of user requirements and system capabilities.

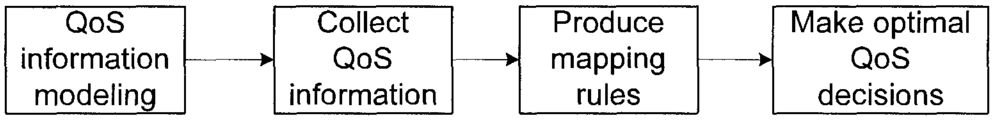


Figure 2-3: QDD Steps

To illustrate the principles of QDD, we take the example of a video delivery service where users specify their quality preferences according to three dimensions: the language of the audio sequence, the size and the frame rate of the video. Figure 2-4 presents a simplified view of this service. We identify three main modules: the quality information manager (QIM), the decision engine (DE) and the adaptation and delivery engine (ADE).

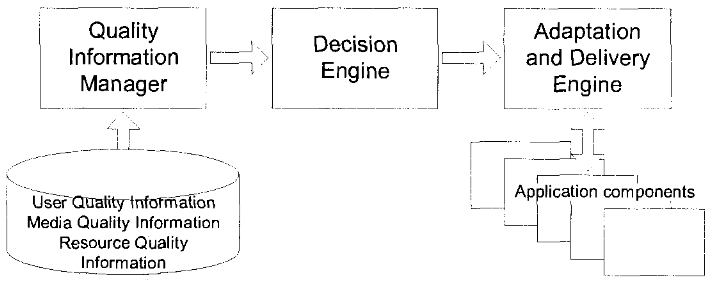

Quality Information Manager
Decision Engine
Adaptation and Delivery Engine
User Quality Information
Media Quality Information
Resource Quality Information
Application components

Figure 2-4: Components of a Video Delivery System

The QIM is in charge of collecting, storing, integrating and providing access to the quality information (QI) used by the decision engine. This information can be classified into three different categories: user QI describing the preferences and requirements of the user in terms of quality levels; media QI describing the characteristics of the video sequences and finally resource QI describing the characteristics of the resources, such as user equipment, servers or network connections.

The decision engine, located at the center of the system, is in charge of making QoS decisions allowing the video delivery under the constraints specified by the users and/or concerning the available resources. Such decisions can be made in a centralized [Nahr01] or distributed [Lima04] manner and may lead to content adaptation, resource allocation or resource adaptation.

The ADE is responsible for executing the plan produced by the decision engine. The ADE interacts with the different components of the system (encoder, video server, network) to finally deliver the video sequence to the user.

The above example illustrates the QoS provisioning based on a QoS information management system. We next describe the steps we propose to build such a system.

2.3 Basic Notions

The central point of the QDD approach is the QoS information models. QoS information models are built based on conceptual notions. Most important notions include: service, user, actor, dimension, category and value.

Service, user and actor

QDD services are offered to users and supported by different *actors*. A *service* is offered to several *users* and a user can access several services. In the description of a service, we are interested in the actors involved in the delivery process, but more specifically those influencing the quality level of the provided service. We thus focus on the elements allowing the description of the quality level offered by the actors supporting a service. Actors can be components of the distributed system such as the communication network, the video server or the database server, but also the objects which are delivered such as video objects, multimedia documents, or web pages. In the example of our video delivery service, we could make a distinction between two different types of actors: media actors, which are video objects to be delivered, and resource actors, which are the resources used for the delivery, such as the user equipment and the network connections.

Quality dimensions

A quality information model is built with the concept of *quality dimension*. Quality dimensions are used to describe objective or subjective characteristics relative to the quality level of the different actors of a delivery service or the quality level expected by the user. Subjective characteristics refer to the quality level perceived by the user while objective characteristics refer to a measurable quality level. An example of a quantitative dimension is *network_throughput*. This quality dimension is objective and can be measured using monitoring tools for communication networks. An example of a subjective dimension can be *response_time* with the values: (unacceptable, bad, good, excellent). This dimension is qualitative since the possible values depend on the perception or the interpretation of the user.

A quality dimension takes its values in a definition domain. These values are used to build expressions associated with dimensions. Expressions can be the declaration of a value: for example *network_throughput* = 1MB or the declaration of a constraint such as 2ms ≤ *response_time* ≤ 5ms.

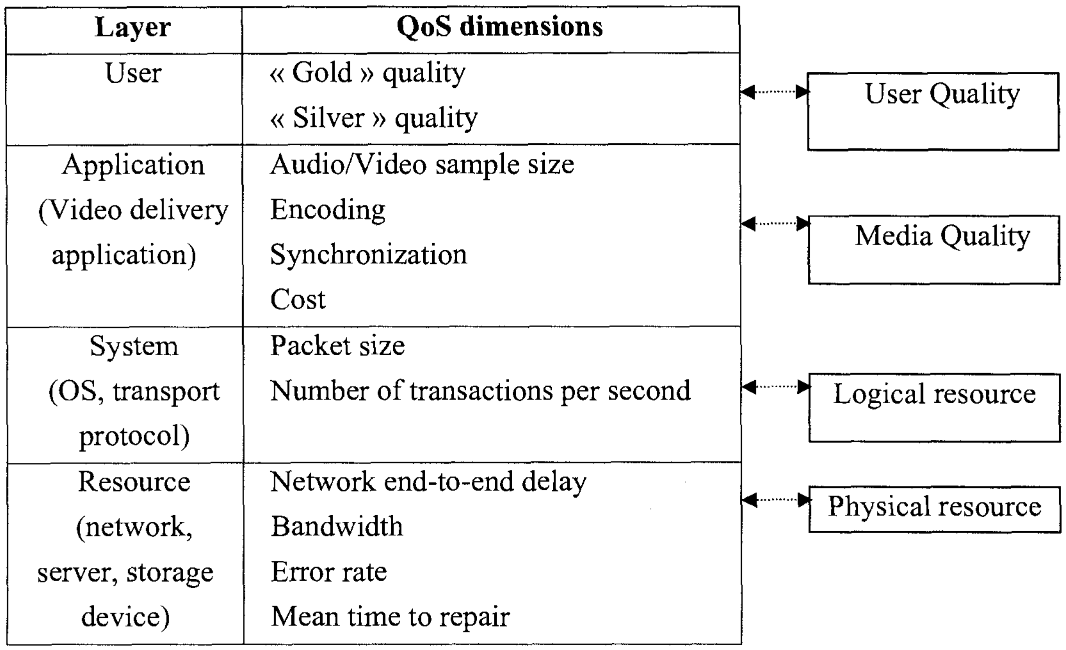
The basic notions described above are used to build QoS information models as presented next.

2.4 Modeling QoS Information

Our QoS information modeling approach is based on the QoS conceptual notions such as dimension, category and value. Starting from these concepts, we are able to build the models allowing description of QoS information. A QoS information model is defined in an open way, meaning that extension and reuse are enabled in the design and deployment. In the QDD approach, QoS information is modeled by two basic models: user model and actor model.

- A user model contains qualitative and quantitative information describing user requirements,
- An actor model integrates the quality dimensions along which is described a quality level for a given system component and can be related to resources or data.

One of the important contents of our research consists in designing the QoS information models. The approach we propose is based on a classification of QoS information where QoS information is divided into categories: qualitative and quantitative. System components are also grouped together into classes so that their QoS information can be classified in an appropriate way. For example, a possible proposal classifies the system resources into two groups: logical and physical. The logical resources include the management resources (e.g., OS, file management system, database management) and the transmission protocols (e.g., TCP/IP). The physical resources include storage devices (e.g., hard disk, memory), communication equipment (e.g., router, input/output interfaces), processing resources (e.g., processor), and perception resources (e.g., camera, microphone) (Figure 2-5).



Layer	QoS dimensions
User	« Gold » quality « Silver » quality
Application (Video delivery application)	Audio/Video sample size Encoding Synchronization Cost
System (OS, transport protocol)	Packet size Number of transactions per second
Resource (network, server, storage device)	Network end-to-end delay Bandwidth Error rate Mean time to repair

Figure 2-5: QoS Information and the Corresponding Models

Based on the conceptual notions of QoS information, different QoS information models can be built. QI models are composed of model elements, each of them describing a quality dimension. We have defined three QI models which can be used to represent a general service: User Quality Model, Actor Quality Model or Core Model [Kerh06]. Figure 2-6 presents the class hierarchy for these QI models.

The model elements of a User Quality Model describe the dimensions used to specify the expected quality level. We make a distinction between Qualitative Quality Model where the dimensions included in the model are qualitative dimensions, and Quantitative Quality Model where the dimensions are quantitative dimensions.

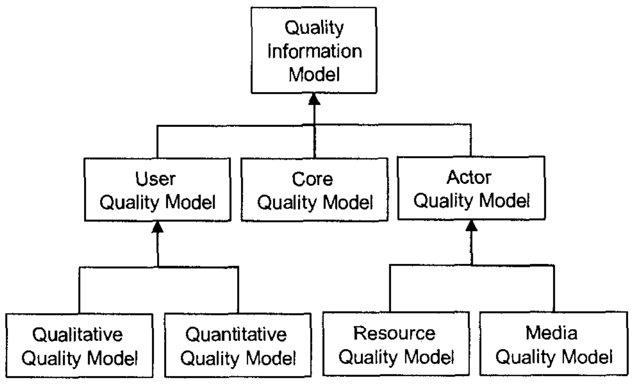


Figure 2-6: Quality Information Models

The model elements of an Actor Quality Model integrate the quality dimensions along which is described a quality level. We make a distinction between Media Quality Model built with the dimensions used to describe the quality level of an object to be delivered, and Resource Quality Model describing the quality level offered by a system component (communication network, database system, video server, user's device, etc.).

The Core Model is unique and contains the predefined set of generic categories and dimensions relevant for all types of QDD services. It can be built on the basis of existing standards. Each of the following QI models: Qualitative Quality, Quantitative Quality, Resource Quality, Media Quality, is associated to one and only one service.

Model management is the kernel of the QDD framework. Some model operations have been defined for different steps of the QDD process, such as derivation, instantiation and mapping [Gerb03]. The derivation operation is used to build QI models from other already existing models. The instantiation of a QI model produces quality information. It creates a container for QI and expressions (values or constraints) on the dimensions included in the QI model. The mapping operation transforms different QI models and deals with the heterogeneity of QI between different components of the system. In this research project, we focus particularly on mapping operations used to transform qualitative or quantitative user QI models into quantitative actor QI models.

Based on the QoS information models presented in this section, a QoS information management system can be built. We next describe the steps to generate QoS information from these models and the structure of a QoS information management base.

2.5 Collecting QoS Information

In the context of QoS information modeling, we conducted research about QoS information collection, so called QoS information monitoring, which is required for generating instances from QoS information models. Monitoring QoS information is also one of the most important QoS activities, used for QoS negotiation. We are interested in both user information and system information. Current QoS-enabled applications usually provide graphical interfaces allowing users to declare their desired quality level and the acceptable cost [Jin04]. On the other hand, QoS information about the runtime environment is collected by monitoring tools, which can be classified into three categories [Agar03]:

- The hardware tools integrated into equipment or devices (i.e., router, switch, bridge). These tools can measure particularly the QoS of the transmission medium or environment, or performance, etc.
- The software tools integrated into applications. These tools can be used to measure the end-to-end QoS at the application layer.
- The monitoring tools independent from hardware devices or applications. They are developed independently from devices and applications. Using specific

protocols, such as ICMP, RTCP, SNMP, etc., they can measure some particular QoS parameters of the systems.

Monitoring tools usually record statistical data about the working state of components into *logfiles*. These files, whose size can vary from some megabytes to some gigabytes, contain huge volumes of data values [Casa05]. Examples of statistical data can be:

- Real-time statistical generation of how content is consumed,
- Ability to analyze data by time of day,
- How many times content was requested,
- Total number of megabytes transferred,
- Costs incurred,
- Quality of transfers,
- etc.

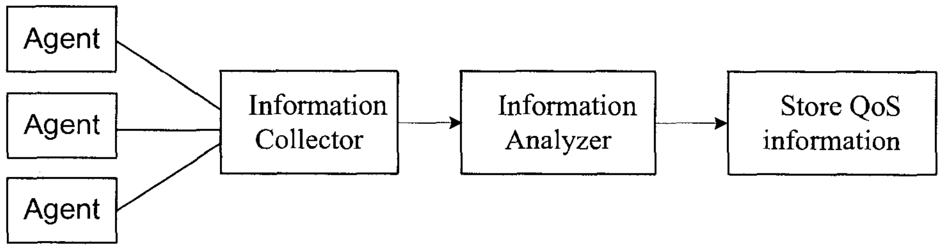


Figure 2-7: Collect QoS Information

Monitoring tools are often working independently and do not support the aggregation of information in order to build the relationships among different parameters. Few monitoring tools are accompanied by an analyzing module that is able to discard out-of-order information. Indeed, most of them return only the raw data and the user must extract the appropriate information according to his requirements. Data mining techniques can therefore be used to extract QoS information or relationships between

QoS information. Figure 2-7 shows our implementation architecture of a QoS mapping builder we proposed [Nguy06] taking as input statistical data from monitoring tools called agents. The *Collector* is used for storing statistical data coming from different sources into a database and the *Analyzer* refines the data set.

Category	Dimension
Performance	Throughput
	Response time
	Elapsed time
	Number of concurrent programs
	Number of concurrent users
Utilization	UCT utilization
	Memory utilization
	Disk utilization
	Buffer pool utilization

Table 2-1: Sample QoS Information of a DBMS

In this architecture, the agents are implemented in all system components that contribute to the service provision. Basically, these components are geographically distributed, therefore the *Collector* has to communicate with the agents using a specific protocol, such as SNMP [Harr99]. Each agent has a QoSIB containing the QoS information of the attached component. For example, the QoSIB for a database management system (DBMS) includes the categories and dimensions as described in Table 2-1.

Collecting QoS information is required for building instances of the QoS information models; that is, the association of collected values with the dimensions defined in the models. In practice, QoS information for a system component can be collected alternatively by different methods or tools. For example, the bandwidth provision of a router can be reported by a built-in hardware tool that counts the number of packets processed by router interfaces, or by a software tool measuring the input and output at the upstream and downstream links. The current collecting methods can be classified based on the system layers. For example, at the application layer, QoS information is usually collected through the user graphical interfaces (GUI); while at the resource layers, QoS information is gathered by monitoring tools without graphical interfaces [Agar03]. Another classification approach is based on the number of participants of the collecting process. For example the end-to-end delay should be measured based on the cooperation of the sender and the receiver, while the QoS parameters related to the media encoding are usually defined by only one actor (e.g., video provider). We recognized that the QoS information collection methods are diversified. Thus, in order to generate QoS information dynamically, the association of a given QoS parameter with specific collection methods should be defined in the QoSIB. The QoSIB we propose is therefore composed of four components:

- Service type, which can be broadcast or interactive.
- Service component, which provides the service (e.g., network).
- Monitoring tools, which we can use to collect QoS information of the component
- ID, which is the identification of the QoS information base provided by the collecting tool.

Table 2-2 shows an example of a QoSIB for the communication network, where three QoS categories (Performance, Availability and Reliability) are considered. Each category contains a set of dimensions for which we need to collect QoS values. For each dimension, some monitoring can be used alternatively. The collection operation may be performed solely by one of the actors participating in the service (i.e., provider or client) or both, depending on the method implemented for the monitoring tool.

Category	Dimension	Colletion tools (Agents)	Actors
Performance	Bandwidth	Router commands (hardware)	Provider
		Netperf (software)	Client
		TReno (software)	Provider and Client
		Netest (software)	Client
	End-to-End Delay	Pathload (software)	Provider and Client
		TReno (software)	Provider and Client
	Jitter	Agilent (hardware)	Provider
		Jitterlyzer (hardware)	Provider
		Iperf (software)	Provider and Client
Availability	Loss probability	Router commands (hardware)	Provider
		Iperf (software)	Provider and Client
		Ping (software)	Client
	Reachability	Traceroute (software)	Client
		Ping (software)	Client
Reliability	Mean Time To Failure (MTTF)	Router commands (hardware)	Provider
	Mean Time To Repair (MTTR)	Router commands (hardware)	Provider

Table 2-2: Sample QoSIB for Network Component

Once the QoS information base has been generated from QoS information models using the collecting tools, we can define the mappings in order to translate QoS information among different components, which is presented next.

2.6 Mapping QoS Information

From the state of the art presented in Chapter 1, we noted that information management plays a decisive role in the QoS provisioning process. Some research has been dealing with the QoS management problem in general, without taking into account the QoS information management as a specific issue. The diversity of QoS information, as well as the evolution of distributed systems, have led to the fact that the current QoS approaches, where QoS information is managed in an ad-hoc manner, may become no longer compatible with new system requirements. We believe therefore that there is a need for abstraction of QoS information and the modeling and mapping can be used to deal with the QoS information management. One of challenges for the current QoS mapping activity is the limited number of mapping rules generally based on mathematical formula, which can not satisfy the increasing amount of QoS information and the variety of system components.

QoS information collected by specific tools provided with each system component and user specification can have variable formats needing to be unified. For example, QoS specification at the user layer often contains very abstract information such as "good", "bad", or "DVD quality", "telephone quality", etc. Lower layers, namely service and system and resource layers, state their offers using more technical details such as frame rate, number of color bits, or available bandwidth, packet rate, or CPU throughput, memory buffer capacity. Therefore in order to help the QoS manager to provide the QoS corresponding to user specifications, QoS information mapping is required. It aims at translating QoS information between different layers. In addition, there can be several components running simultaneously in the same layer and providing similar services (i.e., two servers executing an application). In such a case, QoS mapping is also needed to compare the QoS provided by different components in order to make optimal QoS decisions (i.e., in terms of cost).

Our research for QoS information mapping aims at generating the mapping rules from available QoS information contained in the QoS information base. The process is shown in Figure 2-8, and includes:

- Classification of QoS mappings based on the system architecture identification (e.g., layer-based or component-based). QoS mappings of a given system can therefore be represented in a systematic way.
- Obtaining and pre-processing QoS information from available sources.
- Generating QoS mapping rules in a flexible way, enables consideration of all possible mappings when a QoS decision is made. Optimal QoS decisions can be made only if all possible mappings are taken into account.

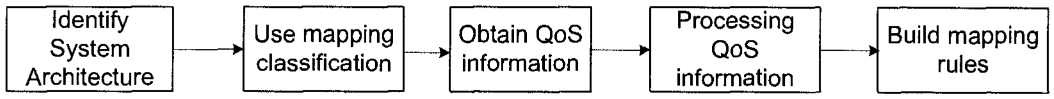

Identify System Architecture
Use mapping classification
Obtain QoS information
Processing QoS information
Build mapping rules

Figure 2-8: QoS Mapping Rule Building Process

2.6.1 Classification of QoS Mappings

Traditionally, a service is usually implemented using the vertical layer approach where the QoS specified or offered by each layer can be expressed by different abstract levels. QoS mapping activity is used to "perform the function of automatic translation between representations of QoS at different system levels and thus relieves the user of the necessity of thinking in terms of lower level specification" [Aurr98, Nahr00]. For example, QoS specification at the application layer often includes information at very abstract levels such as "good", "bad", "acceptable", etc. QoS offers at lower layers, such as transport, network and link layers, are described using more and more technical details such as available bandwidth, packet rate, or CPU throughput, memory buffer capacity. Thus, QoS mapping should be provided to describe the relationships between information from different layers. It is essentially qualitative, aimed at translating the different representations of QoS information.

On the other hand, QoS mapping is also required among components of the same system layer, particularly when we consider the middleware and object-oriented systems [Schm02]. QoS information of these components has consequently the same content. In such case, the qualitative mapping is not needed. For example, data transmission service can be provided by two components running respectively on IP and ATM networks. The first component describes the network throughput in terms of bits per second, while the second expresses it using ATM packets per second. QoS mapping in this case is used for unifying the scales, and is rather quantitative.

In general, QoS mapping is deployed to translate: i) the content, or ii) the format, or iii) both content and format of different information. QoS mapping is system dependent and application dependent. For example, the QoS mapping used in a layer-based system should translate QoS information between different layers (e.g., between the transport layer and the network layer), while in a component-based system the QoS mapping can be used to compare the contribution of two similar components (e.g., two servers doing the same task).

Since most current distributed systems usually combine the layer-based and the component-based architectures, the existing research that deals exclusively with layer-based mapping [Jin04] or component-based mapping [Schm02] can be insufficient. The approach we propose for QoS mapping is based on the models where QoS mappings are classified into two main categories [Kerh06] as follows.

- Vertical mapping: when Source and Destination models are located on different layers. The vertical mapping is used to translate information content. For example, it is required to translate a TCP-based dimension to an IP-based dimension.
- Horizontal mapping: when Source and Destination models are located on a same layer. The horizontal mapping is used to unify the scales of information. For example, it is required to translate ATM packet rate to IP packet rate.

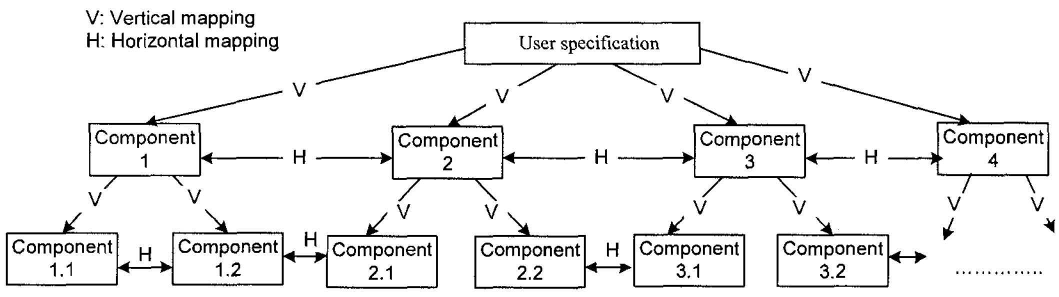


Figure 2-9: Available Mapping Schemes in a Distributed System

In the context of distributed multimedia systems with heterogeneous components, vertical and horizontal mappings should be used in a flexible manner in order to set up QoS negotiations. Since some distributed components can be used alternatively, different mapping schemes can be deployed to express a QoS contract (Figure 2-9). Thus, the QoS manager should be responsible for selecting the optimal scheme considering the system constraints. For example, better video image quality can be provided alternatively by: i) changing the video codec, or ii) increasing the network bandwidth, where in most cases, the first method is cheaper [Nguy05].

We next present the steps to build QoS mapping rules from available QoS information.

2.6.2 Building QoS Mapping Rules

One of the essential conditions for having an optimal mapping scheme is that all possible mappings should be taken into account. This leads to the need for an advanced mechanism for examining mapping rules among all the system components. Along with the vertical architecture, the traditional mapping rules are presented by mathematical formulas. For example [Huar97, Koli02]:

- Packet rate (consequently packet delay) at the transport layer: $R_N = (\lceil M_A / M_N \rceil) \times R_A$, where M_A, M_N are respectively the media sample size and rate obtained from, and R_A is the packet size at the transport layer.
- Inter-arrival time (jitter) at the transport layer: $P_N = (1/R_A)/\lceil M_A / M_N \rceil$
- Packet loss rate at the network layer: $L_N = (L_A \times A_N)/A_a$ where A_a and A_N are respectively PDU average sizes of video and network, L_A and L_N are respectively loss rates at user and network layers.

Unfortunately, the number of formula-based mapping is generally not sufficient for describing all the relationships existing among the specific components of a distributed system. For example, it is very hard (or would be impossible) to express the relationship between video buffering time, video codec, network bandwidth, server processing power and server memory capacity by a mathematical formula.

This issue led us to propose an approach for implementing a QoS mapping builder that is responsible for generating mapping rules from QoS information coming from different sources [Nguy06]. The working principle of the QoS mapping builder consists of mining the statistical data and configuration files containing working states and configurations of each component of the system in order to produce association rules describing the QoS relationships between them. The two main techniques used for this task are: i) classification and prediction of user QoS requirement based on client-side configuration, and ii) clustering system runtime information. Mapping rules are generated under the table-based format (i.e., Table 2-3), which enables the flexibility of the number of input and/or output parameters and the components involved in a mapping rule.

MPEG-2 MOS	Bandwidth (Mbps)
4.0	5.62
4.1	6.00
4.2	6.47
4.3	7.07
4.4	7.88
4.5	8.99
4.6	10.65
4.7	13.37
4.8	18.64
4.9	33.18

Table 2-3: Mapping from MOS (User Model) to Bandwidth (Network Model)

The approach for generating QoS mapping rules we propose includes the following steps:

- *Collect QoS information for users and system.* This information is statistical data.
- *Classify user requirements.* The classification is based on the prediction of user's maximal level capability. A decision tree is built for the classification as shown in Figure 2-10. The classification of user requirements can be performed dynamically at two moments: starting of service or during the service whenever the user configuration is changed. The first moment is used to set up an SLA

(*Service Level Agreement*) between user and provider and the latter served for realizing a transparent adaptation or renegotiation.

- *Cluster system statistical information*. This aims at determining the number of QoS levels for a service from the QoS statistical data corresponding to the number of user classes. Note that the system information we mention here is not limited within the hardware devices but can cover also the software components and the data themself (i.e., video transmission codec). Clustering is a set of methodologies for automatic classification of samples into a number of groups using a measure of association, so that the samples in one group are similar and samples belonging to different groups are not similar. An example of clustering is shown in Figure 2-11 , where:

  Cluster 1 = low level bandwidth

  Cluster 2 = medium level bandwidth

  Cluser 3 = high level bandwidth

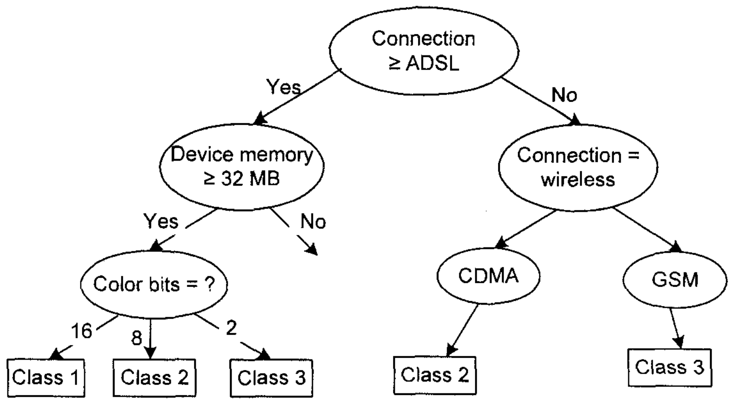
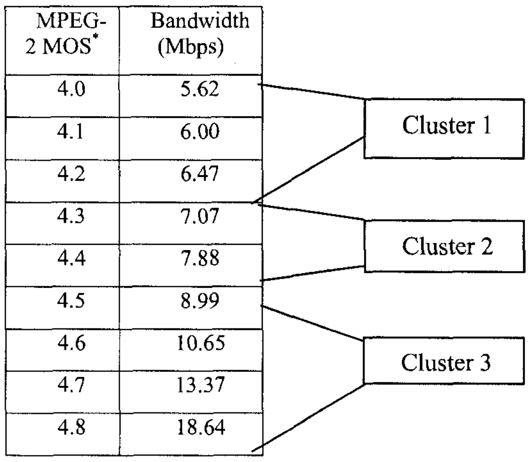


Figure 2-10: Classification of User Requirements

Our experimentation used the K-Means Partitional Clustering algorithm whose performance has been discussed in [Liu04] in order to classify the statistical data.

The clustering computation is repeated until the *centroid* and the square error for all the clusters are stabilized.

- *Define QoS mapping rule.* A mapping rule is an association of user requirements and system offers. It is represented by a table as in Figure 2-12 where a mapping between user classes and system QoS levels is built. Users belonging to the gold class can be served with the gold QoS level, those of the silver class served with the silver QoS level and so on. This mapping is defined based on a set of user parameters and system parameters, which we suppose the most important for our experimental service, such as user connection speed, processing power, storage capability, bandwidth, video data MOS, etc. Mapping between more or fewer parameters is also available. In general, we have three patterns for creating a mapping model:



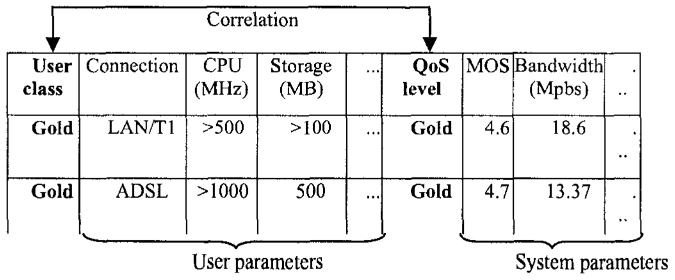
MPEG-2 MOS[*]	Bandwidth (Mbps)
4.0	5.62
4.1	6.00
4.2	6.47
4.3	7.07
4.4	7.88
4.5	8.99
4.6	10.65
4.7	13.37
4.8	18.64

Figure 2-11: Clustering the System Offers

  - One to one mapping. It maps a user parameter to a system parameter,
  - One to many mapping. It maps a user parameter to several system parameters,

[*] MOS - *Mean Opinion Score*

- Many to many mapping: It maps some user parameters to some system parameters. These parameters must be computed together due to the correlation among them.



User class	Connection	CPU (MHz)	Storage (MB)	...	QoS level	MOS	Bandwidth (Mpbs)	...
Gold	LAN/T1	>500	>100	...	Gold	4.6	18.6	...
Gold	ADSL	>1000	500	...	Gold	4.7	13.37	...

Correlation
User parameters
System parameters

Figure 2-12: A QoS Mapping Rule

In summary, the approach we propose allows the building of the QoS mapping rules from available QoS information. It is flexible, because the mapping rules are generated based on the relationships among a variable number of components. A user requirement may correspond to several system configurations obtained by combining system components with different parameters. Therefore the QoS manager can flexibly decide on the QoS mechanisms to be used achieve a given user requirement. The proposed approach is also dynamic, because the changes in the system can be reflected in the mapping rule. When a component is removed from the system, or a dimension is changed, the corresponding column of the tabular mapping rule can be rebuilt based on system working state statistical information.

Together with the QoS information modeling and QoS information generation proposed in Sections 2.4 and 2.5, the QoS mapping rule building process proposed in this section allows the QoS manager to consider and to compare the contributions of all system components in order to provide QoS. We will discuss in the next chapter possible QoS decisions made with the help of the proposed QoS information management system.

2.7 Chapter Conclusions

In this chapter, we presented our work on QoS information modeling and mapping in order to support QoS provisioning. The QoS provisioning process starts with the specification activity and an intermediate component, such as QoS Controller, can be inserted into system to provide the expected QoS level. This component is based on available QoS information of the system, which can be obtained by collection, analysis, mapping and storing operations.

In current QoS architectures, QoS information is managed in an *ad-hoc* manner by integrating into source code. This leads to issues related to system extensibility and the contribution of all system components is not taken into account. In the context of the QDD framework, we proposed to build an independent QoS information management system in order to offer QoS information management services to all components of the system.

We have investigated QoS information modeling to build the proposed QoS information system. QoS information models are based on the basic notions such as dimension and category. Each model corresponds to a system component. A QoS information base is generated from QoS information models using the collection tools to maintain QoS information of all components of the system.

We then proposed an approach to build QoS mapping rules based on statistical data using data mining techniques. The objective is to have as much QoS information as possible concerning all service components of the system and to enumerate all relationships of these components in order to compare their contributions in QoS provisioning. The QoS mapping rule builder considers the different system architectures and all the involved components.

The next chapter is devoted to the QoS decisions within the QDD framework and an experimental application where possible QoS decisions are taken into account.

Chapter 3 Applications and QoS Decisions

In this chapter, we present a case study of our proposed QoS information management system, built using QoS information modeling and QoS mapping, and used for QoS provisioning. The chapter begins with a description of a video streaming application, its QoS issues and current solutions for QoS provisioning. We next present the QoS decision making process within the QDD framework, where we demonstrate that various QoS decisions may be used alternatively and that the resource allocation is not always the best decision. In order to validate the information models mapping and the advantages of our approach in making QoS decisions, we conduct some experiments with the video streaming delivery system.

3.1 Video Streaming Delivery and QoS Decisions

As discussed in Chapter 1, the increased volume of QoS information led to the emergence of a QoS information management system. In Chapter 2, we have presented the steps used to build such a system. In this section, we investigate the ability of using such a system for QoS provisioning in a video streaming system as described in Chapter 1 and we discuss the advantages of the QoS information management system in QoS decision making.

The video delivery service is chosen for the experiment because it is one of the typical distributed applications where QoS issues can be easily experienced. Basically, there are two modes for video transmission over the Internet, namely, the download mode and the streaming mode [Wu01]. In the download mode, users have to download the entire video file and then play it back on a local machine, while in the streaming mode, parts of video content are being received and decoded in real time. The download mode is not concerned by QoS problems because users play their video locally. QoS issues can be experienced with the streaming mode due to the fact that the current Internet does not support any QoS facilities guaranteeing online user's perception requirements.

The video streaming application we built consists of four main components (Figure 3-1):

- Video Provider (VP). Located at the heart of the system, it contains complete information about system architecture, topology, video descriptions and QoS information.
- Streaming Server (SS). It can be installed on any server machine within a distributed network to provide streaming video. SS does not store data locally; instead, it loads video clip files from file management systems then converts them into streams to send to clients. SS is designed as a multi-thread application, thus it is able to serve several clients and several jobs at the same time.
- Client Player (CP). It is a video player program used to interpret a number of video encoding types such as MPEG, AVI, MPG, H.263, etc., as well as real-time streams. Client player has also graphic interfaces, allowing users to contact the VP to obtain the movie list or to seek movies by keywords.
- Video Administrator (VA). It is a stand-alone program providing graphic interfaces to control system activities. It can be used also to import video clips into the database and to configure SSs and VP.

Video streams are transmitted from SS to CP using the RTP protocol (Real-time Protocol). A signaling protocol is implemented to establish connections between: i) CP and VP, ii) VP and SS, and iii) SS and CP.

Such typical video streaming systems usually place high demands on system resources, and can present some QoS issues, such as:

- Video latency. It is due to network delay (alternatively bandwidth), server delay (alternatively processing speed), storage server delay, client delay (processor speed or I/O system), or video content quality.
- Image/audio clarity. It is due to network packet loss and jitter, data noise, server processing error, or displaying error.

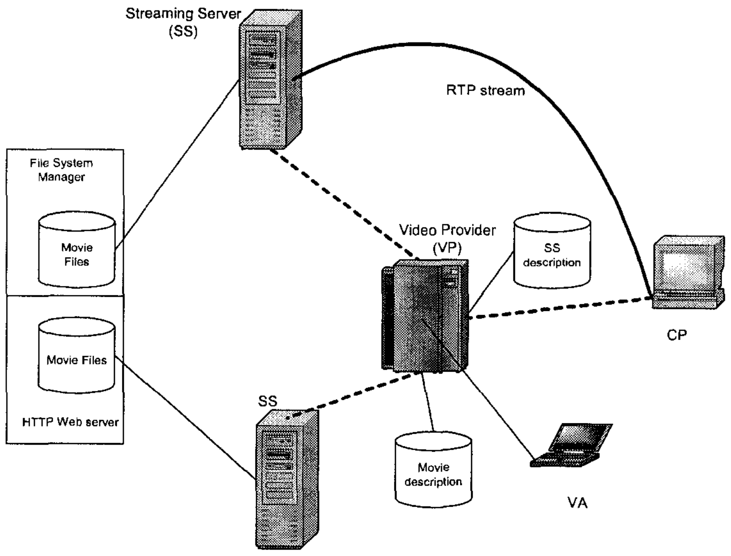


Figure 3-1: Video Streaming Delivery Application

Actually, video streaming services have been provided by different suppliers, e.g., AOL [Aol07], Vidéotron [Vid07], CBC [Cbc07], etc., through Web interfaces. The system consists usually of several streaming servers installed within the supplier's network. Dedicated lines are deployed to connect the streaming servers to the Internet. The service is basically best-effort and no specific user QoS level is guaranteed. The suppliers aim only at maintaining the service with their current network and server capacity. When the streaming quality decreases due to the overload of network or servers, users will experience video rebuffering or even that the video session is restarted.

In [Scha03], the authors proposed a CORBA-based QoS management architecture providing video streaming service. The architecture is able to configure three types of resources: processor, communication and memory resources. The QoS is provided based on priority levels. The QoS decision can be processor rescheduling, bandwidth reservation or memory allocation. This architecture, however, does not consider the video documents as resources themselves.

Some other video streaming architectures with QoS capability, such as [Wu01], focus on the video frame processing and bandwidth allocation problems. They do not take into account non-performance parameters, such as semantic requirements or service context. For example, the user's preferred language is not considered (e.g., English or English with French subtitle videos). In our proposed architecture, users may also specify QoS requirements on the video content. This can be achieved with the help of a QoS information management system which contains video language-speaking information.

In addition, using alternatively different decisions for QoS provisioning is not taken into account in current research and commercial products. The conventional QoS management mechanisms take into account only parameters related to performance, and focus particularly on network quality, such as network delay, packet loss, packet disordering and jitter. This approach leads to the fact that re-allocation is usually the strategy to be deployed when a QoS violation is detected. However, we have pointed out in [Nguy05] that re-allocation is not always the optimal way to address QoS problems in the general case. For example, we demonstrated that content optimization (e.g., data compression), resource allocation and adaptation can be deployed alternatively in order to deal with the network congestion problems.

The ultimate goal of the QDD approach is to facilitate the selection of optimal QoS decisions based on available QoS information and mapping rules. Taking the example of a video streaming application, a request for video transmission with a certain level of quality can be expressed in terms of: the processing time of the CPU, amount of the buffer memory and the available bandwidth. A mapping resulting in using less memory may lead to more processor or more bandwidth consumption, and vice versa. Thus, if users or providers put constraints on some QoS dimensions (i.e., processing cost, limited size of memory or the bandwidth threshold), using some mapping rules can lead to a better solution regarding the resource constraints. In general, in a distributed system, several components can offer similar services, e.g., two video servers providing a same video sequence. QoS provisioning should therefore deal with the optimization problems, with respect to user or provider expectations.

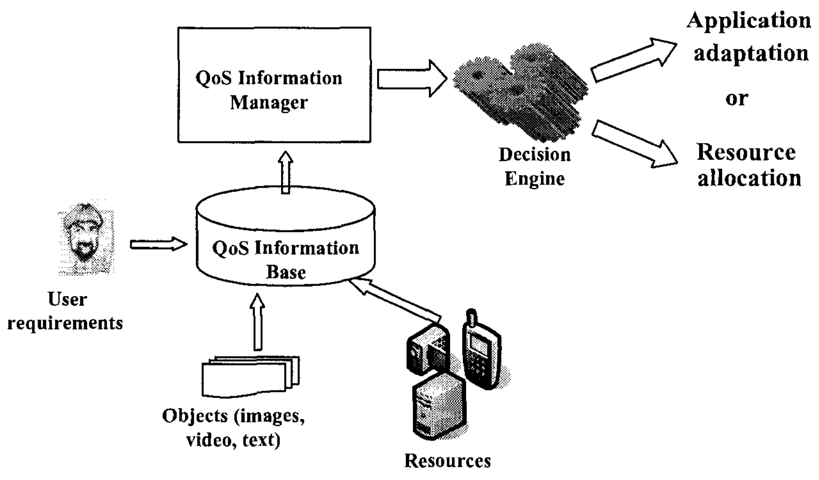


Figure 3-2: Making QoS Decisions in the QDD Architecture

Figure 3-2 shows the QoS decision making process within the QDD architecture, which is implemented in the VP component. QoS information coming from various sources of the system (e.g., users, documents, physical devices, etc.) is stored in a QoS information base. The QoS information manager provides the requested components with appropriate QoS information and QoS mapping rules. The decision engine computes the optimal QoS provisioning strategies according to user requirements and minimizes the overall cost of service provisioning.

As traditional QoS approaches concentrate principally on the communication network [Schm99], the preferred decisions addressing the QoS violation problem are usually related to the network configuration (i.e., bandwidth re-allocation or server re-configuration). With our QDD perspective, we take into account the QoS information of the overall system in order to make optimal decisions. For example, with the video streaming system, we can have the following situations [Nguy05].

- When delay increases, leading to lower video transmission rate, possible QoS decisions are:

- Allocating more bandwidth on the existing path. This costly decision is usually chosen in a provider-oriented QoS architecture;
- Changing the current transmitting server (or changing path). This decision requires further QoS information about streaming servers in the system, but that may be useful if the violation comes locally and uniquely from the current server;
- Changing transmitting codec. This decision requires that QoS information about video codecs should be taken into account, but the overhead may be smaller.

- When jitter increases, leading to poorer video smooth layout, possible QoS decisions are:
  - Increasing temporary buffers of the transmitting servers or on-path network equipment. This decision is often costly;
  - Increasing receiving buffers of clients. This decision is quite simple but needs further QoS information on the client side.
- When packet loss rate increases, leading to poorer video image/audio quality, possible QoS decisions are:
  - Changing to a more reliable transmission protocol with advanced error solving mechanisms, such as forward error control, retransmission, error concealment or group of picture based error spreading [Pyun02]. This decision poses severe impact on both the client and server;
  - Changing transmitting codecs (e.g., some codecs are more sensitive to loss than others [Robu03]). This decision is simpler, but it requires QoS

information about video codecs and computation algorithms should be implemented.

3.2 Examples of QoS Information Models and QoS Mapping

In the proposed QoS provisioning architecture for video streaming delivery, QoS information is collected by user graphical interfaces and system monitoring/configuration tools. Figure 3-3 illustrates the QoS information models of such a video streaming delivery system. Two largest models are derived from the *Core* model: *User* model and *Actor* model. As mentioned previously, qualitative dimensions are handled by the *User* model while quantitative dimensions are managed by the *Actor* model. The *User* model contains qualitative dimensions, expressed in terms of user MOS (Mean Opinion Score) such as image quality, audio quality and content display quality. The *Actor* model contains two sub-models: *Network* and *Streaming Server*. The *Network* model represents the communication service, where we focus on the transport layer. It includes three quantitative dimensions: delay, jitter and packet loss. The *Streaming Server* model includes four sub-models: *Encoding*, *Storage*, and *Operating System*. The *Encoding* model describes the quality of video documents handled by the streaming server. It contains four quantitative dimensions: frame rate, frame size, number of color bits and number of audio channels. The *Storage* model corresponds to the file storage system. It contains three quantitative dimensions: memory utilization, caching and error rate. The *Operating System* model represents the OS platform where the streaming server is implemented. It contains three quantitative dimensions: CPU utilization, number of threads supported and file control properties.

Mapping rules are built and defined on the QoS information base. A qualitative dimension of the *User* model can be mapped to different quantitative dimensions of *Network*, *Encoding*, *Storage* and *Operating System* models so that the system can make the best possible QoS decision given the resource constraints.

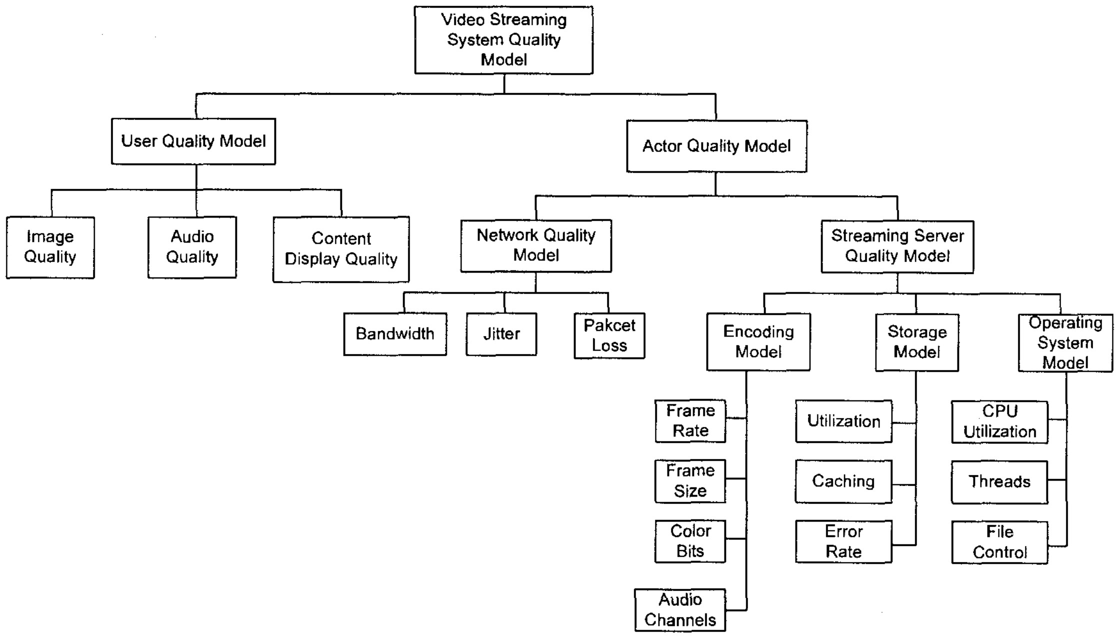


Figure 3-3: Quality Information Models for Video Streaming Delivery System

Let us consider for example a mapping from content display quality, which is specified by users in term of the capability of displaying the whole video content continuously and smoothly, to network and streaming server parameters. In order to maintain the content display quality level, networking resource allocation is usually deployed in the current video streaming architectures. Using the QoS information management system we proposed, other decisions can be considered. In our example, the bandwidth dimension of the *Network* model is defined within a domain value consisting of 5 values: 30kbps, 60kbps, 90kbps, 120kbps and 150kbps. The number of color bits dimension of the *Encoding* model has 3 possible values: 24, 16 and 8. The content display quality dimension of the *User* model has 3 possible values: bad (smooth display time is less than 25%), acceptable (about 25-50% smooth display time), good (50-75% smooth display time) and excellent (75-100% smooth display time). A table-based mapping rule from *User* model's content display dimension to *Network* model's bandwidth and *Encoding* model's number of color bits dimensions is defined as in Table 3-1. The rule is built as follows:

Content display quality (User model)	Bandwidth (kbps) (Network model)	Number of color bits (Encoding model)
Excellent	150	24
Excellent	150	16
Excellent	150	8
Excellent	120	16
Excellent	120	8
Good	120	24
Good	90	8
Acceptable	90	16
Acceptable	60	8
Bad	90	24
Bad	60	16
Bad	30	8

Table 3-1: Mapping from Content Display Quality to Network and Encoding Dimensions

- Users specify their QoS specification for video display smooth quality based on Mean Score Opinion (MOS) method.
- The QoS information management system determines the configurations that can satisfy each video display smooth QoS level considering the network dimensions and the encoding dimensions which we can change.
- A table-based mapping is built combining the video display smooth QoS levels and system configurations using two system dimensions: bandwidth and number

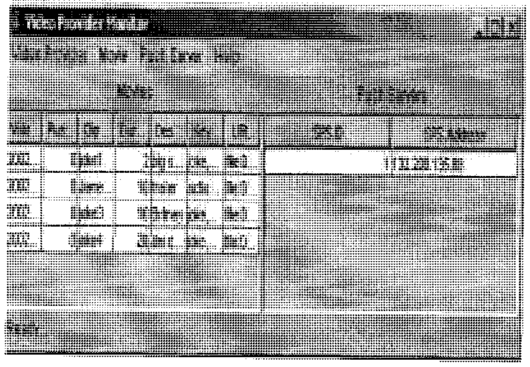
of color bits. The percentage of smooth layout frames over the total number of displayed frames is counted and the relationship between the system configuration and QoS specification is established.

In our experiment, the most important user requirement is to understand the whole video content so the video display smoothness quality should be considered. The video display smoothness quality can be improved in case of network congestion by increasing bandwidth or by reducing the number of color bits. In the traditional approach where only network-related dimensions are taken into account, the solution for improving video display smooth is bandwidth allocation. It is costly. Regarding the cost of additional network resources, we observe that reducing the image quality in terms of the number of color bits is a more cost-effective solution.

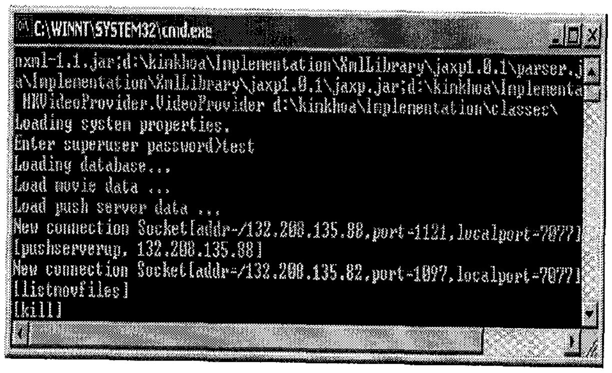
3.3 Running Screenshots

Our experimental application is implemented using Java, and we use *Java Media Framework* (JMF) API for image and audio processing [Sun07]. Figure 3-4 shows some running screenshots of the Video Provider and Video Administrator. When the Video Provider is launched, it loads the movie database, the list of registered streaming servers and the QoS information base of the system. This information can be changed by users through Video Administrator interfaces. Figure 3-4 (c) shows content description of a clip in the movie database and its QoS information is given in Figure 3-4 (d). QoS information of a clip includes video quality and audio quality dimensions which are determined by analyzing video codecs.

A signaling protocol has also been designed based on text commands in order to exchange control messages among Video Provider, Video Administrator, Streaming Server and Client Player. Based on this protocol, the Video Administrator gets access to the database managed by the Video Provider. There can be several streaming servers in the system. Each Streaming Server registers with the Video Provider through the signaling protocol. When a Streaming Server is launched, the Video Provider checks whether it is allowed to stream the movie data. If it is, the Video Provider will connect it to requested clients.

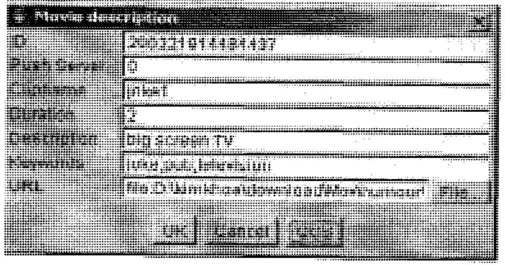


(a) Video Administrator



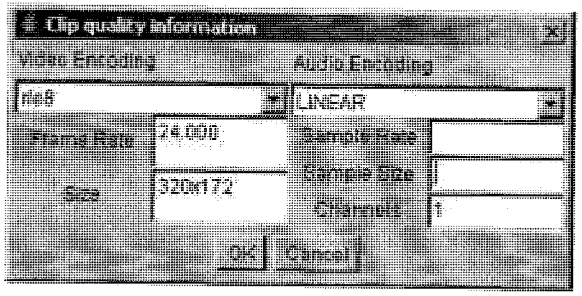
```
C:\WINNT\SYSTEM32\cmd.exe
nxml-1.1.jar;d:\kinkhoa\Implementation\XmlLibrary\jaxp1.0.1\parser.j
a\Implementation\XmlLibrary\jaxp1.0.1\jaxp.jar;d:\kinkhoa\Implementa
 HKVideoProvider.VideoProvider d:\kinkhoa\Implementation\classes\
Loading system properties.
Enter superuser password>test
Loading database...
Load movie data ...
Load push server data ...
New connection Socket[addr=/132.208.135.88,port=1121,localport=7077]
[pushserverup, 132.208.135.88]
New connection Socket[addr=/132.208.135.82,port=1097,localport=7077]
[listmovfiles]
[kill]
```

(b) Video Provider



ID	200321914191437
Push Server	0
Clipname	inkat
Duration	2
Description	big screen TV
Keywords	[illegible]
URL	[illegible]

(c) Movie description

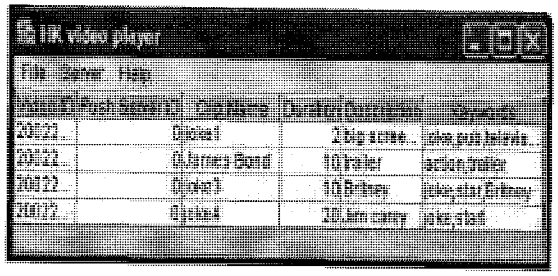


Video Encoding	h263
Frame Rate	24.000
Size	320x172
Audio Encoding	LINEAR
Sample Rate	
Sample Size	
Channels	1

(d) Movie QoS information

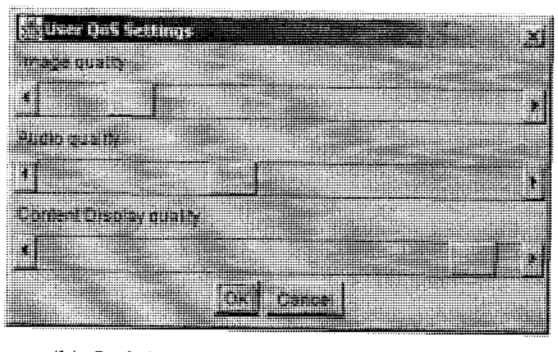
Figure 3-4: Video Administrator and Video Provider Screenshots

Figure 3-5 shows some screenshots of the Video Player. When a user runs a Video Player instance on his device, the Video Player will connect to the Video Provider using the signaling protocol. Figure 3-5 (a) shows the list of available movies in the system, which users get through a command. In this list, there is also information about streaming servers that are currently hosting movies, the duration of each movie and keywords describing movie contents, etc. With an interface given in Figure 3-5 (b), a user may specify the QoS levels he wants. In this case, his most important requirement is the continuity and smoothness of video display which allow him to understand the whole content of the clip. The user also specifies that he may accept low quality image and audio. When the user chooses playing a movie, a command is sent to the Video Provider, which then selects an appropriate Streaming Server to transmit the movie stream. The Streaming Server contacts the desired client using the signaling protocol to trigger the streaming session (Figure 3-5 (c)).

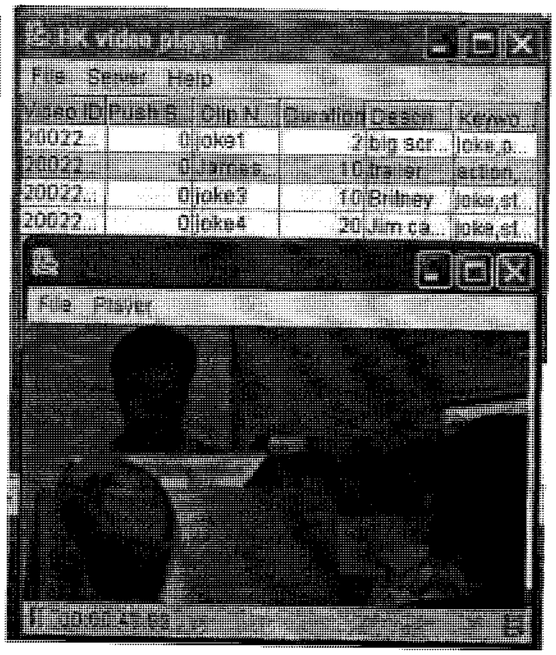


Video ID	Push Server ID	Clip Name	Duration	Description	Keywords
20022...	0	joke1	2	big scree...	joke,pus,televis...
20022...	0	James Bond	10	trailer	action,trailer
20022...	0	joke3	10	Britney	joke,star,Britney
20022...	0	joke4	20	Jim carry	joke,star

(a) Video Player main interface



(b) QoS level setting for a movie before playing



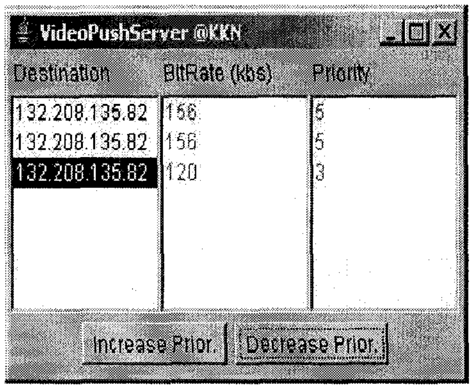
Video ID	Push S...	Clip N...	Duration	Descri...	Keywo...
20022...	0	joke1	2	big scr...	joke,p...
20022...	0	James...	10	trailer	action...
20022...	0	joke3	10	Britney	joke,st...
20022...	0	joke4	20	Jim ca...	joke,st...

(c) Video streaming by Video Player

Figure 3-5: Video Player Screenshots

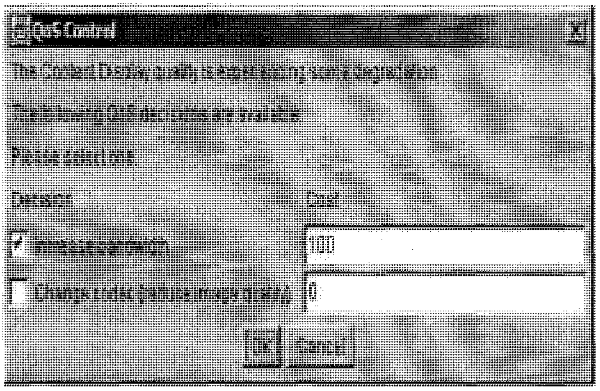
The Streaming Server interface provides the administrator with information about the current transmission session, the corresponding bitrates and the priority of the streams. In Figure 3-6 (a), there are three opened sessions for the same destination streaming three movies with different priorities. The two first streams are transmitted at 156kbps and the last one is at 120kbps. The administrator can control the bandwidth of a given stream by changing its priority. There are five priority levels in our experiments.

When there is a QoS violation, e.g., the bandwidth is decreasing or some packets are lost, the system is required to make QoS decisions based on available QoS information about service components and mapping rules. Figure 3-6 (b) gives two possible QoS decisions with associated costs. The first decision is to buy more bandwidth, for example, by increasing the priority of the current stream. The second is to change the video codec, i.e., from 24 color bit depth to 8 color bit depth. Since in this case, the user is more concerned with the video content display than the quality of image and audio, the second decision may be better considering the cost.



Destination	BitRate (kbs)	Priority
132.208.135.82	156	5
132.208.135.82	156	5
132.208.135.82	120	3

(a) Streaming Server main interface



The Content Display quality is experiencing some degradation

The following QoS decisions are available

Please select one

Decision	Cost
[✓] Increase bandwidth	100
[ ] Change codec (reduce image quality)	0

(b) QoS Control

Figure 3-6: Streaming Server Screenshots

Experiments have been conducted in the context of a QoS controllable environment, where a *perturbator* is implemented for controlling some QoS parameters of the system.

At the first stage, we take into account only the network related parameters, namely delay, and packet loss, and video document related parameters, such as encoding and color depth. Server, database and client related parameters will be added in the future implementations. Client behavior in terms of video frame rate, image and audio quality is targeted to be observed. The *perturbator* captures data packets on the stream between client and server. Each packet is then processed according to the quality level it is assigned. The advantages of such a *perturbator* in comparison with a conventional traffic generator include:

- Priority levels can be changed immediately as users require,
- Effect on the client and/or server side can be observed immediately,
- Different parameters can be treated together or separately as necessary,
- The *perturbator* can be removed from the system with no impact on the current data stream.

We have also considered a set of possible mappings in the experimental system, including:

- Mappings from users' QoS requirements to network dimensions, e.g., from video quality to network delay and packet loss.
- Mappings from users' QoS requirements to streaming server dimensions, e.g., from video quality to the video color depth and frame size
- Mappings from streaming server dimensions to network dimensions, e.g., from video color depth and frame size to network delay and packet loss.

Each mapping rule is associated with a cost, defined by the system administrator. Whenever an environmental parameter is changed (e.g., network delay increases due to the change of the stream priority), the system automatically computes the optimal QoS decision based on available mapping rules and their costs. Therefore, such video streaming system also allows us to conduct experiments on the performance of resources, based on the comparison of their contribution to service provisioning.

3.4 Chapter Conclusions

This chapter has presented the QoS management using the QoS information management facilities provided by a QoS information management system. A video streaming delivery system is used as example to demonstrate the advantages of our approach.

We investigated different QoS decisions that can be used alternatively for QoS provisioning in a video streaming delivery system, and pointed out that the resource allocation decision is not always the optimal one. We then propose some alternative decisions which can be used to deal with QoS problems.

We described the video streaming application we implemented in order to validate the utility of the QoS information management system. Such system is used to provide requested components with appropriate QoS information. The ultimate goal is to make best possible QoS decisions considering user specifications and resource deployment costs. Through an example of the video display smoothness dimension, we demonstrated that QoS can be provided efficiently using QoS information management services and a specific table-based mapping rule. In this example, the cost of resource utilization for our QoS decision is lower compared to the traditional network bandwidth allocation decision.

We also provided details of implementation of our experimental video streaming application and then suggested further experiments to be conducted. Such a QoS information management approach may be used effectively for QoS management in other systems, or specific devices, for example, the core routers that we will present in the second part of the thesis. A core router is required to support several protocols and data flows. Traffic from different protocols, such as IP and MPLS, can be treated with different priorities so there is a need for QoS information management in order to manage user-level requirements and QoS capabilities of systems and links. QoS dimensions from a given layer or protocol must be expressed in terms of dimensions belonging to other protocols or layers in order to achieve the communication over heterogeneous networks. Therefore flexible and dynamic QoS mapping should be considered, such as the mapping from IP to MPLS traffic. Thus our proposed QoS mapping builder architecture can be used.

This chapter ends the first part of our PhD research related to the QoS provisioning at the application level. We have presented a complete QDD framework where the ultimate goal is to provide QoS according to user requirements based on QoS information of all service components of the system. Mechanisms to support the QDD have been developed with QoS information modeling and mapping, that allow us to take into account the contribution of all service components. With the QoS decision making process described in this chapter, the QDD is able to help to improve the quality and efficiency of high quality services regarding the evolution of user requirements and system characteristics.

PART II. SCALABLE AND DISTRIBUTED SOFTWARE ARCHITECTURE FOR NEXT GENERATION ROUTERS

Chapter 4 Architectures of Routers

This chapter begins the second part of the PhD thesis. In the previous part, we investigated a QoS management architecture at the application level based on the assumption that the underlying devices provide the facilities allowing the application to obtain QoS information and to modify the QoS parameters. This allows the making of efficient QoS decisions, based on information about the QoS capabilities of the devices. For example, the video streaming application with QoS-enabled features should be able to configure its communication channel so that an appropriate amount of bandwidth is allocated to transmit the video with the user requested quality level. This operation can be achieved through the interaction between the software application and the hardware devices, such as routers, switches, or access servers, that provide communication services. In addition, the QoS information base is also built with the help of agents running on devices. Therefore, QoS-enabled devices are essential for QoS architectures.

One of the main concerns about system operators is their networking quality. Most current distributed systems are IP-based because of the evolution of the Internet. However, IP is not appropriate to provide QoS because it does not support mechanisms for traffic control and congestion avoidance. Therefore, one of the preferred scenarios for current service providers is using IP to allow users to access the services, and then using QoS-enabled and/or faster protocols in their core network [Chao07]. One example of such protocols is MPLS (*Multi-Protocol Label Switching*), which provides traffic engineering and QoS features, such as bandwidth reservation or VPN (*Virtual Private Network*), etc. Typical current communication systems consist usually of small-scale routers providing the access at the edge of networks and large-scale routers, which are MPLS-enabled, at the core.

This part of the thesis presents the design of an efficient distributed software architecture for large-scale routers providing high quality services in the core networks by ensuring that the performance of the switching operations is fast enough. The core routers may be considered as the most critical components of distributed systems, due to the large amount of traffic to be switched. For example, some researchers claim that the growth of traffic in the core networks can reach 400% to 500% per year [Bu04]. Such a requirement leads to the emergence of robust and highly scalable routers. Resiliency and availability need also to be taken into account.

This chapter is organized as follows. Section 4.1 motivates our research and discusses the need of distributed architectures for router software. We next describe the key tasks of a router. In Section 4.3, we review the evolution of the router generations. Section 4.4 surveys the hardware architecture of the next generation routers. Section 4.5 discusses the software architectures of routers and review the studies on distributed routing architectures. A summary of the chapter is presented in Section 4.6.

4.1 Motivation

Highly scalable routers are required for today's core networks. Indeed, until very recently, core network operators answered the growth of Internet traffic by adding more routers, usually mid-size routers, in their network. This approach, referred to as router-cluster [Chao02], imposes extra cost for management and maintenance, particularly when the number of connections grows very fast. Due to the increase of the size of the router clusters, the routers increasingly use their expensive interfaces (e.g., optical ports) to carry intra-cluster traffic instead of value-added traffic. Another issue is that the number of control message exchanges in the network and the routing table size explode, which overloads the capacity of mid-size routers. In addition, these mid-size routers are not scalable to support high-speed optical interfaces (e.g., OC-192).

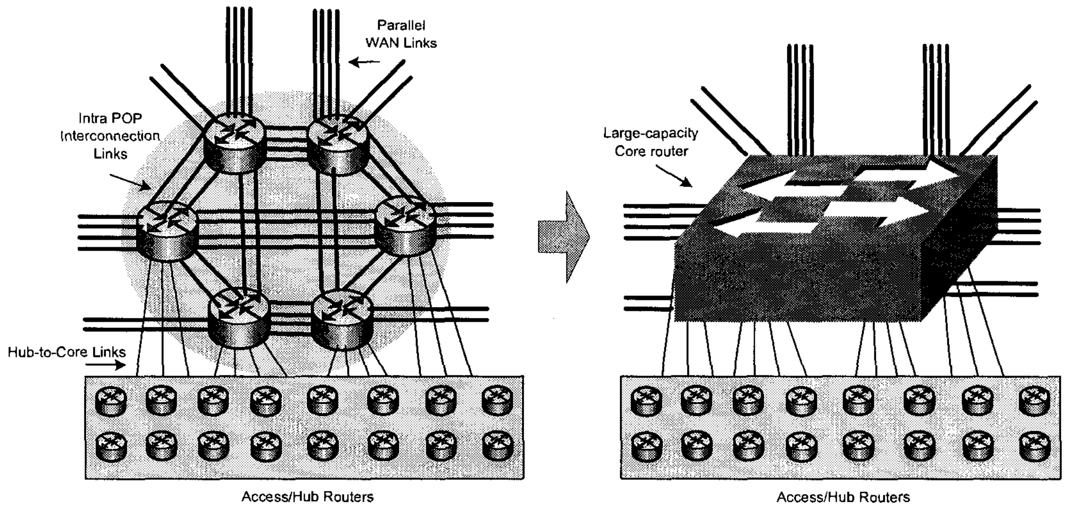


Figure 4-1: Replacing a Cluster of Mid-Size Routers with a Large-Capacity Scalable Router [Chao07]

Taking advantage of the hardware evolution, a more cost-effective approach to deal with the ever-increasing traffic in the networks consists in replacing a cluster of mid-size routers by a next generation router with large switching capacity of multiple terabits or even of a few *petabits* (Figure 4-1) [Chao07]. Such routers will satisfy the Internet traffic growth with no need to be replaced for a couple of years, keeping the core network configuration unchanged, therefore resulting in a more efficient and reliable system. When the traffic requirement increases, the operator adds new interface modules without replacing the whole router. The main challenge for this solution is to have a good router design with *high robustness*, *scalability* and *resiliency*, for both its hardware and software architectures.

The router should be scalable both in terms of the number of ports which can be connected to the router and in terms of the data forwarding capacity. In addition, it is also important that a router can be dynamically extended without disrupting the router normal operation [Hide06].

Due to the growing number of nodes in the Internet, the tables which must be maintained in the router, such as routing tables, forwarding tables, QoS policy, etc. become larger. As a consequence, the lookup operations performed on these tables take more time. In particular, the next generation routers should be able to handle a huge number of routes (e.g., some hundred thousands), such as BGP (*Border Gateway Protocol*) routes. A router with several BGP peers in a core network will have to handle large amounts of messages (e.g., flapping (refreshing) rate of 100,000 routes per three minutes [Nguy07a, Hype04]), which in turn consumes a significant amount of computing and memory resources.

In addition, the general requirement for router availability in Internet core network is 99.999% [Nguy07a, Hide06]. The resiliency is therefore important both in the control plane and data plane of the router. The router also needs to maintain forwarding functionality if the control plane temporary goes down and to provide redundant functionality [Shai06].

As prices of hardware material are sharply decreasing, more processors and memory chipsets can be added to the router platform in order to increase its overall power. For example, some recent products have been provided with thousands of optical interfaces and multiple terabits throughput (e.g., 400 Gbps throughput by 8,960 OC-3 interfaces or 2,240 OC-48 interfaces [Avic06], 2.5 Tbps throughput [Juni07]). Some models have been forecast to reach multiple *petabits* switching capacity with 64,000 optical interfaces [Dupl05]. The hardware architectures of the next generation routers are basically distributed, with the control and line cards interconnected by a very high speed switch fabric. The control card is designed with one or more powerful processors and a large memory capacity, aimed at running the main control and management tasks. The line cards do mainly data forwarding using built-in specific network processors and traffic manager chipsets. In recent router models, the line card is also equipped with extra memory and a general-purpose CPU, allowing it to share some processing tasks that tend to overload the control cards [Chao07].

One research issue is to design a software architecture that can exploit efficiently the new router hardware platform. In general, a distributed software architecture is required to fit into a distributed hardware platform. However, due to legacy techniques or business models, we observe that, even in the recent router products [Cisc05], some of the software components still remain centralized, particularly routing protocol modules. Since the control card of a router is responsible for all routing tasks, it can be easily overloaded by overwhelming traffic in core networks, especially when the routing table gets flopped (updated/refreshed). In addition, bottlenecks can be experienced in the centralized software architecture when the control card is unable to process the huge number of requests coming from different line cards. The time for route establishment and time to recover are also issues in a centralized architecture because every protocol message must go through the control card, leading to additional delay overhead. These limitations led to the need of distributed architectures for software implementation, particularly for very highly scalable routers. Indeed, the starting point of our study was the modular design approach, which has been presented in [Hide06, Dori07], where routing software components, namely the control plane and data plane, can run independently on the same or separate CPUs and interact with each other regardless of their respective physical location. We extend this approach to allow the router control plane processes to run on different router cards considering the task sharing among them. Distributed models for specific processes, such as routing, signaling and routing table management, are also taken into account.

4.2 Key Functions of a Router

Generally, a router must perform three fundamental functions: compute best routes, forward data packets, and ensure that service agreements are met.

4.2.1 Compute Best Routes

The first function is to compute best routes which data packets should take through the network to their destinations. The route computation has to take into account various policies and network constraints [Zini02]. For example, the best route can be required to maximize network efficiencies, to deliver the fastest possible response times to users, to minimize bandwidth usage costs, or to meet some other user specifications. In the current generation routers, the route computation is accomplished by a route processor (also called routing engine). Routers determine best paths by sharing inter-networking information with other neighboring routers. The route processor is actually the "brain" of the router and is dedicated to communication with neighbors. This communication enables the route processor to build a route database, or routing table, which allows the forwarding engine to send packets across optimal paths through the network. In addition, the route processor can communicate with other routers to provide them with its routing table which helps them to identify the best routes and select the optimal paths. Such exchanges are achieved by the routing protocols [Hala05]. The routing protocols exchange messages containing networking information. These messages are called routing updates. Every routing protocol has its own format for routing updates and its own algorithm for exchanging and analyzing the messages. All routing protocols can be classified as Interior Gateway Protocols (IGPs) or External Gateway Protocols (EGPs) [Zini02]. IGPs run inside an autonomous system (AS) and perform so called intra-domain routing functions. Widely used IGP protocols include the *Open Shortest Path First* (OSPF) [Moy98] and the *Intermediate System-to-Intermediate System* (IS-IS) [ISO02]. EGPs run between ASs. The currently most used EGP is *Border Gateway Protocol* (BGP) [Rekh95]. In a core router, the BGP module has to handle a very large number of routes (e.g., some hundreds of thousands of routes) and ASs (e.g., tens of thousands of ASs).

4.2.2 Forward Data Packets

The second function of the router is to forward data packets received on an ingress interface to the appropriate egress interface for transmission across the network. Forwarding relies on the best route information computed by the route processor. The forwarding function is achieved by *forwarding engines*. The forwarding engine consults a Forwarding Information Table (FIT) which contains a complete set of forwarding information for all destinations learned by the routing protocols or by the list of static routes. Based on the destination address and/or TOS fields of the IP packet header, the forwarding engine looks up the FIT to find the next hop and the appropriate egress interface to forward the packet. As traffic loads grow, the processing resources required for FIT lookups increase. Thus a router may need several forwarding engines.`

4.2.3 Service Function

The third major router function is the service function. A router should provide quality of service according to the traffic specification and the Service Level Agreements (SLA) between providers and their customers. This function is accomplished by a *service engine*. As mentioned, the router requires a servicing system to perform a set of tasks such as packet buffering, filtering, policing, shaping, marking, etc., in order to provide the proper QoS. Each ingress interface on a line card receives packets which are examined by a forwarding engine and directed to the egress interface associated with its destination IP address. An example of the role of the service engine is the case where multiple packets arriving simultaneously on different interfaces need to be forwarded to the same output interface. A queue must be provided as a temporary waiting area in which packets are queued up for transmission and ordered in the queue following their relative priority. The order in which they are transmitted is determined by the policy settings configured by the network administrator.

4.3 Evolution of Hardware Architectures of Routers

This section reviews the evolution of hardware router architectures and discusses the needs of the next router generation, which is targeted by our research on software architectures.

Three generations of routers have been recognized [Awey01, Hide06, Medh07].

4.3.1 First Router Generation: Bus-based with Single Processor Architecture

Routers of the first generation, born in late '70s or early '80s, were basically made of a single central processor (CPU) and multiple interface cards interconnected through a shared bus. The CPU runs a commodity real-time operating system and implements the functional modules, including the forwarding engine, the queue manager, the traffic manager, and some parts of the network interface, especially Layer 2/Layer 3 processing logic in software.

Figure 4-2 shows the architecture and data processing of the first router generation. An incoming packet (1) at a line card (so called ingress line card [Hala05]) is forwarded to the buffer memory through the shared bus (2). The central CPU extracts the headers of the packet (3) and uses the forwarding table (4-5) to determine the outgoing line card (so called egress interface) and port. The packet is subsequently prioritized by the queue manager (6) and shaped by the traffic manager (7). Finally, the packet is transferred from the memory (8) to the appropriate output port in the egress line card (9-12).

The central CPU saves some of its cycles, which are mostly used for packet forwarding, for running the routing protocols. Whenever a route change occurs, it updates the routing table and the forwarding table. The central CPU also executes the management functions for configuring and administering the router.

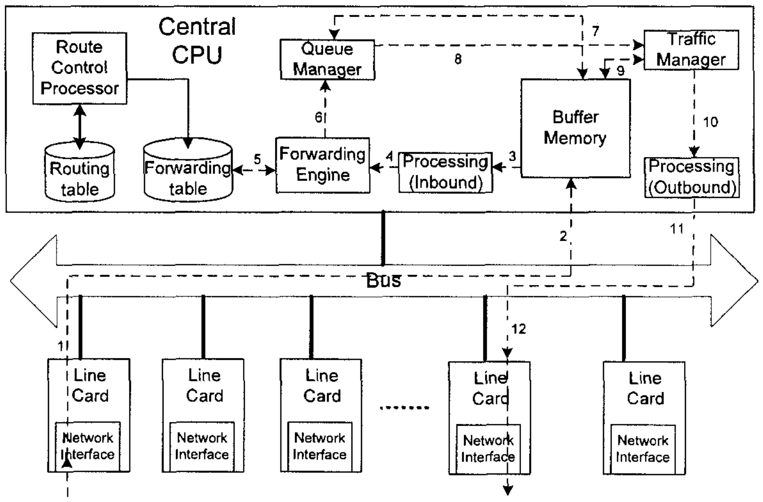


Figure 4-2: First Generation Router with a Single Central Processor and a Shared Bus

These routers suffer from two main drawbacks:

- Data packets travel through the bus twice: the first time from the ingress interface to the central CPU, and the second time from the central CPU to the egress interface. Thus, the bus is a severe bottleneck for the router throughput.
- Data packets are buffered in a centralized memory and the lookup operation needs intensive memory access, so bottlenecks can easily be experienced by the memory and central CPU.

Basically, the performance of these routers heavily depends on the throughput of the shared bus and on the speed of the central CPU; taking into account the current speed and memory parameters, these routers are not scalable and cannot meet today's bandwidth requirements.

4.3.2 Second Router Generation: Route Caching Architecture

The second router generation, presented in '80s, was designed with line cards able to perform some packet forwarding operations locally. Unlike the previous architecture, more intelligence is added to the line cards, with processor, memory and forwarding caches.

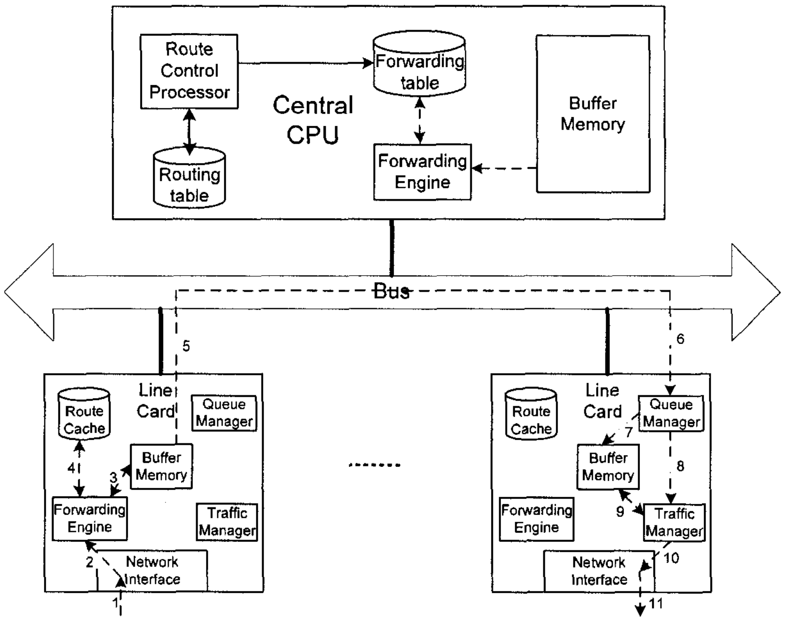


Figure 4-3: Second Generation Router with Route Cache Architecture

Figure 4-3 shows the architecture of a next generation router and the data circulation. The router has a central CPU maintaining a central forwarding table and the line cards cache a subset of the master forwarding table based on recently used routes. When a line card receives a data packet (1-2), it first looks the local cache for the next hop to forward the packet (3-4). If no entry is available in the cache, the line card sends a request to the central CPU. Otherwise, the data packet is transferred directly from the ingress line card to the queue manager of the egress line card (5-6) and then to the appropriate output port

(7-11). The advantage of this architecture is the increased throughput due to the forwarding cache of recently used addresses in the line card, which allows the line card to process packets locally most of the time. However, the shared bus is still a potential bottleneck because it does not allow more than one data packet to go across at the same time. In addition, the throughput is highly dependent on the incoming traffic. There is still an important amount of traffic that needs to travel the bus twice. Due to these drawbacks, this architecture can neither scale to high capacity links nor provide complex traffic pattern-independent throughput.

4.3.3 Third Router Generation: Switch-based Architecture

The third router generation, i.e., the current widely used router generation, was introduced in ’90s to solve bottlenecks of the second generation. The shared bus has been replaced by a switch fabric which allows multiple packets to be simultaneously transferred across, hence increasing the performance. The switch fabric is basically a crossbar connecting multiple cards together, thus providing large bandwidth for transmitting packets among line cards [Awey01, Chao07]. In this generation, multiple forwarding engine cards are connected in parallel to achieve high speed packet processing rates (Figure 4-4). Each forwarding engine card hosts a forwarding table. Thus data packets can be processed by the forwarding engine cards without going through the central CPU. When a packet comes in a line card, the packet header is stripped and sent to a forwarding engine on one of the forwarding engine cards for validation and routing. The forwarding engine determines the outgoing port where the packet should be transmitted. The packet is then moved from the source line card to the destination line card and eventually sent out to the next hop.

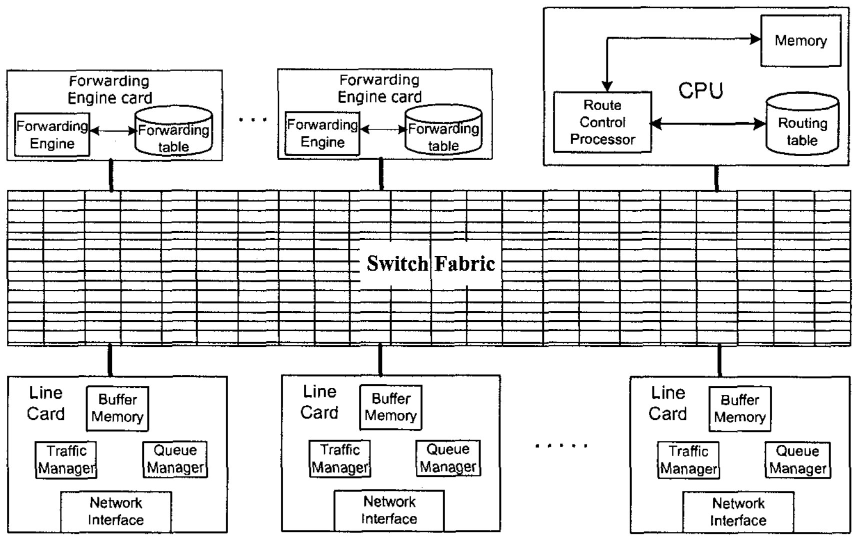


Figure 4-4: Third Generation Router with Switch Based Architecture

Basically, there are three main bottlenecks which can potentially be experienced in a first and second generation router: processing capacity, memory bandwidth, and internal bus bandwidth. Hence, the switching architecture has been deployed in the third generation routers in order to replace the internal bus. In addition, processing resource has been added on line cards in order to increase the processing speed and memory capacity. Simultaneous packets can therefore be transferred among different pairs of network interfaces. Multicast capabilities are also enabled. However, due to rapid growth of the Internet, the architecture is not able to meet the expected amount of traffic (i.e., multiple terabits or *petabits* per second). The separation of the forwarding engines from the line cards increases the load on the switch fabric for internal messages exchanged and adds extra delay for packet switching. The architecture is not scalable to support a large number of line cards. These issues led to the birth of the next generation routers, which we will next describe in Section 4.4.

4.4 Next Generation Routers

The hardware architecture of the next generation routers is essentially switch-based, with a switching capacity of multiple *terabits* or *petabits* per second, satisfying different QoS requirements [Chao02]. First commercial products that appeared in the market were Avici's TSR [Avic06], Juniper's T1600 [Juni07]. Some prototypes are also being developed, such as HyperChip's PBR1280 [Hype04]. We now describe the advanced features of the next generation routers.

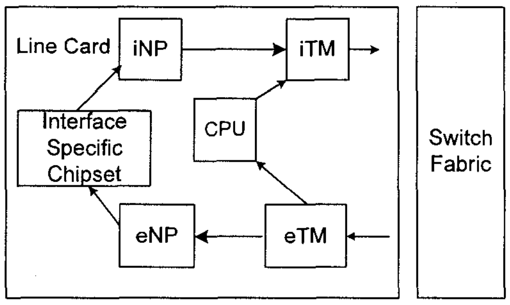


Figure 4-5: Components of a Typical Line Card

- The line card provides one or more interfaces to external devices (such as other routers) and connects these interfaces to the switch fabric, as shown in Figure 4-5. As in the third generation, data packets are processed locally on the line card without going through the control card. Principal elements of a line card include: interface specific chipset, network processors (NP), traffic manager (TM) chipsets, CPU and memory.
    - Network processors are dedicated to packet processing. Their tasks typically include: Media Access Control (i.e., to handle the raw data stream coming from the physical links), Packet Processing (i.e., header and payload processing), Packet Classification (i.e., to identify and

classify traffic according to the QoS requirements) and Policing and Traffic Management (i.e., to perform vendor-specific functions). Nowadays, a network processor can handle flows at OC-48 (2 Gbps) or OC-192 (10 Gbps) line rate or even faster [Chao07].

  - Traffic manager chipsets bridge the network processor and the switch fabric. The ingress TM (iTM) handles queuing and scheduling by applying different buffering strategies so that flows can share limited buffers according to the traffic requirements. The iTM is also responsible for multicasting. The egress TM (eTM) processes data from switch fabric before data are sent to the network processor. If packets are segmented into cells in the ingress chip, reassembling the cells into original packets is done by the eTM.

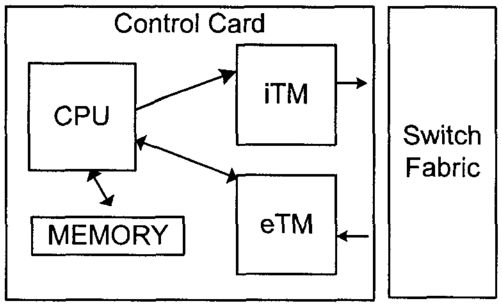


Figure 4-6: Components of a Typical Control Card

- The control card is designed to run the main tasks of the routing protocols (i.e., BGP, OSPF, IS-IS and MPLS), and the Routing Table Manager (RTM), as shown in Figure 4-6. Complementary modules such as the Command Line Interface (CLI) are also hosted by the control card in order to provide interfaces to the system user for configuration purposes. The control card architecture is very similar to that of a line card, but it has no line interfaces. The basic difference between them lies in the processing power and storage capabilities: they are far superior on the control card. The control card has one Ingress Traffic Manager (iTM) chip and one Egress Traffic Manager (eTM) chip. These chips provide an interface between the local processor and the switch fabric planes. The iTM and eTM chips are exactly the same as the ones used in the line cards.

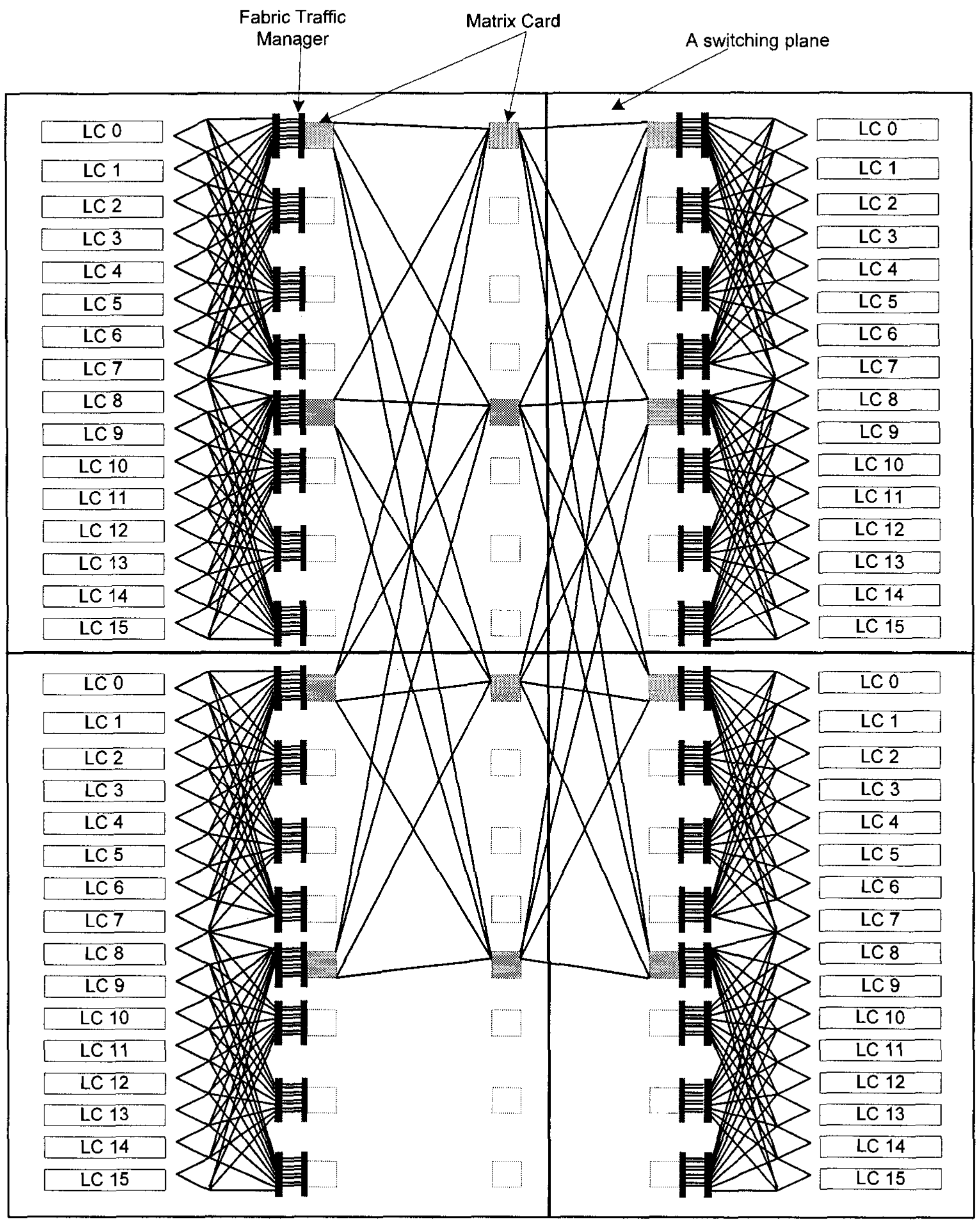


Figure 4-7: Sample of a Four-Plane Switch Fabric Interconnecting Router Cards

- The control and line cards are interconnected by a scalable switch fabric. The switch fabric is distributed into identical and independent switching planes. In our research, we rely on a switch fabric model provided by [Hype04], which consists of four planes as shown in Figure 4-7. The switch fabric is made of so called matrix cards, which provide data switching functions. Per-flow scheduling, path balancing and congestion management within the switch fabric are achieved by the Fabric Traffic Manager chipsets integrated on the matrix card. Each line card or control card has an ingress port and an egress port connecting to a matrix card. Each switching plane is made of the same number of matrix cards. In such a model, each switching plane has bandwidth to handle a full OC-48 port or equivalent. Several topologies may be used to connect the matrix cards where the Benes topology [Chao07] is the most recommended, due to its non-blocking characteristics. It allows the switching system to be non-blocking using only O(NlogN) switching elements, rather than the O(NxN) required for a crossbar topology.

Consider the forwarding and routing mechanisms in a next generation router as illustrated in Figure 4-8. Data packets come in by the iNP of the ingress line card, which contains a FIT table which is used to determine the path to the destination. Packet classification is also done by the iNP. Packets are then forwarded through the iTM where traffic engineering policies are applied. They then travel through the switch fabric to the egress line card. The eTM and eNP of the egress line card forward the packet to the next hop in the direction of the destination. Control packets, on the other hand, are filtered by the iNP of the ingress line card and forwarded directly to the control card or the CPU of the line card where they will be processed by the routing protocol modules. The iTM and eTM chipsets located on the control card are responsible for managing flows of control packets. Control packets may also be sent out to external routers in the network through appropriate line cards.

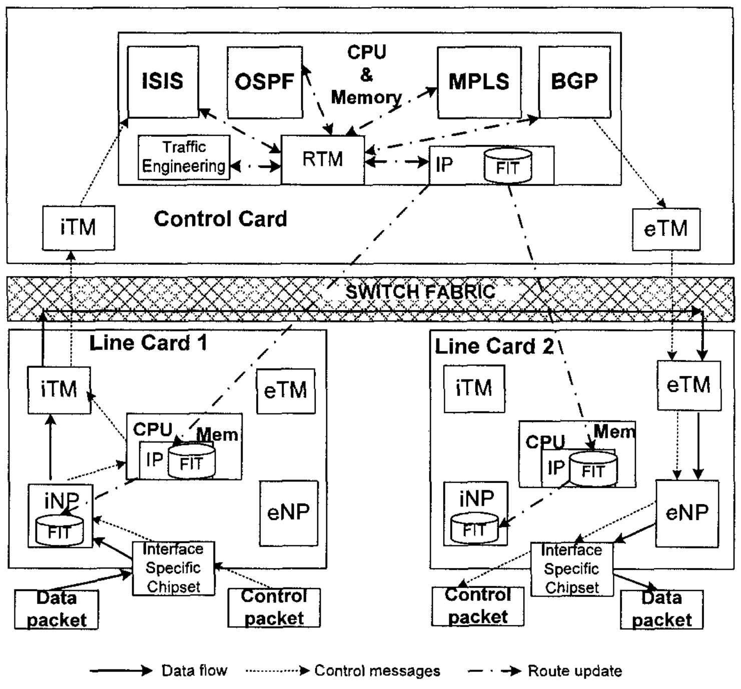


Figure 4-8: Architecture of Next Generation Router

One of the primary requirements for next generation routers is scalability. In general, a core router has to exchange control messages with hundreds of peers. According to the growth of bandwidth, a large number of line cards (i.e., few hundreds or thousands) needs to be added to the router platform. This imposes several challenges to the operation of routing protocols. Current generation routers provide a throughput of multiple *terabits*, while next generation routers, assuming a distributed architecture, will reach a throughput of a few *petabits* per set of thousand line cards. In some practical networks [Hype04], core routers are expected to support more than 300,000 routes with a flapping rate of 100,000 routes per three minutes, which exceeds the capacity of a single control card. Task sharing should therefore be taken into account, in order to make the system more scalable. In addition, resiliency is also an issue. One possible solution consists of having additional control cards. Each control card runs an instance of a routing protocol module or manages some parts of the global routing table. However, the control cards are often costly and the processing capabilities are not improved much due to the quantity and delay of messages exchanged between the different control and line cards in a system. Another solution is to take advantage of available resources on the line cards in order to perform some control tasks. This will be investigated in the next chapter.

4.5 Review of Software Architectures of Routers

This section provides a literature review of software architectures of routers. We begin with the monolithic architecture used in legacy routers and then present some distributed architectures for separating software modules.

4.5.1 Monolithic Software Architecture

Legacy routers (i.e., first and second generations) are built with one CPU on a control card handling all basic modules such as routing engine, packet forwarding and service engines. The routing engine handles a set of routing protocols such as IS-IS, OSPF, BGP and MPLS that run together and interchange information such as routes or labels. The exchange and coordination of these protocols are generally done via a Routing Table Manager (RTM). Figure 4-9 shows the software architecture of the first and second router generations. In such an architecture, the RTM is responsible for retrieving information learned from the different routing protocol modules, making decisions for selecting best routes and accordingly generating the best route table (FIT), which can be used later in forwarding the packets to the corresponding destinations.

The advantage of such an architecture is the ease of management since all the routing protocols run together on the same control card. The synchronization and message exchange mechanisms are also quite simple to implement. However, the main issue of such legacy systems is their monolithic code base with all forwarding and routing processes competing for the same CPU and memory resources. Consequently, as the demanding packet forwarding process consumes almost all the CPU capacity, the other functions are left starving for CPU cycles. Clearly this type of software architecture can only be used for small and medium size routers.

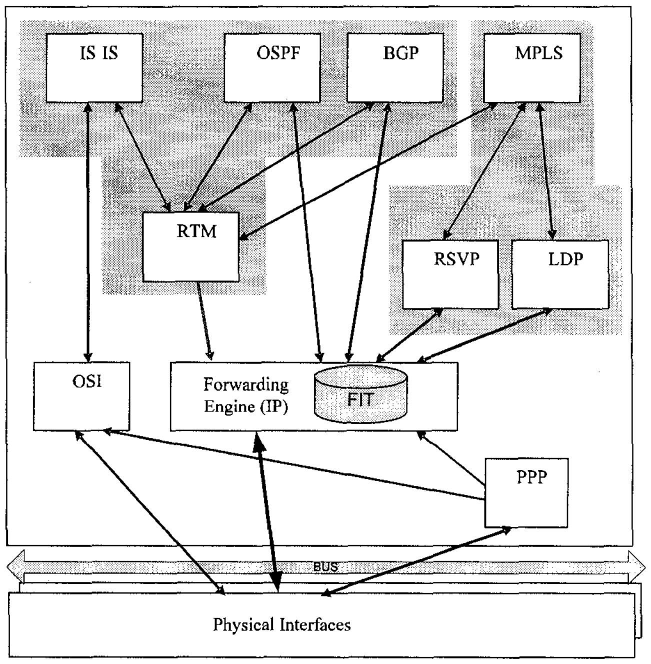


Figure 4-9: Software Architecture of First and Second Router Generations

The current generation of routers (third generation routers), as described above, consists of a control card and a set of line cards connected via a switch fabric. Line cards contain very high speed interfaces (i.e., OC-48). This new hardware capacity needs a more distributed software architecture in order to exploit the hardware platform. For instance, the current software architecture, as of the HyperChip PBR1280 [Dupl05] shown in Figure 4-10, consists essentially of:

- One controller card that hosts all routing protocols. There can be an additional control card used for backup and redundancy.
- A given number of line cards, which perform:
    - IP forwarding and/or MPLS label switching at the hardware level, and
    - IP forwarding at the software level for exceptional packets (e.g., control packets).

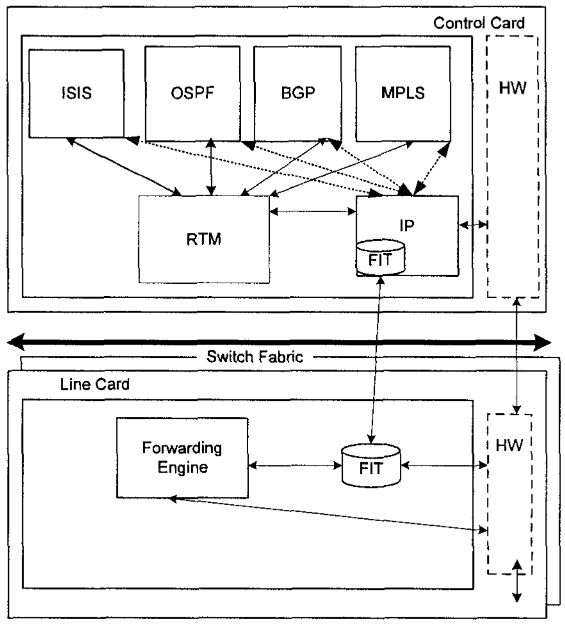


Figure 4-10: Software Architecture of the HyperChip PBR 1280

In addition, there is a FIT on each line card. The RTM located on the control card receives the best routes learnt by routing protocols. Overall best routes are selected and then recorded in the FIT of the IP stack. The FIT on each line card is downloaded from the FIT on the control card via the switch fabric. The performance and the fault tolerance of the router are hence improved because each line card is able to make the forwarding decisions by itself without the need to send requests to the control card or to use a separate forwarding engine. However, there is still a potential bottleneck at the control card where all routing protocols run simultaneously. Therefore, such an architecture is not scalable. The large capacity of line cards and the switch fabric is not exploited efficiently due to the location of almost all processor and memory resources for the computations on the control card.

Unfortunately, the largest vendors in the market, such as Cisco and Juniper, do not publish their software architecture, so we have no clear idea about the architecture of their control cards. However, in their recent products [Cisc05, Juni07], there is no control function running on the line cards. Therefore, all routing protocols should be handled by the control cards. This does not allow the control card to serve a large number of line cards. For example, the current Cisco 12000 series products cannot support more than 16 line cards per chassis [Cisc05].

Due to the growth of the Internet routing tables and the web-based traffic, the software architecture used in the current routers (third generation routers) becomes inefficient. Third generation routers are moving to the edge of networks and leave room for the next generation routers, which are much more powerful. For example, in 2005, HyperChip Inc. has announced a new core router model, which may support a very large number of line cards and control cards (e.g., 64,000) and able to provide a very high throughput up to 1280 Gbps. The software architecture for next generation routers should therefore be more distributed and more scalable.

4.5.2 Current Distributed Software Architectures

Until very recently, no distributed software architecture for the router control plane has been considered. Third generation routers are still provided with no extra memory on the line cards. Therefore control tasks are mainly performed at the control card level. The need of distributed software architectures for routers has led to the birth of the next generations.

The current research on the distributed software is mainly related to the router operating system, which is originally motivated by the trend to extend the forwarding function of routers. The existing architectures, mostly based on open-source, are aimed primary at providing a software prototype to implement interfaces or communication protocols among router components. The router prototypes on which the software architectures are developed are small scale, or even general-purpose computers with no specific chipsets. The software architectures are still implemented with no control function on line cards. More precisely, the line card is considered as a simple forwarding element of the router where no routing, signaling or management task can be hosted.

We classify the current distributed software routers in two categories:

- *Distribution of processes*. In this category, the router software is composed of independent processes, which can run simultaneously on the hardware platform. Each control function (i.e., a routing protocol) is achieved by an independent process.

  A specific example of such an architecture is the Router Plugin [Deca00]. The software framework supports dynamic loading and unloading of plug-in modules at run-time into the kernel of the operating system (OS). Each protocol or forwarding engine is defined as a module. The architecture is implemented in the Net BSD operating system kernel, which is an open source Unix operating system. The forwarding engine is designed with extended functions, such as packet-to-flows mapping and filtering.

Another example of distributed software in this category is XORP (*eXtensible Open Router Platform*) [Hand05] which is also open-source and Unix based. In XORP, the routing software is modularized into one process per protocol and extra processes for management, configuration and coordination. It also defines a forwarding engine abstraction (FEA), which allows running the higher-level subsystem on top of different types of forwarding engines.

- *Distribution of tasks*. In this category, each router function (i.e., routing protocol or forwarding) can be split into different tasks. Each task can be achieved by a router component. Thus a function may run at different locations of the router.

  A typical example of such architectures is ForCES, presented by the IETF [Dori07], which can be considered as the most notable framework for distributed routers. The ForCES architecture is defined in terms of exchange of information between control elements (CEs) and forwarding elements (FEs). A group of CEs and FEs together forms a network element (NE) which can be considered as a router in the traditional sense. The ForCES protocol is used to associate the CEs and FEs. It updates the FEs with configuration information from the CEs, queries for information by the CEs or sends asynchronous event notifications to CEs. Using the ForCES protocol, the CEs may also configure the processing functions on the FEs.

Based on the ForCES architecture, one can attempt to redefine the control functions of a router in order to share the processing tasks between the control cards and the line cards. In [Deva03], the authors present a Distributed Control Plane architecture, where some message processing, particularly HELLO processing, is handled by the line card. An example for distributed OSPF architecture demonstrates that when HELLO processing is moved to line cards, failures can be detected faster and Shortest Path First (SPF) calculations can be run as frequently as required without affecting the load on the control plane processor.

Another example of distributed software in this category is the distributed OSPF architecture presented in [Hype04]. It is a partial distribution of OSPF consisting of message processing and link state databases located on the line card. The control card handles the SPF computation and routing table management. A specific protocol is designed to achieve the communication between control and line cards.

Basically, the first category is not able to efficiently exploit the next-generation router hardware platforms because the computing resources on line cards are not used for control processing. It is rather suitable for third generation routers with multiple control cards and therefore is not considered in this thesis.

The second category seems more suitable for next generation routers. However, the current ForCES architecture does not consider some specific hardware features of the next generation routers such as the general purpose CPU and available memory on line cards. As described, the forwarding element in the ForCES architecture may correspond to the network processor (NP) on line cards of a next generation router. Since the network processor is required for key data processing functions, such as table lookup or flow classification, the integration of some control functions into this forwarding element as proposed in [Deva03] may slow down the data forwarding speed of the line card. In addition, the NP is often designed for specific interfaces so reprogramming the NP is costly.

Our approach presented in the next chapter enhances the ForCES framework by exploiting the general-purpose CPU and memory on line cards, instead of the network processor. This keeps the forwarding element intact so the forwarding performance is not influenced by new control functions implemented on the line card. In addition, using the general purpose CPU on the line card allows more control functions to be offloaded from the control card in order to increase the scalability.

4.6 Chapter Conclusions

The purpose of the second part of this thesis is to study architectures for IP routers, which play essential role in providing the QoS in distributed systems. The key tasks of a router include the best route computation, data forwarding and service provisioning. As the traffic in the core network is increasing rapidly, new architectures for routers are required in order to improve the robustness, scalability and resiliency.

This chapter has reviewed the hardware architectures of three router generations and discussed their evolution. The first and the second generation routers have been designed with a common bus which is not able to serve a large number of interfaces and traffic requirements. The current widely used routers belong to the third generation which is made with a switch fabric. It enables the parallel processing and increases the number of interfaces the router can support. We have also investigated the structures of the control card and the line card of a router and the internal processing mechanisms for data and control packets.

The hardware architecture of the next generation routers has then been analyzed and we focused on the large switching capacity, computing and memory. We also reviewed the software architectures of current routers, which are mostly monolithic, and some trends of software distributions, namely the distribution of processes and distribution of tasks.

We demonstrated that the current software architecture does not fully exploit such a robust platform, thus new distributed software architectures need to emerge. To this end, the next chapters will propose a new framework to develop distributed software architectures, followed by the designs of specific modules, such as routing, signaling, Routing Table Manager and MPLS.

Chapter 5 Proposal for Distributed Software Architecture for Next Generation Routers

In this chapter, we propose a new distributed and scalable framework, aimed at redesigning the current software architectures to fit into the new hardware platforms of the next generation routers. The chapter consists of two parts. The first part is dedicated to a new distributed software framework we propose for next IP generation routers where we focus on generic distributed architectures for routing and signaling protocols. In the second part, we present the application of the generic distributed architectures to specific protocols, such as OSPF and MPLS/LDP.

The generic architecture presented in the first part is based on the redistribution of many of the existing functions on different router components without modifying them, while maintaining their interfaces with other functions. In particular, most control functions will be located on the line cards instead of on the control cards. It goes along with the transfer of some tables, e.g., forwarding table, routing tables and adjacency tables, from again the control card to the line cards. For both routing and signaling protocols, we first review the centralized architecture before proposing the generic distributed one, and discuss the feasibility of its implementation. We also assess the advantages of such a distributed architecture. The objective of such a redistribution of the functions of the routing and signaling protocols is to increase the scalability and resiliency. Because of the distribution, control tasks can be processed in parallel on different hardware components of the router. In the next chapters, we will present a detailed analysis of the performance of the distributed architecture.

In the second part of this chapter, we discuss the ability of applying the proposed distributed architectures for the design of distributed architectures for OSPF and MPLS/LDP. Existing centralized architectures of these protocols are analyzed. We propose then the software modules to be distributed. Distributed architectures for related components such as RSVP-TE and Routing Table Manager (RTM) are also proposed.

In chapters 5, 6 and 7, we mainly focus on how to design distributed and scalable software architectures that fully exploit the distributed hardware platforms of next generation routers. Such software architectures include:

- *A generic distributed architecture for routing protocols.* We develop a model to implement the routing protocols, particularly IGP protocols, for next generation routers in a distributed way. This is presented in Section 5.1.1.
- *A generic distributed architecture for signaling protocols.* We develop a model to implement the signaling protocols for next generation routers in a distributed way. This is presented in Section 5.1.2.
- *A distributed architecture for a typical routing protocol, namely OSPF.* We apply the generic distributed architecture for the OSPF protocol, taking into account the specific features of OSPF. The distributed OSPF architecture is RFC 2328 compliant. The general description of the proposed architecture is described in Section 5.2. The performance evaluation of the architecture is discussed in Chapter 7.
- *A distributed architecture for a typical signaling protocol, namely LDP.* We investigate the application of the generic distributed architecture for LDP. The architecture is RFC 3036 compliant. This is presented in Section 5.2.2.2. The

details of the implementation architecture are provided in Chapter 6 where we discuss the challenges and their solutions, as well as the resiliency issue.

- *Distributed architectures for the Routing Table Manager.* We propose three distributed architectures for the Routing Table Manager, taking into account the capacity of routers and the architectures of the routing protocol modules. This is presented in Chapter 7.
- *A distributed architecture for MPLS.* We develop a distributed MPLS architecture for next generation routers, including both data and control planes. The architecture is RFC 3031 compliant. This is presented in Section 5.2.2.2 and in Chapter 6.
- *A distributed architecture for RSVP-TE.* We investigate a distributed architecture for RSVP-TE, which is similar to LDP protocols, with additional features for traffic engineering. The architecture is RFC 3209 compliant. The mechanisms we propose for LDP message processing can be reused for RSVP-TE and the distributed path computation supporting the traffic engineering is as described in the third proposed distributed RTM architecture. The description of the distributed RSVP-TE architecture is outlined in Section 7.1. It has been thoroughly investigated in the M.Sc Thesis of Saloni Neri [Neri07].

5.1 Generic Distributed Architecture for Routing and Signaling

In order to take advantage of the next generation router platform which provides additional processing and memory resources on line cards, we investigate the ability of moving some control functions from control card to line cards. This section provides a basis for such a distribution with the new challenges. Existing protocols in the router will be re-implemented based on this approach with respect to the RFC specifications. The architecture we propose is based on the following assumptions:

- The router is based on next-generation architectures, consisting in a distributed platform, with separated control cards and line cards. Lines cards have full capacity of memory and processing power, and are able to perform all the data forwarding and some control tasks.
- The communication between router cards is achieved through a specific device (called Switch Fabric, *SF*) that is able to provide the required bandwidth and other QoS demands. The forecast switching capacity is in order of a few *petabits*.
- There is a specific communication channel between line cards, enabling them to exchange control information with a negligible impact on data flows. This channel shares the bandwidth on the switch fabric with data flows. In our implementation environment, see [Dupl05], this channel is designed as an abstract layer called Distributed Service (DS). It provides a synchronization mechanism to manage module activations, monitoring and state transitioning facilities (active, backup, in-service upgrade, etc.). DS maintains a distributed database allowing requesting modules to get appropriate messages. Thus messages needed to be sent are flooded to DS and the destination will be notified.

The key features of the software architecture we propose are as follows.

- A control component runs on a control card assuming the cooperation of the different line cards and interfacing with other modules (e.g., user interfaces, management units, etc.). Additional control cards can be added to share processing tasks. However, as load balancing at the control card level is not investigated in this thesis, we can assume the router has only one control card.

- A link component runs on each line card of the router, taking care of the protocol procedures like route establishment, update advertisement, notifications, etc. Link components are loaded into built-in processors and use available memory capacity of the line cards.

A communication protocol is used for message exchanges between the control card and the line cards. This protocol runs on top of DS and interoperates the control components and link components in order to establish routes over networks.

Basically, the software implemented on the router platform has to carry out the following three primary functions:

- *Data Forwarding.* This function can be entirely achieved by the line cards in the current generation routers. The control card is no longer involved in this procedure. Once routes have been established and recorded into the routing table, data can be forwarded easily by network processors. Therefore the distributed software architecture we propose focuses on the routing and does not deal with data forwarding.
- *Routing and Signaling.* This is basically the information exchange (e.g., link state information) between different internetworking nodes in order to determine the paths through a network. In the centralized architecture, this function is performed by the control card. The distributed software architecture we propose aims to migrate most of this function to the line cards. The control card can be required, from time to time, to provide a global view of the network topology, according to the needs of the path computation.

- *Control and Management.* This includes different tasks used to select optimal paths learnt from different routing and signaling protocols, addressing mechanisms, routing table updates and notifications, user interfaces and interfaces between different modules. In the distributed software architecture we propose, this procedure can be shared between the control card and line cards.

Some recent studies [Deva03] have also presented the distribution of the control plane based on the functions performed by control cards and line cards. For example: i) link-specific functions are performed on line cards, ii) update functions are performed by control cards, and iii) protocol-specific functions need to be considered on a case-by-case basis for distribution and there is no standard model. This approach can deal efficiently with medium scale routers (e.g., routers with some tens of line cards and few hundred interfaces [Deva03]). However, core routers, and especially *petabit* routers require enhanced distributed architectures in order to run on their high-scale hardware platform (e.g., thousands of line cards). The software architecture we propose in this section provides a full distributed mechanism where the main tasks of the control plane are able to run on the line cards. We take into account the nature of routing and signaling protocols in order to propose an appropriate distribution scheme.

The distributed software architecture we propose, inspired from a peer-to-peer model for distributed multimedia applications [Nguy07b], assigns the data forwarding and signaling tasks to the line cards, and shares the control between line cards and the control card. The router can therefore be considered as a distributed system consisting of a group of line cards playing the same role, and one or some control cards acting as super nodes. The direct communication between any pair of router components (i.e., line card or control card) making the router to be similar to a peer-to-peer system, particularly in the case of a very large number of line cards.

Basically, the principal function of a router is to establish routes between two or several nodes. It is usually achieved by two types of protocols: routing and signaling. Routing protocols usually require route computations which identify the best routes while signaling protocols may not need the best routes as they are primarily dealing with reacheability. Hence, we next present the proposed distributed software architecture which will be then refined for each type of protocols in subsequent sections.

5.1.1 Routing

Routing protocols are deployed to determine the best routes from one router to other nodes in a network [Moy98]. It means that the router has to compute the paths based on information collected from other nodes. So, it must be aware of the whole network topology which is generated combining the route update messages sent by other routers in the network and its own routing information. Our distributed architecture for routing protocols aims mainly at the IGP protocols, such as OSPF and IS-IS because the load imposed on the control card to process these protocols is heavy. The EGP protocols, particularly BGP, are not considered in this thesis. Indeed, one (or few) dedicated control card is usually used to host the BGP module in the current routers [Dupl05]. The distribution of the BGP on line cards needs more complex solutions in order to overcome the current implementation issues related to the loopback address that allows a router to communicate with a given remote BGP speaker through any of its line cards. This will be taken into account in some future work.

5.1.1.1 Current Centralized Architecture for Routing Protocols

Current routing protocols are mainly processed at the control card level. All protocol messages must go to the control card where a protocol processing module. This architecture works as follows (Figure 5-1):

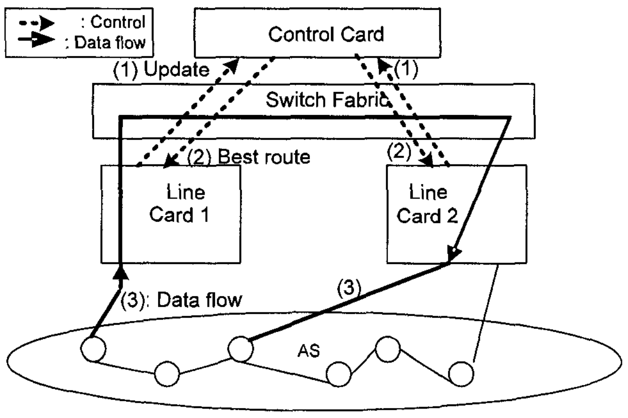


Figure 5-1: Current Centralized Routing Model

- When the router is connected to a network, the routing protocol module running on the control card discovers its neighbors. It broadcasts Hello messages through all line cards to reach direct connected neighbors. Specific discovery messages can also be addressed to given non-direct connected destinations indicated by the user.
- Route update messages come from neighbors through a line card ports are sent to the routing protocol module located on the control card. Similarly for a link change which is detected by the link manager module running on each line card. The control card re-computes the best routes for the corresponding routing domain. Usually, all line cards connecting to a given routing domain will send the same route update message, therefore the control card waits for a convergence before executing the computation. New best routes are then update to all forwarding tables located on the line cards.

5.1.1.2 Proposed Distributed Architecture for Routing Protocols

In order to apply the distributed software architecture for an IGP routing protocol, we will assign a line card as *proxy* in charge of the route computation for a routing domain where the router connects to. In practice, it can be the first line card on which the routing protocol is activated, or the first line card that receives the update message. Every line card willing to participate to the routing procedure for a domain must send a PROXY message to the control card in order to get the address of the proxy of this domain. Route update messages which a line card receives or discovers by itself must then be forwarded directly to the proxy. The proxy computes the best routes for the domain then updates the global routing table handled by the control card.

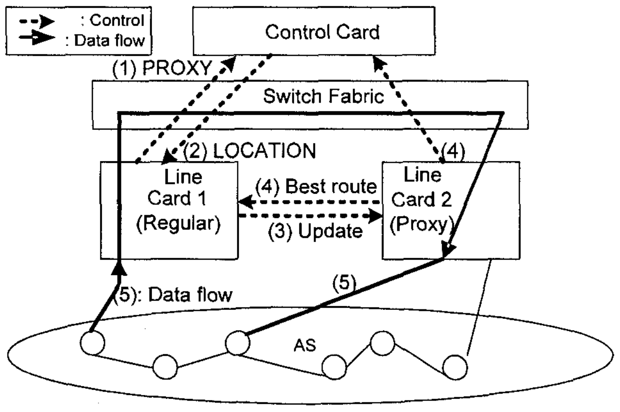


Figure 5-2: Distributed Routing Model

In order to achieve this objective, the proposed model proceeds as follows (Figure 5-2):

- When the router is connected to a network, each link component running on a line card that acts as a peer has to discover its neighbors. This procedure can be achieved by broadcasting Hello messages to all physical interfaces of the line card (e.g., in case of OSPF protocol) or by sending a request to a given destination indicated by the user (e.g., in case of BGP protocol [Bu04]).

- When a link component receives a route update message from neighbors or detects a link change, it sends a PROXY message to the control card to obtain the location of the line card that handles the route computation for the domain to which it belongs. The control card lookups its database for the proxy of the corresponding domain. A LOCATION message is sent back to the requesting line card indicating the location of the proxy. The requesting line card then directly forwards the update message to the proxy. In case of control card failures, the link component continues to send periodically HELLO messages in order to maintain the adjacency. If the proxy is temporary down while there is a new request, the control card assigns the proxy tasks to another line card in the domain in order to achieve the route computation.
- Upon receiving the update message, the proxy rebuilds the network topology and re-computes the best routes. Then it sends them to the control card in order to advertise them to all line cards in the system. The network processors will use these routes to forward the data packets.

5.1.1.3 Distributed vs. Centralized Architecture for Routing Protocols

The proposed distributed architecture has exactly the same functions for all routing processes associated with the conventional routing protocols as the centralized architecture. It maintains interfaces with other software modules. Neighbor routers view a distributed architecture based router and a centralized architecture based router the same because their protocol messages are the same. We compare the centralized and distributed architectures on the following points:

- *Neighbor discovery.* Both architectures discover neighbors by sending Hello messages through line card ports. The number of Hello messages sent in both case is the same.
- *Reception of update messages.* For a given routing protocol, both architectures receive the same number of update messages (one message per line card port).
- *Best route computation.* The best route computation is achieved in a sole component of a router for both architectures. In the centralized architecture, it is handled at the control card level, and in the distributed architecture, it is processed by the proxy line card. However, in both cases, they have the same link state database built from the same update messages, therefore the results of the route computation will be the same.

When the routing functions are distributed onto line cards, we need to take care of the synchronization of link state databases (LSDB) among line cards in the same domain. We propose to add an LSDB synchronization component to the proxy line card as described in Section 5.2.1.2, allowing the proxy line card to update the other line cards in its domain about newly received link state messages. Such a mechanism ensures that all line cards in the same domain have the same LSDB because they receive the same update from the proxy line card. Since all link update messages are sent first to the proxy line card of each domain, the proxy line card contains the complete LSDB of the domain. After having computed the best routes of the domain, the LSDB synchronization component of the proxy line card sends the new best routes and the updated LSDB to the other line cards. Thus, the LSDBs on all line cards belonging to the same domain are identical.

5.1.2 Signaling

Signaling protocols, on the other hand, are aimed principally at establishing a path, which is not necessary the best route, between two nodes in the network [Zhan02]. Figure 5-3 shows a diagram of a router connecting two different *autonomous systems (AS)* AS1 and AS2, respectively using Line Card 1 and Line Card 2.

5.1.2.1 Current Centralized Architecture for Signaling Protocols

Basically a typical signaling protocol (e.g., LDP) module implemented in the current router works as follows.

- When there is a request for routing from AS1 to AS2 through the router, Line Card 1 (*ingress*) forwards the request to the protocol module running on the control card.
- The control card determines the line card which is used to connect to AS2 (Line Card 2 - *egress*). The control card forwards then the request to AS2 through Line Card 2. The control card must therefore maintain a routing table containing requests from all line cards. An entry in this table represents an LSP.
- When there is a response from AS2, Line Card 2 forwards the response to the control card in order to complete the corresponding entry of the routing table. The control card sends a response to AS1 through Line Card1. The control card updates the forwarding table on all line cards with the new entry of its routing table. A data flow can then be triggered between AS1 and AS2 through Line Card 1 and Line Card2.

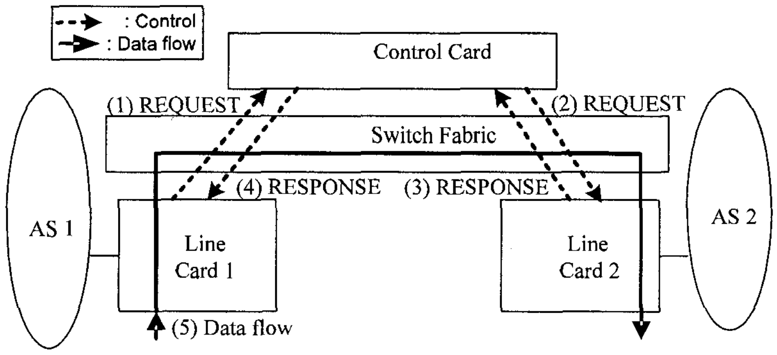


Figure 5-3: Current Centralized Signaling Model

5.1.2.2 Proposed Distributed Architecture for Signaling Protocols

Our distributed model proceeds as follows (Figure 5-4):

- When the router is connected to a network, each link component, acting as a peer has to detect its neighbors (i.e., AS1 for Line Card 1 and AS2 for Line Card 2).
- When there is a request for routing from AS1 to AS2 through the router or if the router wants to build a path between two ASs following a user request, the link component on Line Card 1 (*ingress*) sends a SEARCH message to the control card requesting the address of the line card (*egress*) to be used in order to establish a connection with AS2.
- The control card replies by a LOCATION message containing the address (e.g., IP address) of Line Card 2. The control card maintains a routing table allowing it to be aware about the topology of all networks the router connects to. There are two possible situations:
    - o If there is only one connection to AS2 (e.g., through Line Card 2), by looking up the routing table, the control card is able to determine the egress line card.

- If there is more than one connection to AS2, the control card has to compute the most appropriate line card based on specific parameters defined by routing protocols (e.g., link state).

- When receiving the LOCATION message from the control card, Line Card 1 sends a REQUEST message to Line Card 2 asking for a connection to AS2. In general, some additional parameters (e.g., protocol specifications, QoS or traffic engineering) are also included in the REQUEST message.

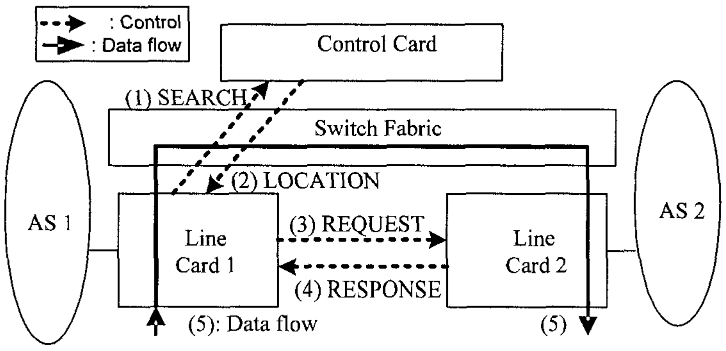


Figure 5-4: Signaling Model

- Line Card 2 is responsible for negotiating with AS2 about the new route. It can be done with an iterative procedure where routing parameters are re-negotiated between AS1 and AS2 with the help of REQUEST and RESPONSE messages exchanged between Line Card 1 and Line Card 2 until they can settle an agreement for their requirements.

- When two ASs and the router agree about the parameters for the new route, a new entry will be added into the routing table of the router by which the forwarding engine can trigger the data flow between Line Card 1 and Line Card 2. The control card can also be notified about this update.

5.1.2.3 Distributed vs. Centralized Architecture for Signaling Protocols

Again, the proposed distributed architecture for signaling protocols has exactly the same functions for all signaling processes associated with the conventional signaling protocols as the centralized architecture. They only differ in their implementation: distributed vs. centralized. Interfaces with other software modules are maintained. Neighbor routers view a distributed based router and a centralized architecture based router the same because their protocol messages are the same. We compare the centralized and distributed architectures on the following points:

- *Neighbor discovery.* Both architectures discover neighbors by sending Hello messages through line card ports. The number of Hello messages sent in both cases is the same.
- *Determination of the egress line card.* The egress line card is determined for each request in both architectures by the control card. Therefore, the result is the same in both cases. It is similar for a response.
- *Messages.* In the centralized architecture, a request message is sent from the ingress line card through the control card to the egress line card. In the distributed architecture, it is sent directly from the ingress to the egress line card. Since the egress line card is the same in both architectures, the destination of the request message is unchanged. It is similar for a response message.

In the distributed architecture, there is a duplication of processing task on the line cards. For example, each line card involved in a given LSP keeps a copy of information about the LSP. Therefore, the parallel processing is enabled.

In the proposed distributed architecture, there is a synchronization issue related to two line cards involved in a given LSP, i.e., Line Card 1 and Line Card 2 in Figure 5-4. In order to send data from domain AS1 to domain AS2 and vice versa, Line Card 1 and Line Card 2 must contain the same information about the LSP. This is achieved by REQUEST and RESPONSE messages exchanged between Line Card 1 and Line Card 2. However, if one of these messages is lost due to software errors or internal transmission channel errors, it will creates some inconsistencies in the databases of Line Card 1 and Line Card 2. In order to avoid those inconsistencies, an acknowledgement mechanism is proposed as described in Section 6.3.1, where each REQUEST and RESPONSE message is associated with an acknowledgement. Data transmission is trigged on the LSP only when all acknowledgements are received. The synchronization issue is solved by such a mechanism because it makes sure that protocol messages are exchanged in sequence, e.g., if the REQUEST is not yet received, a RESPONSE will not be sent. In addition, domain AS1 starts sending data only after receiving the confirmation from domain AS2 and the LSP information of the two line cards is complete and identical.

5.1.3 Advantages of a Distributed Architecture

Obviously, migrating some of processing tasks from the control card to line cards can reduce potential bottlenecks experienced on the control card when the number of requests is increased, according to the number of line cards and routes the core router has to support. The most important feature is that our model can take advantage of the additional resource available on line cards of the *petabit* router. In addition, the model we propose has the following advantages:

- *Performance.* Parallel processing is available in our model and waiting queues can be avoided. Line cards can independently process the routes they are involved in, without having to wait for the reply from the control card.
- *Scalability.* The router will be more scalable if some control tasks, particularly the signaling, can be processed by line cards. The control card will assume only the most complicated tasks, the tasks that need human interactions or the tasks used to interoperate different line cards.
- *Resiliency.* If the control card is required to perform all control tasks, system will be totally shutdown when the control card fails. One of the possible solutions is to have an additional control card, to be used as a backup for the primary control card [Ngu07a]. However, control cards are often costly. Having a backup mechanism at the line card level as described in Chapter 6 provides a better solution: it is faster to recover from line card failures; moreover a line card is much cheaper than a control card.
- *Availability.* Since HELLO messages can be sent directly from the line cards, the time to recover from failures will be reduced. The resulting congestion at the control card level will not slow down the procedures on the line cards.

On the other hand, the distributed architecture can raise some additional management overheads, as follows.

- *There are more messages exchanged between the control card and line cards.* Although the heaviest processing tasks have been eliminated on the control card (i.e., route computation and message sending/receiving), the control card is still responsible for the cooperation of the line cards. Such activity is supported by an internal protocol, thus additional message exchanges are required.

- *The software complexity is increased.* the router can be seen as a completely distributed system hence additional functions must be implemented, such as timing and inter-card synchronization. Line card software should also provide extra-functions, such as message processing, table management or inter-protocol communications.

Although there would be some trade-off due to the migration of control functions from the control cards to the line cards, we believe that the proposed distributed architecture is a good candidate for dealing with next generation router issues, particularly with a large number of line cards.

5.2 Case Studies

5.2.1 Routing: A Distributed OSPF Architecture

The OSPF (Open Shortest Path First) protocol [Moy98] is used for computing the shortest paths from one router to other nodes in a network. It is a hierarchical interior gateway protocol (IGP) for routing in IP networks, using a link-state in the individual areas that make up the hierarchy. A computation based on Dijkstra's algorithm is used to calculate the shortest path tree inside each area. In each OSPF-enabled router, a link state database (LSDB) is constructed as a tree-image of the network topology, and identical copies of the LSDB are periodically updated on all routers in each OSPF-aware area.

5.2.1.1 Overview of Centralized OSPF Architecture

The OSPF module is implemented in current routers in a centralized way at control card level as shown in Figure 5-5. A filter is defined at the iNP of each line card in order to forward all OSPF messages to the control card where they are processed by an OSPF module. There is a Link State Database (LSDB) managed by the OSPF module. The LSDB contains information of all OSPF links. OSPF best routes are computed from LSDB and updated to the RTM. They will then be compared to best routes coming from other protocols (i.e., BGP, RIP, IS-IS, etc.) in order to select the overall best routes. This final result is recorded to the FIT of the router through the IP stack.

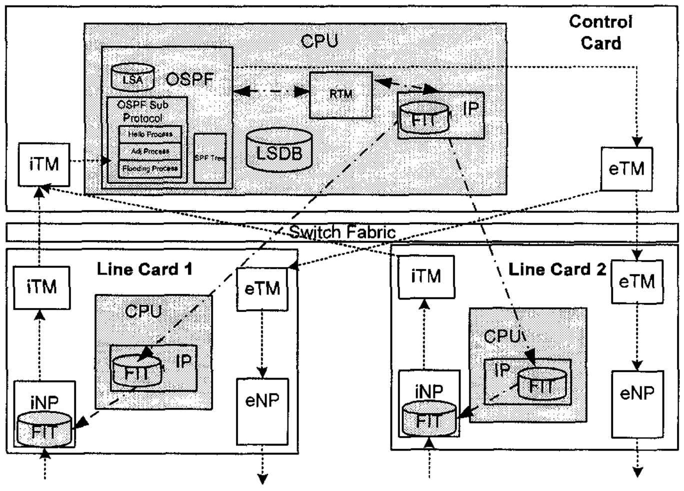


Figure 5-5: A Centralized OSPF Architecture

Basically, such a centralized OSPF architecture consists of the following components:

- *A link state database (LSDB),* containing all OSPF links of the routers,
- *A link state advertisement table (LSA),* containing the advertisements the router has to send to its OSPF neighbors,
- *A SPF tree computation component*, used to compute the OSPF network topology from the LSDB,
- *A Hello process*, used to discover OSPF neighbors. Hello messages are sent from this process to neighbors through appropriate line cards,

- *An adjacency management process*, used to manage OSPF neighbors,
- *A flooding process*, used to send OSPF advertisements to all OSPF neighbors.

As discussed in the previous section, such a centralized architecture will easily lead to overloading problem at the control card, especially when the line cards are added to satisfy the growing traffic demand.

5.2.1.2 A Distributed OSPF Architecture for Next Generation Routers

One of the distributed implementation models for OSPF has been proposed in [Deva03] based on the ForCES framework [Yang04], where Hello protocol is handled by line cards while the database synchronization and the path computation are still performed by the control card. Some experiments have been conducted with a router platform having 10 line cards. However, it remains a partial distributed architecture and the number of line cards to be tested is limited. On an advanced architecture containing thousands of line cards and interfaces, overload can easily be experienced.

Note that distributed OSPF architectures are RFC compliant. In other words, all functions provided by a distributed architecture are the same ones in the centralized architecture. The difference between the two architectures resides in the internal processing mechanisms. While all messages are processed at the control card in the centralized architecture, in the distributed architecture they are processed at different locations of the router. In [Hype04], the authors have also implemented a distributed OSPF architecture where line cards handle OSPF messages processing and link state database management. The path computation is still performed on the control card. We improve this implementation with our proposed generic distributed architecture for routing protocols. In our architecture, not only the Hello protocol but also control tasks (i.e., path computation and database synchronization) can be migrated to line cards, increasing the scalability, availability and robustness of the system. The distributed OSPF architecture (Figure 5-6) consists therefore of two modules: the OSPF Control Component (OCC) and the OSPF Link Component (OLC). The first one operates at the control card level and the second one operates at the line card level. It is essentially a full OSPF distribution where only some configuration and management functionalities remain centralized at the control card level.

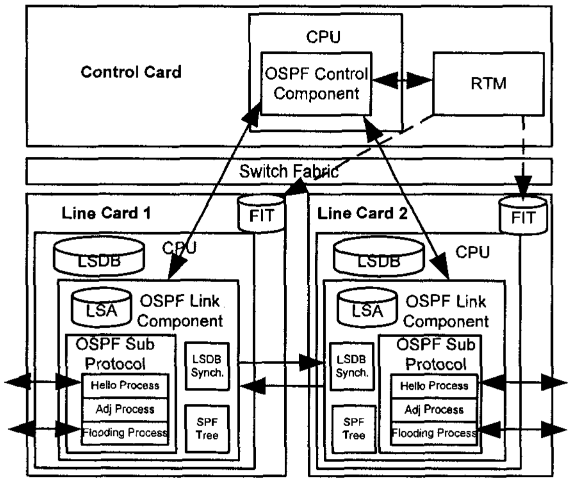


Figure 5-6: Distributed OSPF Architecture

The OCC module performs the following tasks:

- Maintain a global view of the network topology and select the proxy for each domain,
- Update best routes to RTM (Routing Table Manager),
- Interact with the internal world of the router via IRP (Internal Routing Protocol),
- Interface with user and other modules (e.g., MPLS, QoS).

The OLC handles most of the control tasks, including:

- Send and receive OSPF packets (through the interface ports). This is achieved by a Hello Process.
- Run Hello protocol in order to establish the adjacency with its neighbors. This is achieved by an Adjacency Process.
- Synchronize its LSDB (Link State Database) topology database with other OLCs in the same domain. This is done by a LSDB Synchronization component. This component is added to deal with the synchronization issue which appears in the distributed architecture. Whenever a new link state message comes in, it is forwarded to the proxy line card of the corresponding domain. The proxy line card updates the link state database and then re-computes the best routes of the domain. The LSDB Synchronization is then used to update the LSDBs of the other line cards in the domain.
- Flood LSAs (Link State Advertisement) to the external world. This is achieved by a Flooding Process.
- Run SPF (Shortest Path First) if it is assigned as a proxy. This is achieved by a SPF Tree component.
- Generate LSAs from an LSA table.

The distribution of OSPF we propose makes the router much more robust, scalable and resilient [Nguy07b]. As discussed in Section 5.1.3, it is more robust, because the path computation can be executed on line cards, thus different domains can be processed in parallel. The scalability is also further enhanced because the router resources used by OSPF could be adapted to the amount of the routing traffic of the network. Finally, the overall router resiliency is also improved because a line card failure will not lead to the lost of the adjacency with the neighbors on the other line cards.

The performance evaluation of the proposed distributed OSPF architecture compared to the centralized one, in terms of CPU cycles, memory consumption and messages exchanged, will be presented in Chapter 7 where we study the distribution of routing protocols in the context of the distributed architectures for the Routing Table Manager (RTM).

5.2.2 Case Study for Signaling: A Distributed MPLS/LDP Architecture

In this section, we describe the ability of applying the generic distributed architecture for signaling protocols we proposed, to implement MPLS with LDP as primary signaling protocol for next generation routers. Actually, MPLS support is one of the primary requirements for current core routers [Chao02].

5.2.2.1 Overview of Centralized MPLS Architecture

The traditional IP routing is a hop-by-hop forwarding paradigm. When an IP packet arrives at a router, the router looks at the destination address in the IP header, does a route lookup, and forwards the packet to the next hop. If no route exists, the packet is then dropped. This process is repeated at each hop until the packet reaches its destination.

In a MPLS [Rose01] network, nodes are forwarded hop-by-hop based on a fixed-length label. This label, so called Label Switched Path (LSP), determines the route that the packet will take to the destination. Thus, the routing process is done only in edge routers, so called Label Edge Router (LER), then the packet is simply switched over transit routers, so called Label Switch Router (LSR); in consequence, the forwarding speed is improved. In traditional IP based networks, all packets from a given source to a given destination travel on a best route determined by a routing protocol. Hence the additional services such as VPN are not enabled. In addition, nodes on the best route can become critical points due to the overload, while other nodes in the network can be inefficiently utilized. Based on labels, MPLS provides flow management, traffic engineering, quality of service (QoS), VPNs (Virtual Private Networks) and Any Transport over MPLS (AToM).

A MPLS label is a 20-bit identifier, added to the MPLS data packets to forward them over network. Packets sharing the same forwarding criteria, so called forwarding equivalence class (FEC), e.g., that experience the same delay, carry the same label. Therefore, a LSP is a combination of a FEC and a label. In order to define the LSPs among routers in a network, signaling protocols are deployed, where most used are LDP [Ande01] and RSVP-TE [Awdu01].

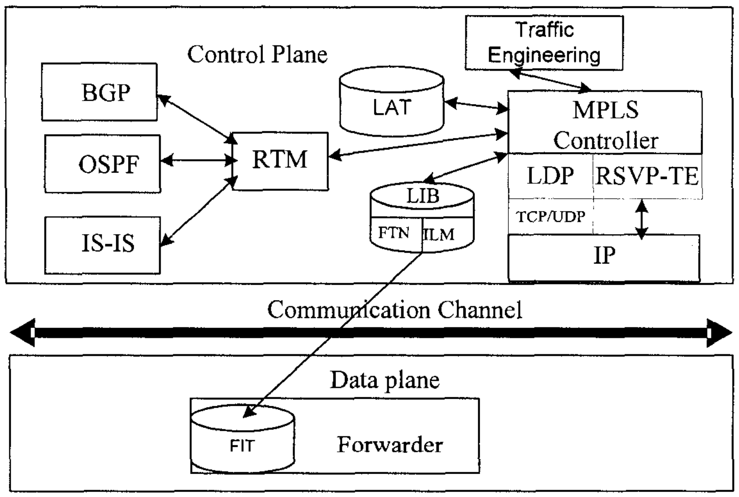


Figure 5-7: MPLS Architecture

In a next generation router, a MPLS architecture consists of two parts (Figure 5-7): data plane and control plane. The data plane is responsible for forwarding MPLS data packet based on information provided by the control plane. Basically, on current routers, data plane is built in the hardware level in order to accelerate the switching speed and the forwarding is achieved by specific chipsets. The control plane, on the other hand, is implemented at the software level and can easily be configured and upgraded. There is a Forwarding Information Table (FIT) on the data plane, that contains the LSPs generated by the signaling protocols located on the control plane. The ingress LER at the incoming edge of the network classifies the IP packets, pushes a label on the data packet that matches a FEC. When a LSR receives a labeled packet, it does a label swapping that consists of:

- Lookup the FIT based on the incoming interface and incoming label,
- Find an appropriate outgoing interface and outgoing label, and
- Replace the incoming label by the outgoing label and send the packet out the outgoing interface.

The egress LER at the outgoing edge of the network will then perform a pop operation to remove the label and restore the original IP packet.

The MPLS control plane has the following components:

- *Signaling protocols*. LDP and RSVP-TE can be used alternatively. They interact with IGPs (Interior Gateway Protocol), like OSPF [Moy98] or IS-IS [ISO02] to compute the path for the ingress LER based on traffic engineering criterion. Traditionally, IGPs provide IP best routes to reach all routers in network using Shortest Path First (SPF) computation. With MPLS traffic engineering facility, IGPs can be invoked to run Constrained SPF (CSPF) to generate path for traffic-engineering (TE) tunnels. The difference between SPF and CSPF is that the latter one takes into account more than one metric, instead of just a single cost for a link between two neighbours as in the previous one. Based on the IGP path calculation, MPLS signaling protocols establish LSPs. The interaction between signaling protocols and IGPs are often achieved through a module called RTM (Routing Table Manager) which gets routing information coming from different routing protocols and selects the overall IP best routes of the system. LDP is located on top of TCP and RSVP-TE use directly the raw socket service provided by IP.

- *Label Allocation Table (LAT).* It contains the available label space of the system. For example, a typical router is provided with 10 interfaces, each one can support at most 100 LSPs, so the number of labels the router can supply is 1000. The LAT table is shared between the signaling protocols in order to avoid ambiguity in the label interpretation.
- *Label Information Base (LIB).* It is a set of tables containing the label mappings. Once a LSP is completed, it is recorded into the FIT located on the data plane. The following tables are required:
    - FEC-To-Next Hop Label Forwarding Entry (NHLFE) Table (FTN). The FTN is used at the LERs for making MPLS forwarding decisions for unlabeled packets. Each entry of the table defines a mapping from a FEC to a NHLFE which contains the instructions for forwarding MPLS packets qualified for a specific flow. The NHLFE includes one or more outgoing

labels, operations on the packet label stack, layer 2 encapsulation, specifications for traffic shaping and policing, the outgoing layer 2 interface.

  - Incoming Label Map Table (ILM). The ILM table is used at the LSRs for making forwarding decisions for labeled packets.

- *Other tables*, such as Multicast Routing Forwarding Table (MRF, used to map a FEC to multiple NHLFEs), Differentiated Services Mapping Table (DSM, used to map Differentiated Services Code Point values to layer 2 interface reservation handles) can also be provided on the MPLS control plane depending on the router capabilities.
- *Traffic engineering*. It takes care of the interfaces with user and the link management module in order to control the bandwidth consumption on LSPs and advertise the neighbors.

In addition, MPLS control plane can also interact with EGPs (Exterior Gateway Protocol), like BGP, in order to distribute labels in the network using protocol extensions [Rekh01].

The next section focuses on the MPLS control plane with LDP as principal signaling protocol to distribute labels in the network.

5.2.2.2 Towards A Distributed MPLS/LDP Architecture for Next Generation Routers

Basically, distributed MPLS/LDP architectures are RFC compliant. All functions provided by a distributed architecture are the same ones in the centralized architecture. However, while these functions are processed only at the control card in the centralized architecture, in the distributed architecture they are achieved at different locations of the router.

The underlying idea of our research is the distribution of some of the MPLS control functions on line cards, as presented in [Ngu07a] and [Ngu07b]. It is leveraged by the capacity of the multi-purpose CPU and additional memory available on the line cards of the next generation routers which are able to perform complex operations. The switch fabric is also enhanced allowing multiple flows to be transferred among router cards. There are basically two approaches for control plane distribution. The first one is based on the sharing among the control cards. Each control card performs a part of the routing protocol, or covers a part of the network [Ngu07a]. The drawback is that failures on a control card will lead to a partial service interruption. Even if backup control cards are added, extensions are required to conventional routing protocols in order to ensure 100% resiliency or current sessions will be restarted. In addition, the control card is still a potential congestion point. In this section, we focus on the second approach that migrates some control functions to the line cards. This can deal efficiently with both the resiliency and scalability. The control cards overload can be reduced considerably if some processing tasks are released. The congestion hence can be avoided and a control card can serve more line cards. If the protocol processing is achieved at the line card level, current sessions can be maintained for a while in case of control card failures waiting for the control card to be restarted, making failures transparent. Thus, the resiliency is improved. In addition, some failures can be recovered in a faster way from the line card level.

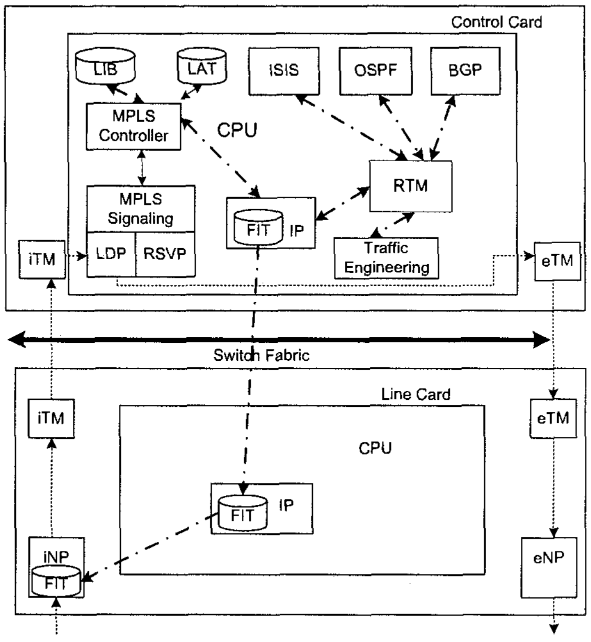


Figure 5-8: Centralized Architecture of MPLS /LDP

The current centralized MPLS implementation is illustrated in Figure 5-8. As we can see, the data plane is entirely implemented on the line cards. Any MPLS data packet is first processed at the iNP of the ingress line card (i.e., the line card that receives incoming data packets) which determines the egress line card (i.e., the line card that sends out data packets) and the outgoing port based on the FIT. Labeling operations (i.e., pushing, popping and swapping) are all achieved by this iNP. The egress line card does only data forwarding. The MPLS control plane, in other words, is implemented entirely on the control card. Control messages (i.e., LDP or RSVP-TE messages) are filtered by the iNP of the ingress line card and forwarded to the control card for further processing. The MPLS Signaling module is in charge of combining, selecting and unifying information provided by LDP and RSVP-TE so that messages going to the MPLS Controller have the same format. The MPLS Controller achieves the label generation, table management and interfaces with other modules and users. It updates the LSPs to the FIT which contains also the IP best routes of routing protocols handled by a RTM. Messages are processed on the control cards and are then sent to the peers through appropriate line cards. The main advantage of this architecture is that it is easily upgradeable. Line card architecture is simple and does not handle complicated software components. Once a router needs to be upgraded, the control card is replaced or its software is rebuilt. The line cards remain intact during the upgrading procedure.

However, the centralized architecture suffers from several drawbacks related to the scalability and resiliency. Particularly, the table management and protocol processing performed on the control card will slow down the processing speed. Such an architecture cannot be used for very high scalable routers with expected *petabit* switching capacity and thousands of line cards. In addition, the next generation routers are expected to be installed in the core networks in at least few years without having to be upgraded [Chao02]. Thus, the upgradeability is much less concerned than the scalability and resiliency. Highly scalable control ports and software tools [Deca00] can also help to ease the software installation process on all line cards when deploying a distributed architecture.

Towards a distributed MPLS architecture, we aim at improving the following components (Figure 5-9):

- *MPLS data plane.* Although in the centralized architecture, the data plane is implemented on the line cards, the processing task is carried out mostly at the ingress line card. A load sharing between ingress and egress line cards can be considered in the distributed architecture.

- *LAT table*. Access to the LAT can be done at the line card level instead of the control card. Processing speed can also be accelerated if a line card can locally decide to generate labels according to the requests from its peers.
- *LIB table*. The current LIB is located on the control card and contains the overall LSPs of the system. It is then copied to the FIT of each line card. This is an extra overload since some LSPs never go through the given line card (this is because each line card hosts only some hundred LSPs while there can be hundred of thousands LSPs in the whole system). Moving LIB to the line cards in the distributed architecture may accelerate the protocol processing and hardware recording. In addition, the memory requirement of the line card is reduced by including only the required LSPs in the LIB of the line card.

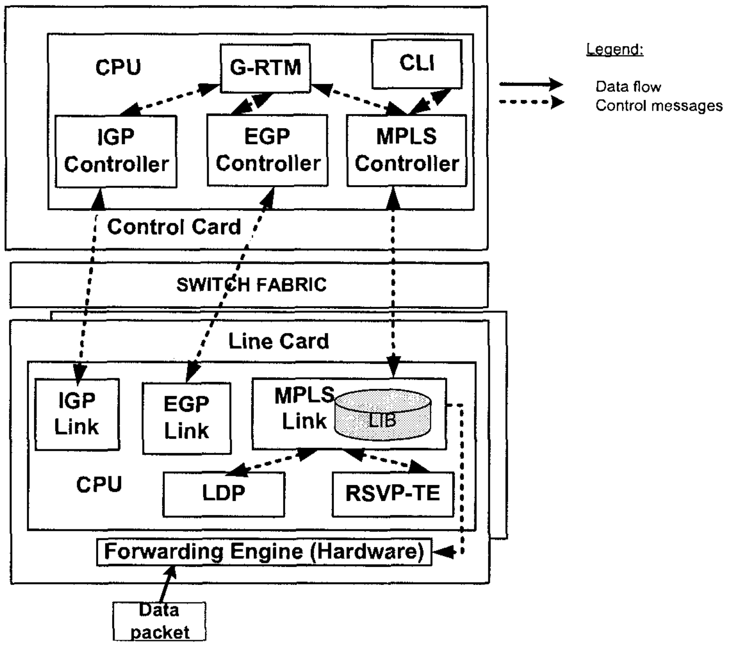


Figure 5-9: Overview of MPLS Distribution Architecture

- *Signaling protocol.* This section deals principally with LDP. In the distributed architecture, the control card is no longer involved in the LDP message processing and transmission. The scalability and resiliency can be improved as discussed above. The LDP adjacency manager is also migrated to the line cards along with LDP. This can also optimize the adjacency tables because the line cards will manage only the neighbors to which it effectively communicate, while the current MPLS Controller on the control card has to manage all adjacencies of the system.

5.3 Chapter Conclusions

This chapter has presented a novel distributed framework for software architectures for the next generation routers, which is able to fully exploit the new hardware features. Considering the requirements of scalability and resiliency, it is a distributed approach where software functions can be shared among router components, namely control cards and line cards.

Such a framework distributes control plane functions onto control and line cards. It includes generic distributed architectures for routing and signaling protocols, distributed and scalable architectures for Routing Table Manager, specific applications for OSPF, MPLS/LDP and RSVP-TE.

The proposed generic distributed architecture for routing and signaling protocols is inspired from a peer-to-peer model. The architecture aims at highly scalable routers with thousands of line cards and *petabit* switching capacity. The validation of the feasibility of such a distributed architecture is also discussed. We described then the case studies to apply the proposed generic distributed architectures for the OSPF and the MPLS/LDP.

The design of the OSPF distributed components are provided, with their functions and interactions.

This chapter also focuses on a distributed architecture for the MPLS because it is one of the most desired features of the next generation routers. We identified the functions and the components of the MPLS module that can be distributed on router cards. Both MPLS data and control plane have been considered, particularly the distribution of tables and the signaling processes.

The next chapter will present the outstanding of this application with a descriptive design for MPLS/LDP.

Chapter 6   A Distributed MPLS/LDP Architecture

In this chapter, we propose a new distributed architecture for MPLS/LDP. Based on the distributed architecture for signaling protocols described in Chapter 4, we develop the mechanisms to achieve MPLS/LDP functions on the components of the router, namely control cards and line cards. In order to define such a distributed architecture, we must address the additional challenges, such as synchronization, consistency between data and control planes, distribution of MPLS labels and restoration of tables in case of failures.

The chapter is organized as follows. In Section 6.1, we review the MPLS/LDP framework introduced by the IETF and present the typical components of a MPLS/LDP architecture for a centralized router architecture. We then describe in Section 6.2 the proposed distributed architecture of MPLS/LDP for the next generation routers. Solutions for overcoming the additional challenges for such a distributed architecture are presented in Section 6.3. We also provide the design of the MPLS/LDP tables that are used for MPLS forwarding in Section 6.4. Finally, we perform some evaluations of the new proposed architecture in terms of CPU resource consumption and the number of exchanged messages.

6.1 Overview of LDP

The Label Distribution Protocol (LDP) [Ande01] is a protocol used for distributing labels in MPLS networks. It is defined by a set of procedures and messages by which Label Switched Routers (LSRs) establish Label Switched Paths (LSPs) through a network by mapping network layer routing information directly to data-link layer switched paths.

An LDP session starts by a discovery process which allows an LDP entity (an LSR) to find a remote LDP peer in the network and to negotiate basic operating procedures between them. The discovery process consists of sending HELLO messages over UDP connections allowing the recognition and identification of adjacent peers. An LDP session is then opened between the two LSRs, and they can proceed to exchange MPLS label binding information. The result of this process is a label switched path (LSP), which constitutes an end-to-end packet transmission pathway between the communicating network devices.

By means of LDP, LSRs can collect, distribute, and release label binding information to other LSRs in a MPLS network, thereby enabling the hop-by-hop forwarding of packets in the network along routed paths. An MPLS/LDP module implemented in a router must be IETF compliant [Rose01, Ande01] and supports a number of features, including: LDP adjacencies, LDP session management, Forwarding Equivalence Classes, Label Generation, Label Distribution Modes, Label Retention Modes, Label Switch Path Control and Loop Detection.

Located on top of TCP, LDP can operate in many modes to fulfill user requirements. The most common usage is unsolicited mode, which sets up a full mesh of tunnels between routers. In solicited mode, the ingress router sends a LABEL REQUEST message to the next hop router, determined from its IP routing table. This request is forwarded through the network hop-by-hop by each router. Once the request reaches the egress router, a return message (LABEL MAPPING) is generated. This message confirms the LSP and informs each router of the label mapping to use on each link for that LSP. In unsolicited mode, the egress routers broadcast LABEL MAPPING messages to all their neighbors. Across each hop, the LABEL MAPPINGs inform the upstream router of the label to use for each link, and by flooding the network they establish LSPs among all routers.

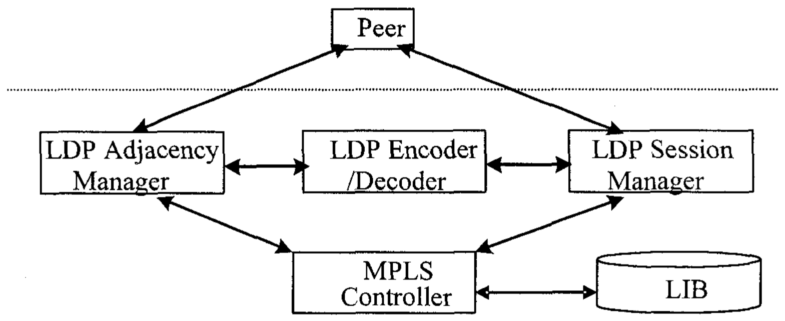


Figure 6-1: Components of LDP and Connection with MPLS

An LDP module includes thee basic components as shown in Figure 6-1.

LDP Adjacency Manager. It sends and receives the HELLO messages to discover LDP neighbors. There are basically two types of adjacencies: Link and Target. Link adjacency is directly connected while Target adjacency is reached through some intermediate nodes. When LDP is enabled on an interface, it sends a HELLO message over the link to discover the peer. If the other peer is also LDP-enabled, it replies and an adjacency is established. Then if an LSP is defined across the two routers, the adjacency will be maintained permanently until the MPLS data transmission is terminated (often by failure of one router). Target adjacencies are manually configured by system administrator.

LDP Encoder/Decoder. It encodes LSP-related requests generated by the MPLS Controller into the LDP messages and passes them to the LDP session manager for forwarding, and decodes incoming LDP messages and passes the appropriate notifications to the MPLS Controller.

LDP Session Manager. It performs main LDP functions such as FEC classification, label generation, label distribution, label retention, label switched path control, loop detection, error processing and notification. The LDP session manager is responsible for: i) opening, accepting, rejecting, and closing the transport (TCP/IP) layer and LDP sessions, ii) notifying the MPLS Controller about newly available LDP sessions and the LDP sessions that have been closed, iii) sending LDP KEEPALIVE messages on idle LDP sessions in order to maintain the connection, iv) processing incoming KEEPALIVE messages, v) packing messages generated by the LDP Encoder/Decoder (such as LABEL REQUEST, LABEL MAPPING, etc.) into the LDP PDUs and sending them over appropriate LDP sessions, and vi) receiving incoming LDP PDUs and passing the LDP messages up to the LDP Encoder/Decoder.

Such a LDP architecture has to process four categories of LDP messages.

- *Discovery messages:* Provide a mechanism in which LSRs indicate their presence in a network by sending HELLO messages periodically. Discovery messages include the LDP Link HELLO message and the LDP Target HELLO message.
- *Session messages:* Establish, maintain, and disconnect sessions between LDP peers. Session messages are LDP INITIALIZATION messages and KEEPALIVE messages.

- *Advertisement messages:* Create, update, and delete label mappings. All LDP ADDRESS messages and LDP LABEL messages belong to advertisement messages.
- *Notification messages:* Provide advisory information and signal error information to LDP peers.

Each LDP message is encapsulated in a so called LDP Protocol Data Unit (PDU) or LDP packet. Figure 6-2 (a) and (b) illustrate the structure of LDP packets. Each LDP packet is made of an LDP header followed by one or more LDP messages. All LDP messages have a common LDP message header followed by one or more structured parameters that use a type, length, value (TLV) encoding scheme. The Value field of a TLV might consist of one or more sub-TLVs.

Except for discovery messages that use UDP as the underlying transport, LDP messages rely on TCP to ensure reliable and in-order delivery of messages. All LDP messages have the format that is depicted in Figure 6-2 (c).

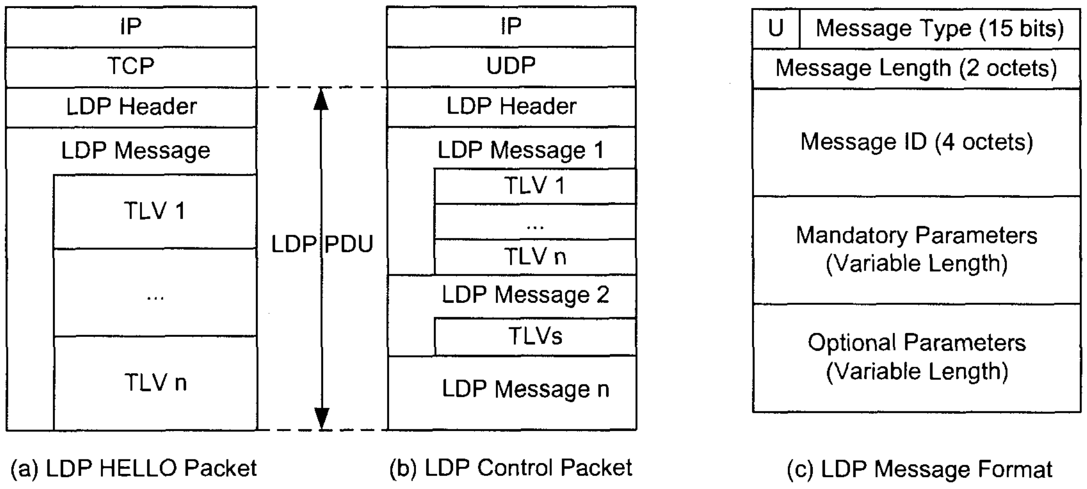


Figure 6-2: LDP Packet and Message Structures

6.2 LDP Architecture

From the top view, in order to achieve the LDP functions described in RFCs [Ande01], a MPLS/LDP implementation architecture has four sub-functions:

- Label and path computation and processing,
- Message sending/receiving and processing,
- Table storage management,
- Interfaces with other modules, such as RTM, QoS, IP stack, CLI, etc.

In a centralized MPLS architecture, the MPLS control plane is primary handled at the level of the control card. Line cards perform only the input/output data exchanges and are not significantly involved in processing of control messages or packets. When the number of LDP sessions is increasing, leading to a large number of routes and labels to process, CPU resources of the control card must be reserved mainly for the LDP processing, taking the precious CPU resources from other modules. The current centralized approach therefore leads to a control card overloading when the traffic increases. This problem can be dealt with an appropriate distribution approach, where a distribution of tasks on the control cards and line cards should be taken into account. Since the control card plays the role of central processing unit for the whole router, and the number of line cards can vary according to different router specifications and system configuration, a client-server model can be effectively deployed. The ultimate objective is to reduce the actual load on the control cards to a minimum.

The MPLS/LDP distributed architecture we propose is based on the distribution of the MPLS control plane, regarding the processing power and memory capacity of line cards in next generation routers. The architecture highlights the following features:

MPLS related functions are improved.

- *MPLS forwarder*. This function, implemented on line cards, is modified so that the MPLS swapping operation can be shared between ingress and egress line cards.
- *MPLS/IP interactions*. This function is redesigned to be performed locally on line cards. Thus the number of messages going through the switch fabric is reduced. In the centralized architecture, the MPLS/IP interactions are done with the help of a global RTM located on the control card. We propose then to do it through the FIT (Forwarding Information Table) instead, located on the line cards.
- *Label provisioning*. In the centralized architecture, new labels are provided at the control card level by a LAT table. Access to the LAT is done through the MPLS Controller. In our proposed architecture, the LAT table is divided into segments, in order to distribute the global label space into the line cards. Therefore each line card is able to make label allocations independently without having to exchange with the control card.
- *MPLS table management*. We allow the MPLS tables (such as LIB – Label Information Base) to be managed on the line cards. This enables MPLS data forwarding decisions to be done locally on the line cards without having to go through the control card.

LDP related functions are also re-designed.

- *LDP module architecture*. The LDP module is migrated entirely into line cards. That way, the control card will never be involved in the processing of received

LDP control messages or packets, neither in the transmission of LDP control messages.

- *Adjacency management.* Our proposed architecture allows the LDP adjacencies to be managed at the line card level. An adjacency table is attached to the LDP module.
- *LDP task sharing.* We balance the LDP message processing task between the ingress line card and egress line card. In the centralized architecture of MPLS/LDP, this task is performed only by the control card. In the new proposed architecture, the LABEL REQUEST processing tasks, including next hop and FEC lookups, will be processed mainly by the ingress line card (upstream line card), while the LABEL MAPPING processing tasks, including label allocation and LABEL REQUEST matching will be processed mainly by the egress line card. A synchronization mechanism between line cards is provided.

We consider below (Figure 6-3) each of the key LDP elements and point out what is modified or not modified in the LDP distributed framework. Whenever it is modified, we mention which reason motivated this modification.

- *LDP Message Encoder/Decoder.* This component is simply moved from the control card to line cards with some little modifications on its interfaces so that it can process messages coming from modules running locally on line cards. The LDP Message Encoder/Decoder creates an LDP message from LSP-related information (e.g., FEC, labels) and decodes an LDP message and passes the appropriate information to the needed modules.

- *LDP Adjacency Manager*. This component is moved from the control card to line cards with some modifications. In the centralized architecture, adjacencies are established by the LDP Adjacency Manager and managed by the MPLS Controller. In our distributed architecture, the LDP Adjacency Manager maintains a table to manage the adjacencies by itself. This allows the adjacencies to be established according to the need of each line card. Components related to the LDP Adjacency Manager include:
    - *Hello Sender*. This process periodically sends HELLO messages on the multicast address over all the physical interfaces. These messages are used to discover peers on the network.
    - *Hello Processor*. This process listens on the multicast address for HELLO messages. HELLO messages are processed to determine the capabilities of the peer. If a HELLO message is acceptable, the message sender is saved in the Adjacency Table to be used later for establishing LDP sessions.
    - *Adjacency Table*. This table contains information about neighbors that are MPLS/LDP enabled routers. There are two types of neighbors:
        - Direct neighbor (or physical connected neighbor), which is discovered by Basic Discovery HELLO messages (sent over UDP)
        - Targeted neighbor (or non-physical connected neighbor), which is discovered by Extended Discovery HELLO messages (sent over TCP)

- *LDP Session Manager.* This component is moved from the control card to line cards with modifications. In the centralized architecture, the LDP Session Manager processes all the LDP sessions of the system and passes information to the MPLS Controller. As the LDP is distributed on the line cards, we add some functions to the LDP Session Managers, allowing them to communicate with each other on different line cards in order to establish the LSP through the line cards. Components related to the LDP Adjacency Manager include:
    - *LIB-Process.* This is a central process assuming the LDP session management. Since an LSP can go through two line cards, the LIB processes can interact with each other through the DS (Distributed Service) in order to achieve LDP negotiations with upstream and downstream routers. The LIB process contributes in the following way:
        - Establishes labels for every MPLS enabled router to which the line card has a connection (physical or non-physical connection).
        - Opens session with the LDP peer and creates an independent session process to manage all the message exchange between the two peers. This session communicates with LIB Process periodically.
        - Writes new entries to the LIB using access services provided by the MPLS Signaling.
    - *Wait-process.* This process waits for requests from peers and starts a new session for each remote peer. When a session is established, the Wait-Process calls the LIB-Process to handle message exchanges.

- *L-LAT (local LAT).* This is a new component which does not appear in the centralized architecture. The L-LAT, managed by the MPLS Signaling component on the line card, contains the label space provided for each line card. When the local label space is not empty, the line card can decide to allocate new labels by itself in response to label requests coming from remote peers. A pool of labels is initially attributed to each line card. When the L-LAT runs out of labels, it may send a request to the G-LAT asking for a new provision. The initial local label space can be configured by system administrators through interfaces with the MPLS Controller running on the control card.
- *LIB.* This component is moved from the control card to the line cards with modification. In the centralized architecture, the LIB, managed by the MPLS Controller, contains all the LSPs of the system, while in the proposed distributed architecture, it contains only the LSPs traversing the line card where it is located. In addition, the LIB in the proposed architecture is managed by the MPLS Signaling and can be accessed by signaling protocols such as LDP. There are two types of LSPs contained in the LIB:
    - *Complete LSPs*: the LSPs that have been established and are used to forward MPLS data packets. Completed LSPs are recorded into the hardware network processor.
    - *Incomplete LSPs*: the LSPs that are being built by signaling protocols.

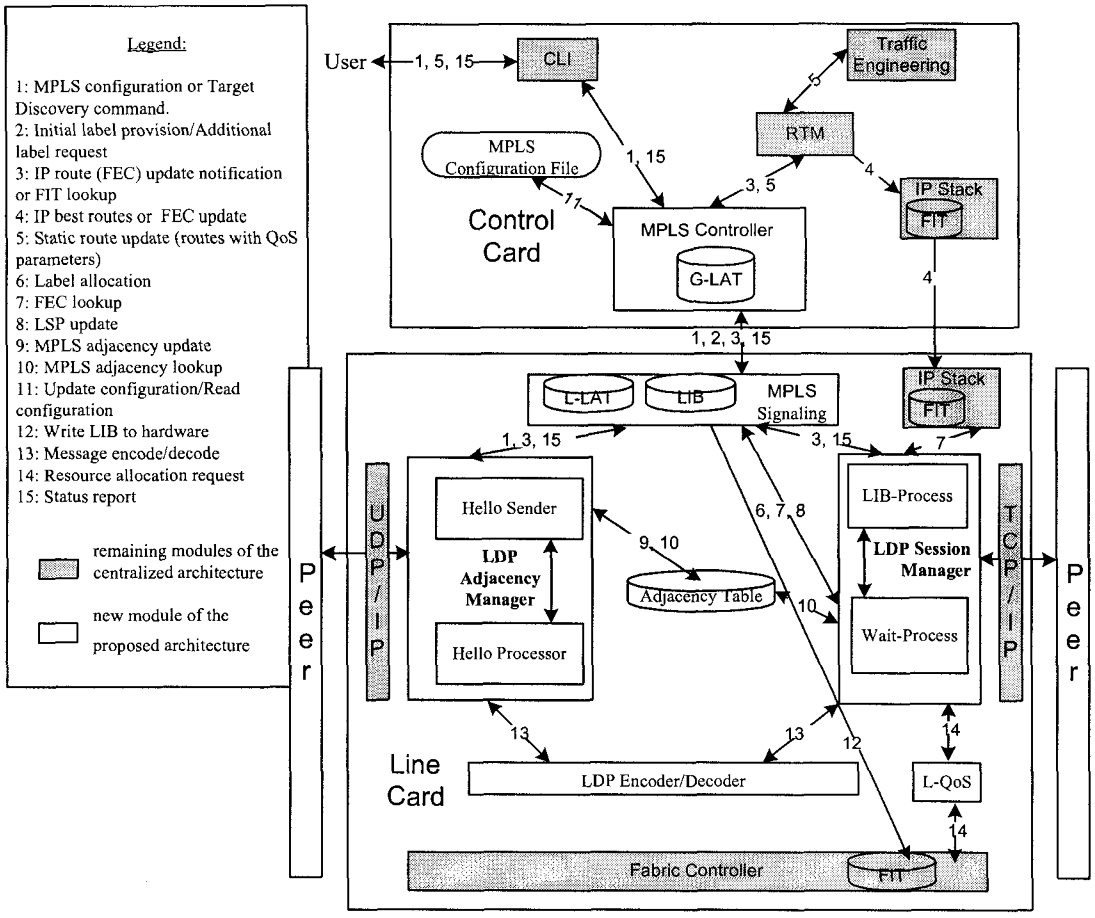

Legend:
1: MPLS configuration or Target Discovery command.
2: Initial label provision/Additional label request
3: IP route (FEC) update notification or FIT lookup
4: IP best routes or FEC update
5: Static route update (routes with QoS parameters)
6: Label allocation
7: FEC lookup
8: LSP update
9: MPLS adjacency update
10: MPLS adjacency lookup
11: Update configuration/Read configuration
12: Write LIB to hardware
13: Message encode/decode
14: Resource allocation request
15: Status report
remaining modules of the centralized architecture
new module of the proposed architecture
User
CLI
Traffic Engineering
RTM
MPLS Configuration File
Control Card
MPLS Controller
G-LAT
IP Stack
FIT
L-LAT
LIB
MPLS Signaling
Hello Sender
LDP Adjacency Manager
Hello Processor
Adjacency Table
LIB-Process
LDP Session Manager
Wait-Process
UDP/IP
TCP/IP
Peer
Peer
Line Card
LDP Encoder/Decoder
L-QoS
Fabric Controller

Figure 6-3: Distributed MPLS/LDP Architecture

The distributed architecture we propose for MPLS/LDP is indeed derived from the general distributed framework of signaling protocols for next generation routers [Ngu07b]. We summarize it in Figure 6-3. In our architecture, the MPLS Signaling module, as well as LDP and RSVP-TE are moved entirely on the line card. The MPLS Controller remains on the control card as in the centralized architecture, but it is much simplified. It still handles the connections with other modules, such as RTM and CLI. The RTM manages the IP routes learnt by routing protocols, combined with the static routes provided by users. These routes are used to specify the FECs (*Forwarding*

Equivalent Class). Basically, the MPLS can get FECs directly from the FIT managed by the IP Stack. However, the RTM is required from time to time to inform the MPLS about the change of the existing FECs. There is a configuration file that contains specific system settings. System administrators may change the configuration through the CLI and the MPLS Controller keeps these running parameters in the configuration file. The MPLS Controller manages a Global LAT (G-LAT), which contains the overall label space of the system. Each line card handles a segment of the G-LAT, called Local LAT (L-LAT), which is used exclusively for the traffic going through the local interfaces of the line card. The L-LAT is maintained by the MPLS Signaling. When there is a label request, the signaling protocol asks the MPLS Signaling for a new label and the MPLS Signaling provides a label in the L-LAT. When the L-LAT runs out of labels, the MPLS Signaling sends a request to the MPLS Controller to get additional labels from the G-LAT. Unused labels of L-LATs may be returned periodically to the G-LAT. For example, due to little traffic, there may still be many free labels in the initial label pool provided to the L-LAT of a given line card after a period of time. In such a case, the MPLS Signaling will send back some labels to the G-LAT, to allow their use by other line cards that are in need of labels.

The LIB is migrated to line cards and is managed by the MPLS Signaling, which controls accesses from signaling protocols. In order to update the FIT of the network processors (NP), MPLS routes and IP best routes are recorded into the CAM (*Content Access Memory*) respectively by the MPLS Signaling and the IP Stack through an interface called the Fabric Controller. The FIT content is used by the NPs to forward IP and MPLS data packets.

The Hello Sender process within the LDP Adjacency Manager periodically sends HELLO messages on the multicast address over all the local physical interfaces of the line card in order to discover peers on the network. It also sends Target HELLO messages ordered by the MPLS Signaling module to indirect destinations according to user requests. The Hello Processor listens on the multicast address for incoming HELLO messages, then processes them to establish the adjacency. The LIB-Process within the LDP Session Manager is responsible for LDP session management. It establishes LDP connections to all Link and Target adjacencies in the Adjacency Table. Since a LSP can go through two line cards, the LIB processes can also interact with each other located on a different line card through the DS in order to achieve LDP negotiations with upstream and downstream routers. If traffic engineering and Constrained-based Routing LDP (CR-LDP) are deployed, the LIB process can interact with a local QoS module (L-QoS) located on the line card, which manages the available bandwidth of all local interfaces. However, CR-LDP mechanisms for resource allocation are not addressed in this thesis. Indeed, the same function can be provided efficiently by RSVP-TE.

As the LDP module is implemented entirely on the line cards, the architecture we propose has the following advantages.

- *Accelerate the HELLO sending rate.* Because the HELLO sending process is triggered locally at the line card level instead of the control card level, HELLO messages go faster to reach neighbors. In the centralized architecture where the MPLS signaling and LDP module run on the control card, HELLO sending process can even be queued due to congestion on the control card.

- *Save control card resource used for HELLO processing.* In the centralized architecture, HELLO process is handled by the control card, leading to control card resource consumption and low speed processing. It becomes critical when the number of line cards is increasing.
- *Manage the LDP adjacencies efficiently.* The Adjacency Table is created and maintained locally by the MPLS module running on the line cards. It contains only the neighbors the line card connects to. In the centralized architecture, the control card has to maintain a large-size table containing all adjacencies of the whole system. The lookup operation is therefore accelerated in the proposed architecture.
- *Reduce the size of the LIB table.* The LIB in the proposed architecture is optimized because it contains only the LSPs traversing the line card. In the centralized architecture, the LIB located on the control card must contain all the LSPs going through the router.
- *Improve the MPLS/IP interaction.* LDP processes can use directly the local FIT located on the line cards to set up the FEC and LSP instead of using the RTM located on the control card.
- *Make the architecture more scalable.* First, in the proposed architecture, LDP sessions are handled in parallel by the line cards. In the centralized architecture, they have to come in to a queue waiting for control card processing resources. Second, LDP message processing, particularly the message encoding/decoding, is performed entirely at line card level so the scalability is increased.

6.3 Challenges and Their Solutions

Such an MPLS/LDP distributed architecture requires solving some new challenges which correspond to issues that do not appear in the centralized architecture.

Synchronization among line cards. In a centralized architecture, all messages are forwarded to the MPLS Controller on the control card. In the proposed architecture, messages from the upstream router (*Ru*) and downstream router (*Rd*) are processed separately. The upstream messages are handled by the LDP module on the ingress line card while the downstream messages are taken care of by the egress line card. In order to establish an LSP between the upstream and downstream routers, the ingress and egress line cards have to pair their messages to their respective peer messages.

Label provisioning. In a centralized architecture, all labels are generated by the MPLS Controller, which is aware of all the LSPs of the system. In the proposed distributed architecture, each line card manages only a smaller set of LSPs and labels are generated locally on line cards. Therefore, mechanisms should be provided to ensure that data coming from different ingress line cards have the same label if they are qualified to be sent out by the same egress line card.

Restoration of the LIB on a line card if there is an error on the line card. In a centralized architecture, there is no MPLS software component on line cards. Therefore, if a line card fails, its forwarding table is simply reloaded from the control card. In the proposed architecture, each line card hosts a local LIB. Therefore if there is no backup mechanism, data will be lost.

Some other challenges such as table management, selecting line cards to perform user's requests, classifying IP packets into FECs, handling socket errors, are common for both centralized and proposed architectures, so these are not considered in this section.

We next describe the solutions we propose to overcome these challenges.

6.3.1 Synchronization Mechanisms

In the traditional architecture where the LDP process runs on the control card, synchronization is not an issue. The control card manages all tables, which are all global; and it also handles the message processing. A queuing mechanism is implemented in order to regulate the access to the control card. Concurrent access to the tables can be solved by a simple locking mechanism.

However, due to the LDP distribution on line cards, LDP messages are processed differently on the line cards. Basically, we distinguish the ingress line card and the egress line card that are involved in a LSP. The ingress line card maintains the connection with the upstream router (Ru) while the egress line card is connected to the downstream router (Rd) (Figure 6-4). Since a line card has in general more than one port, it can be the ingress for a given LSP and the egress for other LSPs.

In Figure 6-4, we can see that the LDP modules on the ingress and egress line cards may receive different messages from the upstream and downstream routers, at different times. However, the final information about the LSP to be built between the *Ru* and *Rd* must be the same in the two LIBs. Otherwise, the MPLS forwarding can not be done. This requirement leads to the need of synchronization between the ingress and egress line cards.

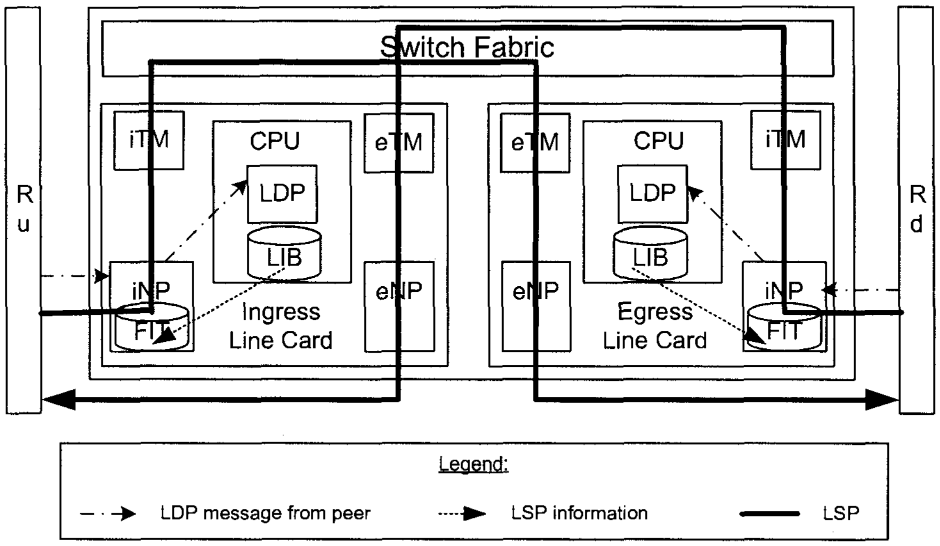


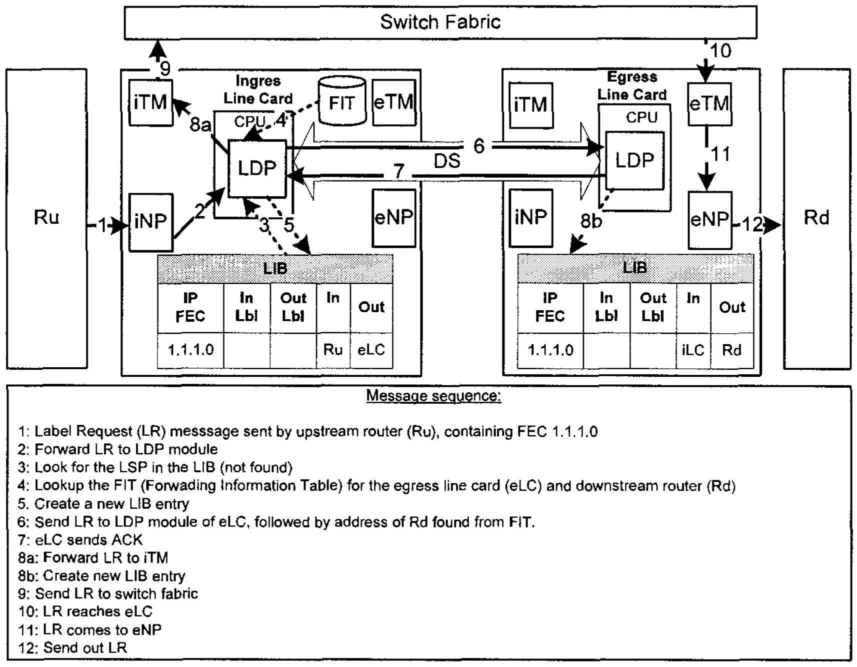
Figure 6-4: Ingress and Egress Line Card of an LSP

In addition, the content of the LIB should be consistent with the routing tables. That is, whenever an IP route (and its corresponding FEC) is changed, the LSP based on the corresponding FEC must be updated. This requirement leads to the need for synchronization between the LIB managed by the LDP modules and the routing table managed by the RTM.

Synchronization between Ingress and Egress Line Cards

In order to deal with the synchronization issue, we share information of each LSP on both the ingress and egress line card, as shown in Figure 6-5 and Figure 6-6. These two figures consider a case of transit routers (LSR). A similar synchronization mechanism for edge routers can be used for edge routers where the IP FEC is mapped to MPLS labels. Details of the synchronization algorithms for both LSR and LER can be found in [Nguy06b].

The fields of the LIB are IP FEC, Incoming Label, Outgoing Label, Incoming Interface and Outgoing Interface. The IP FEC is used as a key field for the table. The incoming label is the label the router provides to the upstream router and the outgoing label is the label supplied by the downstream router, assuming we are using the solicited mode. On the ingress line card, the incoming interface is the port connecting to the upstream router and the outgoing interface is the egress line card. On the egress line card, the incoming interface is the ingress line card and the outgoing interface is the port connecting to the downstream router.



IP FEC	In Lbl	Out Lbl	In	Out
1.1.1.0			Ru	eLC

IP FEC	In Lbl	Out Lbl	In	Out
1.1.1.0			iLC	Rd

Message sequence:

1: Label Request (LR) messsage sent by upstream router (Ru), containing FEC 1.1.1.0
2: Forward LR to LDP module
3: Look for the LSP in the LIB (not found)
4: Lookup the FIT (Forwading Information Table) for the egress line card (eLC) and downstream router (Rd)
5. Create a new LIB entry
6: Send LR to LDP module of eLC, followed by address of Rd found from FIT.
7: eLC sends ACK
8a: Forward LR to iTM
8b: Create new LIB entry
9: Send LR to switch fabric
10: LR reaches eLC
11: LR comes to eNP
12: Send out LR

Figure 6-5: Label Request Handled by Ingress and Egress Line Card

The idea of the synchronization is to pair the messages received from the upstream and downstream routers so that the LDP processes on the ingress and egress line cards will record the same information to their LIBs. The pairing is achieved through the exchanges between the ingress and egress line cards. We consider the mechanism for the two most important messages of the LDP protocol: LABEL REQUEST and LABEL MAPPING. The first message is used for requesting a label-FEC binding and the latter is the reply.

Figure 6-5 and Figure 6-6 illustrate the pairing of these two messages for one LSP on the ingress and egress line cards of an LSR.

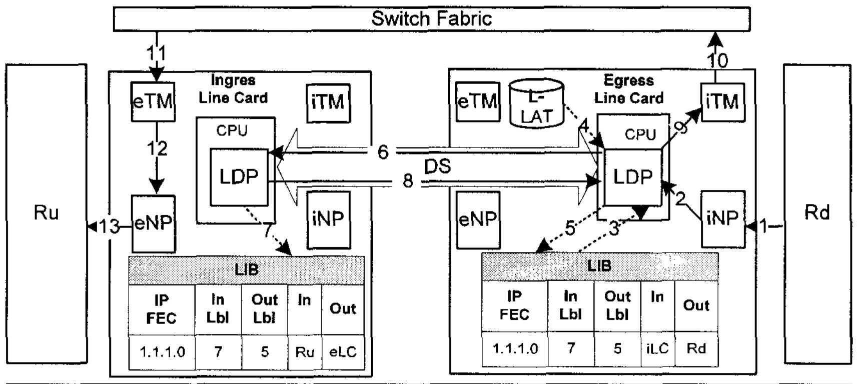
Figure 6-5 shows the processing of a LABEL REQUEST message sent from the upstream router. The LABEL REQUEST message is sent by a remote peer (1). It is filtered by the local iNP and sent to the LDP process (2). The LDP process looks up the local LIB to determine whether an entry with the same FEC is already there (3). The searching key is the FEC contained in the LABEL REQUEST message.

- If an entry is found with all the label fields completed, the ingress line card replies with the label/FEC binding in the LIB.
- If no entry is found, the ingress line card performs the following tasks:
    - Determine the address of the outgoing interface (on the egress line card) by looking up the FIT table with the FEC as the search key (4).
    - Create an entry in the LIB (5), where the fields contain the following values:
        - *FEC*: the FEC that needs a label,
        - *Incoming label*: NULL, this label will be generated when the LABEL MAPPING message arrives in the return path,
        - *Outgoing label*: NULL, this is used to indicate that this entry is incomplete and it is waiting for a LABEL MAPPING reply,
        - *Incoming interface*: address of the sender from which the LABEL REQUEST message is sent, that is the address of the upstream router for the ingress line card or the address of the ingress line card for the egress line card,

        - *Outgoing interface*: the location of the egress line card.
    - Use DS to forward the original LABEL REQUEST message to the corresponding egress line card, followed by the address of next hop found from the FIT (6).
    - After receiving an ACK message from the egress line card (7), forward LABEL REQUEST message to iTM (8a). The message travels the switch fabric (9) in order to reach the downstream router (10), (11), (12).
    - Wait for LABEL MAPPING reply from the egress line card.
- When the egress line card receives the LABEL REQUEST message from the ingress line card sent over DS, it performs the following operations:
    - Send an ACK back to the ingress line card. The ACK message also contains the next hop (7).
    - Create an entry in the LIB (8b), where the fields contain the following values:
        - *FEC*: the FEC that needs a label, this field is used as key to look up the requester in the return path,
        - *Incoming label*: NULL, this label will be generated when the LABEL MAPPING message arrives in the return path,
        - *Outgoing label*: NULL, this is used to indicate that this entry is incomplete and it is waiting for LABEL MAPPING reply,
        - *Incoming interface*: address of the ingress line card,

  - *Outgoing interface*: the address of the downstream router, which is provided by the ingress line card following the LABEL REQUEST message.
- o Wait for LABEL MAPPING reply from the downstream router. The behavior of line cards when receiving the LABEL MAPPING is described in Figure 6-6.

The return path of the LABEL MAPPING message is shown in Figure 6-6. Upon receiving a LABEL MAPPING message (1-2) corresponding to the previous LABEL REQUEST, the egress line card looks up the local LIB (3) to determine if there is an entry waiting for this return mapping message. The search key is the FEC contained in the LABEL MAPPING message.



Message sequence:

1: Label Mapping (LM) message sent by downstream router (Rd), containing FEC 1.1.1.0 and label 5
2: Forward LM to LDP module
3: Look for the LSP in the LIB (found)
4: Generate new Label 7 from L-LAT (Local - Label Allocation Table)
5: Update the LIB entry with new label 5 (from LM) and label 7 (new generated)
6: Forward LM to LDP module of iLC, followed by new label 7
7: Lookup LIB for an entry (FEC as key) and update label fields by label 7 and label 5
8: iLC sends ACK
9: Replace label 5 in the LM by label 7 then forward LM to iTM
10: Send LM to SF
11: LM reaches eLC
12: LM comes to eNP
13: Send out LM

Figure 6-6: Processing LDP Mapping for Previous LDP Request (Solicited Mode)

- If an entry is found (with the label fields still empty), the egress line card:

    - Generates a new label (taken from local L-LAT) (4),
    - Updates the entry; the outgoing label field is filled with the label contained in the LABEL MAPPING message, and the incoming label field is filled with the new generated label (5),
    - Uses DS to forward the original LABEL MAPPING message to the corresponding ingress line card, followed by the new label the egress line card has generated. The address of the ingress line card is found in the entry of the LIB (6),
    - After receiving an ACK message from the ingress line card (8), the egress line card replaces the label in the original LABEL MAPPING message with the new generated label, then forwards LABEL MAPPING message through the iTM to the upstream router (9).
- If no entry is found, the egress line card proceeds as if the LABEL MAPPING message had been sent in the Unsolicited mode. Hence it:
    - Creates a new entry in the LIB table.
    - Looks up the FIT for the ingress line card(s). It is in fact a reverse lookup.
    - Generates a new label (taken from local L-LAT).
    - Fills in the new entry with the new label (to the incoming label field), original label (to the outgoing label field) and FEC (to the FEC field) in the LABEL MAPPING message, address of the ingress line card (to the incoming interface field) and address of the router from which the LABEL MAPPING message comes (to the outgoing interface field).

    - Uses DS to forward the original LABEL MAPPING message to the corresponding ingress line card(s), followed by the new label the egress line card has generated.
    - After receiving an ACK message from the ingress line card, the egress line card replaces the label in the original LABEL MAPPING message by the new generated label, then forwards the LABEL MAPPING message through the iTM to the upstream router.

When the ingress line card receives a LABEL MAPPING message from the egress line card sent over DS, followed by a label, it looks up the local LIB to determine if there is an entry waiting for this mapping previously. The search key is the FEC contained in the LABEL MAPPING message.

- If an entry is found (where the label fields are empty), the ingress line card:
    - Updates the entry (7); the outgoing label field is filled with the label contained in the LABEL MAPPING message, and the incoming label field is filled with the second label.
    - Sends an ACK back to egress line card (8).
- If no entry is found, the ingress line card considers that the LABEL MAPPING message has been sent during the Unsolicited mode. It:
    - Creates a new entry in the LIB table,
    - Looks up the FIT for the upstream router. It is in fact a reverse lookup,
    - Fills in the new entry with the second label (to the incoming label field), original label (to the outgoing label field) and FEC (to the FEC field) in the LABEL MAPPING message, address of the egress line card (to the

incoming interface field) and address of the router from which the LABEL MAPPING message comes (to the outgoing interface field),

  - Sends an ACK back to the egress line card.

The LABEL MAPPING will then be forwarded to the upstream router.

The synchronization issue is solved by the mechanism described above, because:

Information of the LSP is consistent in the LIBs of the ingress and egress line cards. Indeed, the fields of the two LIBs are the same with the help of messages exchanged between the two line cards. The IP FEC is determined on the ingress line card and then sent to the egress line card. The Incoming Label is generated on the egress line card and then sent to the ingress line card. The Outgoing Label is received by the egress line card from the downstream router and then sent to the ingress line card. Finally the Outgoing Interface is determined by the ingress line card and provided to the egress line card.

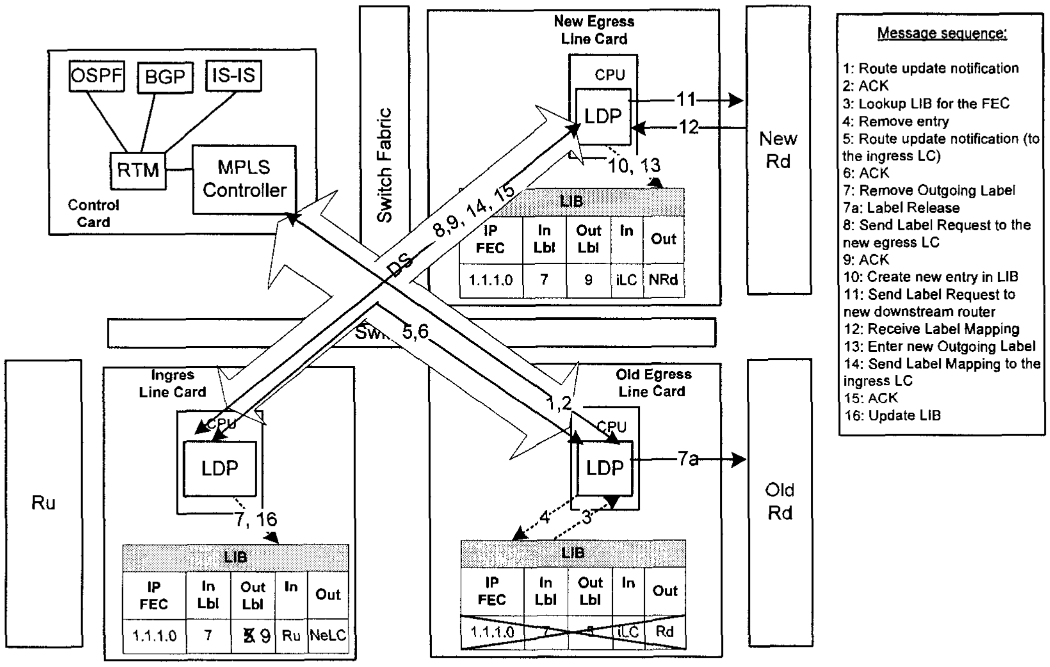
Data transmission is triggered only when the LSP is completed on both the ingress and egress line cards. When the LIB fields are not filled with the desired information, the LSP is not yet recorded in the FIT. The acknowledgement mechanism is implemented to make sure that information provided by each LDP message is first saved in the LIBs of both ingress and egress line cards, then the message is forwarded to the corresponding peer. As shown in the Figure 6-6, the upstream router may start sending data only after receiving the LABEL MAPPING message from the egress line card. The egress line card sends this message when it has received the acknowledgement from the ingress line card. This acknowledgement indicates that all LIB fields are completed so the router is ready for MPLS data switching.

In addition, in the proposed architecture, LABEL REQUEST (or LABEL MAPPING) messages are processed independently, meaning that the LDP module can accept new REQUEST (or MAPPING) messages while waiting for ACK messages from other line cards.

Synchronization between MPLS LIB and Routing Table

In the proposed architecture, the LDP gets information about the FECs from the FIT, which is located on the line cards. The FIT content is updated by the RTM based on routing information from routing protocols, such as OSPF or BGP. The issue is that, whenever a FEC is changed, the corresponding LSP, which has been established and recorded to the LIB, must be rebuilt. For example, routing protocols determine that the best route to a given destination is changed. In such a case, the downstream router of the LSP leading to that destination needs to be changed according to the new best route.

This issue is dealt with the help of the MPLS Controller. In the proposed architecture, the MPLS Controller is located on the control card and has an interface with the RTM. By implementing a "trap" function, the MPLS Controller can be informed of any change in the routing table managed by the RTM. Thus, it will be notified whenever the RTM updates the FITs on the line cards. If this update is related to an LSP that is already built, a synchronization mechanism should be launched. Since an IP route change results in a new next hop for a given FEC, we keep the ingress line card of the corresponding LSP (which connects to the upstream router) and change the egress line card (which connects to the downstream router). Therefore, the synchronization mechanism works as follows (Figure 6-7).



OSPF
BGP
IS-IS
RTM
MPLS Controller
Control Card
Switch Fabric
New Egress Line Card
CPU
LDP
11
12
10, 13
LIB
IP FEC | In Lbl | Out Lbl | In | Out
1.1.1.0 | 7 | 9 | iLC | NRd
New Rd
8,9, 14, 15
DS
Sw 5,6
1,2
Ingres Line Card
CPU
LDP
7, 16
LIB
IP FEC | In Lbl | Out Lbl | In | Out
1.1.1.0 | 7 | 8 9 | Ru | NeLC
Ru
Old Egress Line Card
CPU
LDP
7a
4
3
LIB
IP FEC | In Lbl | Out Lbl | In | Out
1.1.1.0 | 7 | 8 | iLC | Rd
Old Rd
Message sequence:
1: Route update notification
2: ACK
3: Lookup LIB for the FEC
4: Remove entry
5: Route update notification (to the ingress LC)
6: ACK
7: Remove Outgoing Label
7a: Label Release
8: Send Label Request to the new egress LC
9: ACK
10: Create new entry in LIB
11: Send Label Request to new downstream router
12: Receive Label Mapping
13: Enter new Outgoing Label
14: Send Label Mapping to the ingress LC
15: ACK
16: Update LIB

Figure 6-7: Synchronization between IP Routing Table and MPLS LIB

Figure 6-7 shows a synchronization process triggered by an IP route update notification received from the RTM. Using a "trap", the MPLS Controller determines the IP FEC, old next hop, new next hop, old egress line card and new egress line card included in the route update notification message. It then sends the route update notification message to the old egress line card (1). The old egress line card replies with an acknowledgement (2), then looks up its LIB to see whether a LSP with the given IP FEC is there (3). If it is, the old egress line card:

- o Removes the entry (4),
- o Forwards the route update notification message to the corresponding ingress line card (5),

- Sends a LABEL RELEASE message to the old downstream router in order to release the current outgoing label and to stop the current LDP session with the old downstream router (7a).

When the ingress line card receives the route update notification from the old egress line card, it replies with an acknowledgement (6), then:

- Removes the current outgoing label from the corresponding entry in the LIB (7). As the LIB entry is not completed, the current session is temporary halted,
- Creates a LABEL REQUEST, then sends it to the new egress line card whose location is defined in the route update notification, followed by the current incoming label (8).

The new egress line card processes the LABEL REQUEST message from the ingress line card as in the normal case, except that no new label is generated. The new egress line card:

- Replies with an acknowledge message (9),
- Creates a new entry in its LIB for the IP FEC (10), then forwards the LABEL REQUEST to the new downstream router (11),
- Waits for the LABEL MAPPING message from the new downstream router. When this message arrives (12), the new egress line card updates the incomplete entry in the LIB (13) then sends the LABEL MAPPING to the ingress line card (14).

The ingress line card sends back an acknowledgement (15), then updates the incomplete entry in the LIB with the new outgoing label field (16). After receiving the acknowledge message from the ingress line card, the new egress line card records the LIB entry to the FIT to start the data transmission.

Such a mechanism deals efficiently with the MPLS/IP table synchronization issue, because:

The LSP is updated consistently with the routing table. Whenever the routing table is changed, the related LSP is updated through the notification service of the MPLS Controller.

Ingress and egress are informed correctly about the change. The old egress line card is first informed so it stops sending the current data stream to the old downstream router. The ingress line card is then notified to contact the new egress line card. Finally a new LSP is built between the ingress line card and the new egress line card.

Peer routers are informed in order to update LSP. The new downstream router is contacted to establish a new LDP session. The old downstream router is informed to release the label.

In addition, the current upstream router is kept transparent of the change. The old egress line card is switched over a new one without having to shut down the current LDP session with the upstream router. The incoming label is therefore maintained.

6.3.2 Label Provisioning and Data Recovery

In our architecture, the label generation is done by the egress line card instead of the ingress line card. This ensures that packets coming from different sources to the same destination will have the same label. For example, in the case where an LSP has been defined and another ingress line card wants to forward data to the same downstream router, the LABEL REQUEST is sent to the egress line card with a same FEC (because of the same destination). The egress line card looks up its LIB based on the FEC and can immediately return the existing labels without having to contact the downstream router (Figure 6-8).

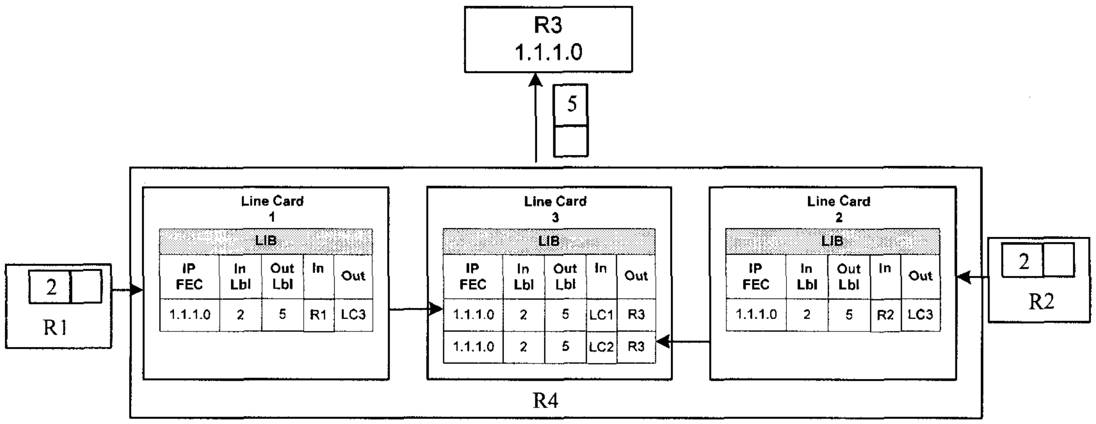

R3
1.1.1.0
5
Line Card 1
LIB
IP FEC | In Lbl | Out Lbl | In | Out
1.1.1.0 | 2 | 5 | R1 | LC3
Line Card 3
LIB
IP FEC | In Lbl | Out Lbl | In | Out
1.1.1.0 | 2 | 5 | LC1 | R3
1.1.1.0 | 2 | 5 | LC2 | R3
Line Card 2
LIB
IP FEC | In Lbl | Out Lbl | In | Out
1.1.1.0 | 2 | 5 | R2 | LC3
2
R1
2
R2
R4

Figure 6-8: Forwarding Multiple Sources to the Same Destination

The resiliency issue is also dealt with efficiently by having the LSP information on the LIBs of both the ingress and egress line cards. We aim at restoring the LIB on failed line cards so that the data transmission can be resumed. As a LSP can go through at most two line cards, we have two scenarios of failures: one line card failed and both line cards failed.

In the case where one of the two LIBs is lost due to an error, information can be restored with the help of the other. The recovering procedure is as follows. The failed line card broadcasts a request to all line cards in the system. Every line card looks up the Incoming Interface and Outgoing Interface fields of their LIBs using the failing line card as search key. Any entry found means that the failing line card is involved in the corresponding LSP. The lost LIB is then restored completely based on these entries.

The case where the two line cards are failed at the same time is hardly ever met. If it does happen, all LDP sessions on the two failed line cards are restarted and all line cards in the system are informed in order to remove the related entries.

For the LSPs going through two ports located on the same line card, which are not much in practice, a traditional backup mechanism with a copy on the control card managed by the MPLS Controller can be deployed. These LSPs are saved at two locations of the router: the LIB of the line card and a table, so called L-ROUTE, on the control card. When the line card is restarted, it sends a request to the MPLS Controller asking for the lost LSPs. The MPLS Controller looks up the L-ROUTE, using the failing line card as the search key. Entries found will be provided to the line card in order to restore its LIB.

6.4 MPLS Data forwarding & Table Management

We now discuss the way the MPLS uses information provided by the LDP in order to forward data packet over networks.

6.4.1 MPLS Tables

In the centralized architecture, MPLS tables are all managed by the control card, putting heavy load on the control card processing resource. In the proposed architecture, these tables are migrated into line cards in order to increase the scalability and efficiency. We now describe the structures of the MPLS tables.

LAT

The LAT (Label Allocation Table) is used to manage label spaces and to track all allocated and ready-to-use labels. The LAT is configured during the initialization of the

MPLS module and is consulted when an LSP is established or removed. In the centralized architecture, only one LAT is handled by the control card that is responsible for all label allocation requests. It is therefore non-scalable and inefficient.

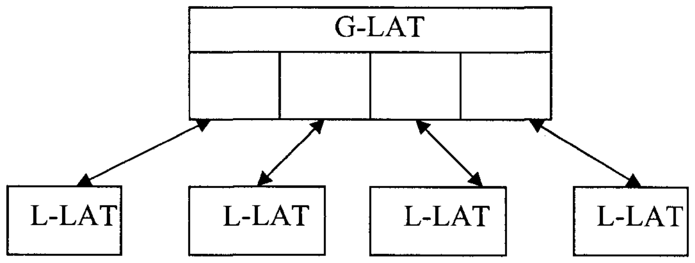


Figure 6-9: LATs in the Distributed Architecture

In our proposed architecture, each line card maintains a local LAT (L-LAT) used exclusively for the traffic traversing its interfaces. The control card keeps a global LAT of the whole system. The G-LAT (Global LAT) is handled by the MPLS Controller. It contains all labels of the system. Labels in the G-LAT are numbered followed the increasing order. The sizes of L-LATs correspond to the label space of each line card, which can be configured by system administrators (Figure 6-9). The L-LAT is managed by the MPLS Signaling module. There is a connection between the MPLS Signaling and the MPLS Controller. When there is a label request, the LDP asks the MPLS Signaling for a new label and the MPLS Signaling allocates a label from the local label space. In the case where there are no local labels left to allocate, the MPLS Signaling sends a request to the MPLS Controller to get additional labels from the G-LAT. The G-LAT can provide a range of labels if possible. If not, the request will be refused.

The segmentation of the LAT can accelerate the label allocation procedure because line cards can assign labels themselves without going through to the control card.

LIB

The LIB contains information for labeling a data packet, changing the current label, or removing a label when the packet reaches the destination. The LIB is subdivided into two main tables: FTN (FEC-TO-NHLFE) and ILM (Incoming Label Mapping).

The FTN table is used for making MPLS forwarding decisions for unlabeled packets. When the ingress line card receives an unlabeled packet, it classifies the packet using a Flow Qualifier. The criteria used to qualify can be QoS class, VPN ID, and so on. If the packet is qualified for an LSP, the MPLS forwarder looks up the FTN table to find an entry which has the NHLFE (Next Hop Label Forwarding Entry) [Rose01] corresponding to the LSP.

- o If an FTN entry is found, the FTN table returns one or more NHLFEs associated with the entry used as instructions for packet forwarding.
- o If the entry is not found, the line card performs IP based forwarding.

Figure 6-10 shows the structure of an FTN record.

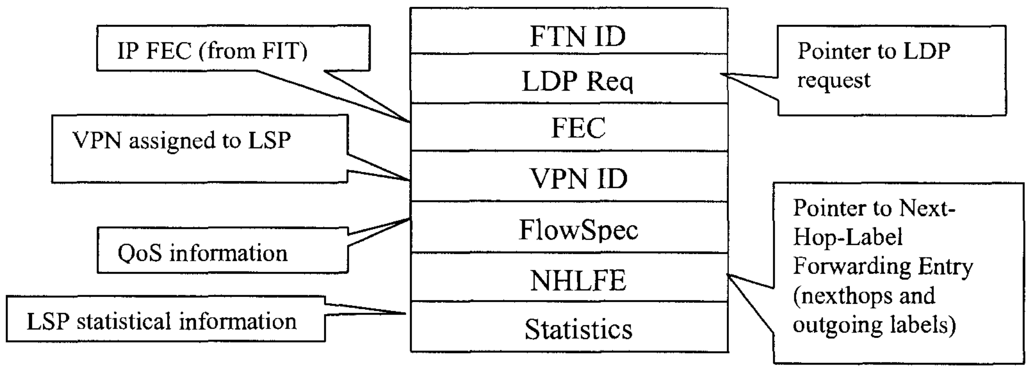


Figure 6-10: FEC-TO-NHLFE (FTN) Table Structure

The ILM table is used for making MPLS forwarding decisions for labeled packets. When the ingress line card receives an MPLS labeled packet, it looks up the ILM table for the next hop and outgoing label using the incoming label as a search key.

- If the lookup is successful, the ILM table returns the NHLFE associated with this label used as an instruction for packet forwarding.
- If the lookup fails, the packet is simply dropped.

Figure 6-11 shows the structure of an ILM record.

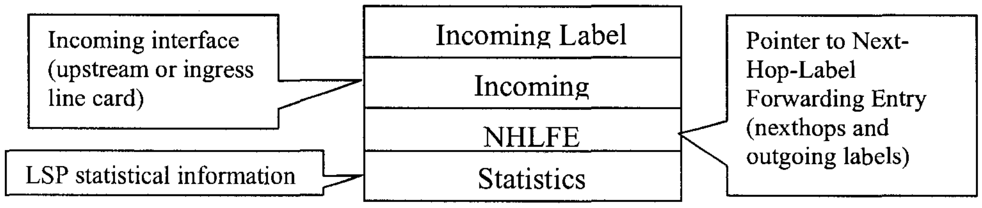
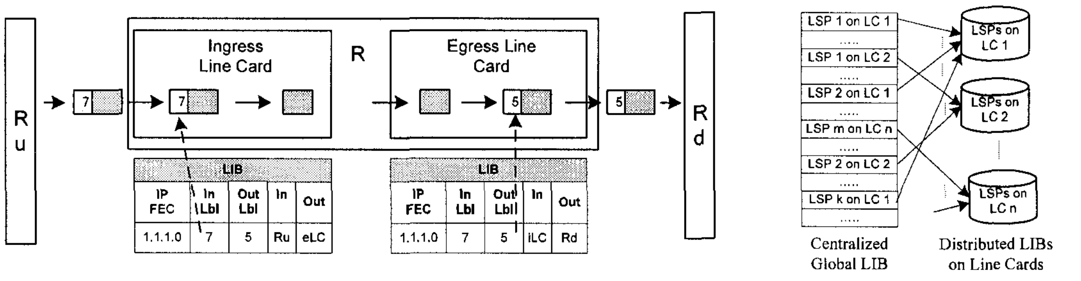

Incoming interface (upstream or ingress line card)
LSP statistical information
Incoming Label
Incoming
NHLFE
Statistics
Pointer to Next-Hop-Label Forwarding Entry (nexthops and outgoing labels)

Figure 6-11: ILM Table Structure

6.4.2 Data Forwarding

When an LIB entry is completed, the MPLS Signaling running on line card records it to the local FIT. As discussed above, both entries on the ingress and egress line card will be registered to the FIT before the routers start the data transmission. An incoming MPLS data packets is qualified for an LIB entry based on its label (incoming label) at the iNP of the ingress line card of the router. If the label of the packet needs to be changed (i.e., incoming and outgoing labels are different), the iNP will strip the current label. The packet is then forwarded through the switch fabric to the corresponding egress line card as indicated on the Outgoing Interface field of the LIB. Flow control mechanisms may be applied on the packet when it enters the iTM before traveling the switch fabric. The packet is then switched to the egress line card where a new label is added based on the Outgoing Label field of the LIB (Figure 6-12 (a)).



(a) Use Distributed LIBs for Label Swapping on Data Packet

(b) Centralized Global LIB vs. Distributed LIBs

Figure 6-12: Using Distributed LIBs

The lookup operation is considerably accelerated in the proposed architecture. Since the LIB on each line card contains only the LSPs that go effectively through the line card, the size of each LIB is considerably reduced compared to the global LIB hosted by the control card in the centralized architecture (Figure 6-12 (b))

The proposed architecture is also able to support data multicasting, which is one of the main requirements of core routers. Nowadays, multicasting is not yet available in the centralized MPLS architecture due to the fact that data packets are all labeled at ingress line cards. Egress line cards are not involved in the packet processing procedure. They simply send or forward the labeled packets to the next hop (Figure 6-13). When a packet is sent to a group of destinations (multicasting), it is cloned at the ingress line card and appropriate labels are added. This is not efficient, however, because the ingress line card has to process repetitively the packets for each destination within the multicast group.

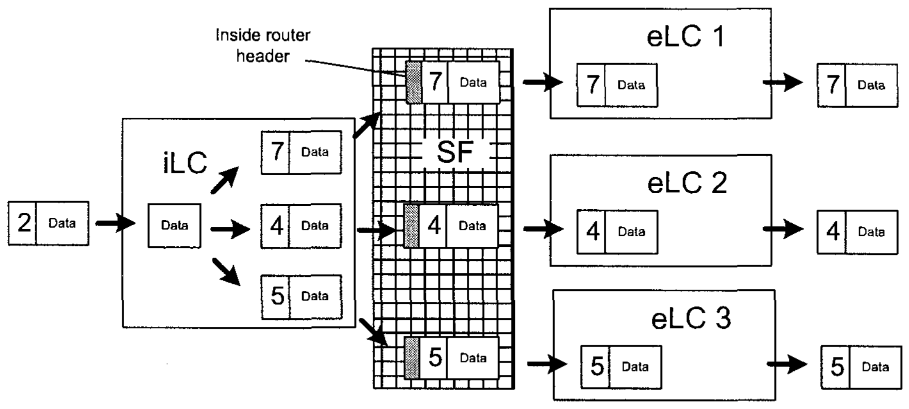


Figure 6-13: Label Swapping in a Centralized Architecture

In order to provide effective multicasting mechanisms, we propose to share the label swapping operation between the ingress and egress line card, as illustrated in Figure 6-14. Since the LIB tables are located on both the ingress and egress line card of a LSP, the label swapping operation can be divided into two sub-operations: popping and pushing. The popping can be performed at the ingress line card while pushing can be done at egress line card. The data packet travels the Switch Fabric with only an internal header, so it can be multicast to the group of destinations. Each egress line card is then responsible for putting the appropriate MPLS label on the data packet before forwarding it to the next hop.

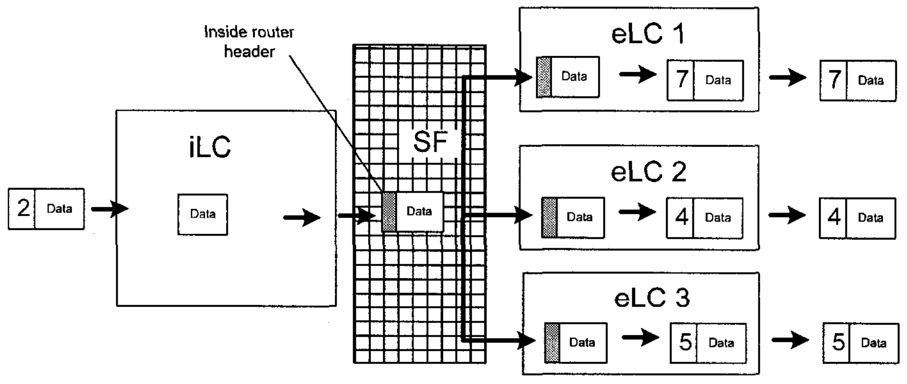


Figure 6-14: Label Swapping in the Proposed Distributed Architecture

Now the whole MPLS/LDP processing at the line card level is summarized in Figure 6-15.

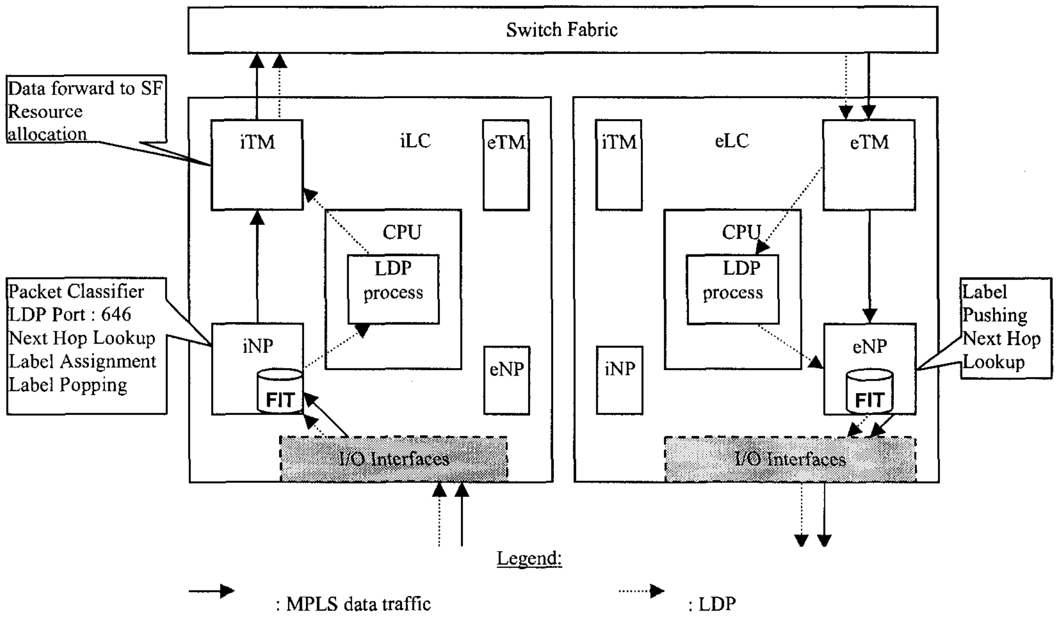


Figure 6-15: Data Packet and Control Message Processing in the Proposed Distributed Architecture

When a packet arrives at a given line card, it is qualified by the Interface Controller to MPLS or IP packet.

- If it is an IP packet, it is identified as a control or data packet, based on the Protocol ID field of the packet header.
- If it is an IP data packet, the iNP will determine whether the packet meets the criteria of a given FEC, using the FTN table of the LIB. If it does, the packet is labeled and then sent to the LSP corresponding to the FEC. Otherwise, the iNP uses the IP forwarding table to forward the packet over the IP network.

- If the packet is a LDP message, the iNP forwards it to the LDP process on the line card.
- If the packet is a MPLS data packet, the iNP uses the ILM table of the LIB to determine the operations to be performed on the packet. The packet label can be changed and the packet is switched over MPLS networks.

6.5 Performance Evaluation

The distributed architecture for MPLS/LDP is indeed proposed in order to fully exploit the advanced architecture of next generation routers. The first requirement for next generation routers is high scalability, which can be expressed in terms of requests or routes the router can support. Since the main bottleneck of the router is the control card, migrating some processing operations from the control card to line cards will save the control card resources and increase the scalability of the router. In this section, we discuss the qualitative and quantitative analysis of the performance achieved by the proposed distributed architecture.

6.5.1 Qualitative Analysis

In general, migrating some of the processing tasks from the control card to the line cards can reduce potential bottlenecks experienced on the control card when the number of requests is increased due to the growth of the number of line cards and routes the core router has to support. In addition, the architecture we propose has the following advantages:

Robustness: parallel processing at the line card level is available in our architecture and waiting queues are avoided. Line cards may independently process the LSPs they are involved in, without having to wait for the reply from the control card as in the centralized architecture. The HELLO sending rate is also accelerated because this process can be triggered locally at the line card level instead of the control card level. The LIB tables and the Adjacency Table are optimized to contain only the LSPs and the neighbors directly related to the line card. Thus, the lookup operation can be sped up.

Scalability: the router will be more scalable if some control tasks, particularly the signaling, can be processed by line cards. The control card will assume only the most complementary tasks; the tasks that need human interactions or the tasks used to interoperate different line cards. In fact, the most important task of the control card is routing and management. Data redundancy is excluded in the LIB and consequently the forwarding table, allowing it to contain more routes. Task sharing between the ingress and the egress line card also enables load balancing among line cards. In Section 6.5.2, we provide practical data demonstrating the higher scalability of our architecture, in terms of processing.

Resiliency: the migration of signaling protocols to the line cards keeps the current session alive if the control card fails. If the control card is required to perform all control tasks, the system will totally shut down when the control card fails. Having a backup control card is a costly solution. We provide a better resiliency mechanism at the line card level as described in Section 6.3.2. Indeed, it is faster to recover from line card failures; moreover a line card is much cheaper than a control card. In addition, problems at the control card level will not slow down the procedures on the line cards.

On the other hand, the distributed architecture can raise some additional management overhead, such as:

Increase in the number of messages exchanged. As shown in the Section 6.3.1, each LABEL REQUEST or LABEL MAPPING message going through the router requires two additional internal messages between the ingress and egress line cards, which is not the case in the centralized architecture. Loading configuration files from the control card to the line cards when MPLS is started or sending user settings to the line cards also consume extra bandwidth on the switch fabric.

More complex software implementation. It is more difficult to upgrade the system because all line cards need to be upgraded instead of just one control card.

Although there would be some trade-off due to the migration of control functions from the control cards to the line cards, we believe that the proposed architecture is a good candidate for dealing with next generation router issues, particularly with a large number of line cards.

6.5.2 Quantitative Analysis

We now conduct a comparison of the performance achieved by the distributed MPLS/LDP architecture we proposed and the centralized one. Table 6-1 shows the configurations of the router used for the experiments in terms of number of line cards per router, number of ports per line card and number of LSPs per port. For each configuration, we compare the performance achieved by the proposed and centralized architectures in terms of CPU cycles and the number of messages exchanged. We work with the assumption of having 10 ports per line card, with an increasing number (between 16 and 128) of line cards per router. The connectivity of the network, in terms of the number of ports per domain and the number of LSPs per port, is also taken into account.

Scenario	Number of line cards / router	Number of ports / line card	Number of of LSPs / port
1	16	10	5
2	32	10	6
3	64	10	7
4	128	10	8

Table 6-1: Scenario Parameters

Let:

N_{LC}: number of line cards in the router. For the configurations shown in Table 6-1, N_{LC} takes the following values: 16, 32, 64 and 128.

$\overline{N}_{Port}$: average number of ports (network interfaces) located on each line card. Usually, all line cards in a router have the same number of ports. In Table 6-1 $\overline{N}_{Port}$ is equal to 10.

$\overline{N}_{LSP}$: average number of LSPs per port. For the configurations shown in Table 6-1, $\overline{N}_{LSP}$ is equal to 5,6,7,8 respectively.

$\overline{N}_{CLabel}$: average number of CPU cycles used for label provisioning operation. This operation is performed on the control card in the centralized architecture, and on line cards in the proposed architecture.

$\overline{N}_{CLIB}$: average number of CPU cycles used to record an LSP to the LIB.

$\overline{N}_{CFIT}$: average number of CPU cycles used by an FIT lookup operation to determine the egress line card and the downstream router based on the IP FEC.

$\overline{N}_{Cmsg}$: average number of CPU cycles used to process a message on the line card or on the control card.

M: required memory to store one route on the line card or on the control card

In the following calculation, we do not consider the HELLO or KEEPALIVE. We also assume that all LSPs go through two line cards.

The number of LSPs that the router has to support is: $\frac{1}{2} \times N_{LC} \times \overline{N}_{Port} \times \overline{N}_{LSP}$

Number of messages going through the switch fabric

In the centralized architecture, for each LSP, the control card receives a LABEL REQUEST message from the upstream router. This message goes through the ingress line card and the switch fabric. The control card then forwards this message to the downstream router, through the switch fabric and the egress line card. In turn, the control card receives a LABEL MAPPING from the downstream router then sends a similar message to the upstream router. Therefore, 4 messages go through the switch fabric for each LSP. Totally, there are $4 \times \frac{1}{2} \times N_{LC} \times \overline{N}_{Port} \times \overline{N}_{LSP} = 2 \times N_{LC} \times \overline{N}_{Port} \times \overline{N}_{LSP}$ messages going through the switch fabric for the whole router.

In the proposed distributed architecture, for each LSP, the ingress line card receives a LABEL REQUEST message from the upstream router. It then forwards this message through the switch fabric to the egress line card. The egress line card sends back an acknowledgement through the switch fabric. Then the ingress line card sends the message to the downstream router through the switch fabric. Therefore, there are 3 messages going through the switch fabric for a LABEL REQUEST processing. Similarly, there are 3 messages required for the LABEL MAPPING in the other direction. Totally, there are $(3+3) \times \frac{1}{2} \times N_{LC} \times \overline{N}_{Port} \times \overline{N}_{LSP} = 3 \times N_{LC} \times \overline{N}_{Port} \times \overline{N}_{LSP}$ messages going through the switch fabric for the whole router.

CPU cycles

In the centralized architecture,

- The control card needs for each LSP: $\overline{N}_{CFIT}$ CPU cycles to determine the egress line card and the downstream router, $\overline{N}_{CLabel}$ CPU cycles to allocate a label from the LAT, $\overline{N}_{CLIB}$ CPU cycles to record the LSP to the LIB, and $2\times\overline{N}_{Cmsg}$ CPU cycles to process a LABEL REQUEST and a LABEL MAPPING. Totally, it requires $(\overline{N}_{CFIT}+\overline{N}_{CLabel}+\overline{N}_{CLIB}+2\times\overline{N}_{Cmsg})\times\frac{1}{2}\times N_{LC}\times\overline{N}_{Port}\times\overline{N}_{LSP}$ CPU cycles to establish all needed LSPs for the whole router.
- A line card needs $2\times\overline{N}_{Cmsg}$ CPU cycles for each LSP: in order to process a LABEL REQUEST (from the upstream router) and a LABEL MAPPING (from the control card) if it is the ingress for a LSP. If the line card is egress, it also needs $2\times\overline{N}_{Cmsg}$ CPU cycles in order to process a LABEL REQUEST (from the control card) and a LABEL MAPPING (from the downstream router). Totally, a line card needs $2\times\overline{N}_{Cmsg}\times\overline{N}_{Port}\times\overline{N}_{LSP}$ to process all LSPs going through its ports.

In the proposed distributed architecture,

- The control card is required to provide the initial label space to all the line cards. Therefore, it needs $\overline{N}_{CLabel}\times N_{LC}$ CPU cycles.
- An ingress line card needs for each LSP: $\overline{N}_{CFIT}$ CPU cycles to determine the egress line card and the downstream router, $\overline{N}_{CLIB}$ CPU cycles to record the LSP

to the LIB and $2 \times \overline{N}_{Cmsg}$ CPU cycles to process a LABEL REQUEST (from the upstream router) and a LABEL MAPPING (from the egress line card). The egress line card needs: $\overline{N}_{CLabel}$ CPU cycles to allocate a label from the L-LAT, $\overline{N}_{CLIB}$ CPU cycles to record the LSP to the LIB and $2 \times \overline{N}_{Cmsg}$ CPU cycles to process a LABEL REQUEST (from the ingress line card) and a LABEL MAPPING (from the downstream router). Since the LSPs are equally distributed on all line cards, the average number of CPU cycles required on each line card is:

$$\frac{1}{2} \times (\overline{N}_{CFIT} + \overline{N}_{CLIB} + 2 \times \overline{N}_{Cmsg} + \overline{N}_{CLabel} + \overline{N}_{CLIB} + 2 \times \overline{N}_{Cmsg}) \times \frac{1}{2} \times \overline{N}_{Port} \times \overline{N}_{LSP}$$

$$= \frac{1}{4} \times (\overline{N}_{CFIT} + 2 \times \overline{N}_{CLIB} + 4 \times \overline{N}_{Cmsg} + \overline{N}_{CLabel}) \times \overline{N}_{Port} \times \overline{N}_{LSP}$$

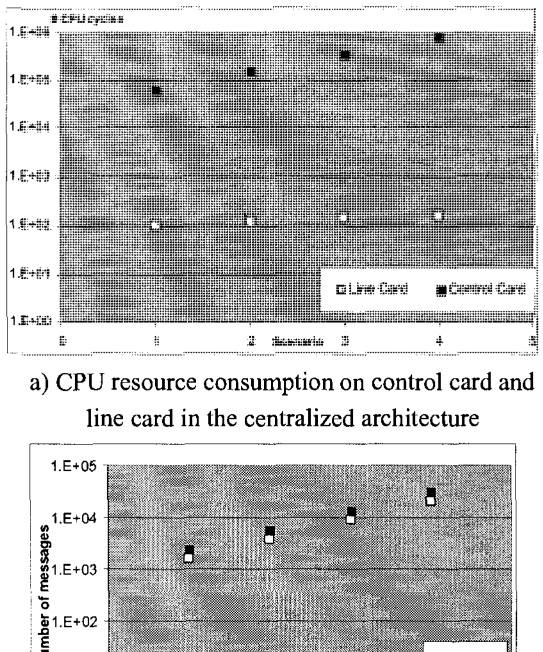
Memory consumption

In the centralized architecture, all LSPs are managed by the MPLS Controller. Therefore, the amount of memory needed for the global LIB on the control card is $\frac{M}{2} \times N_{LC} \times \overline{N}_{Port} \times \overline{N}_{LSP}$. The FIT of each line card has a copy of this table.

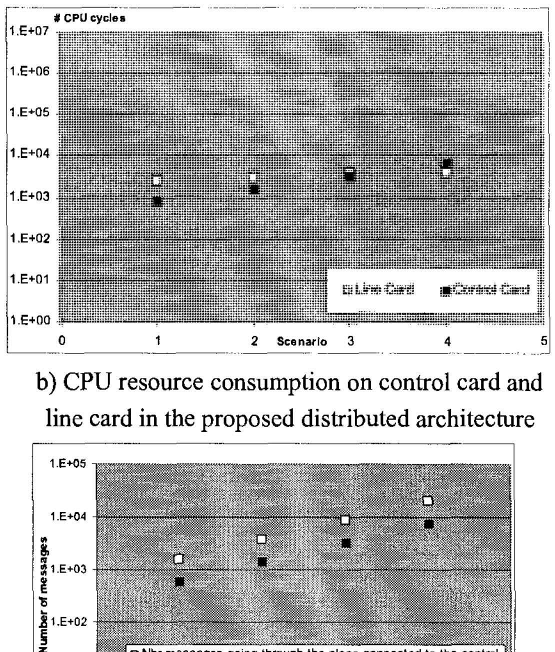
In the proposed distributed architecture, each line card stores only the LSPs effectively going through its port. Therefore, the amount of memory required on each line card is $M \times \overline{N}_{Port} \times \overline{N}_{LSP}$. The whole router needs: $M \times N_{LC} \times \overline{N}_{Port} \times \overline{N}_{LSP}$ memory units.

We can see that the memory needed for the whole router in the proposed architecture is twice compared to the centralized architecture. However, the memory requirement for each line card is much less, especially when the number of line cards increases.

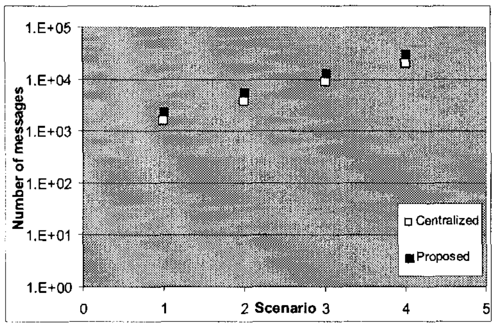
Figure 6-16 (a) and (b) compare the CPU consumption in the centralized and the proposed architecture according to the scenarios in Table 6-1. In the centralized architecture, the CPU requirement on the control card is very high when we add more line cards and/or increase the number of LSPs per port. In the proposed distributed architecture (Figure 6-16 (b)), the CPU load on the control card is lower because most processing tasks are moved to the line cards such as message processing, label provision and table update.



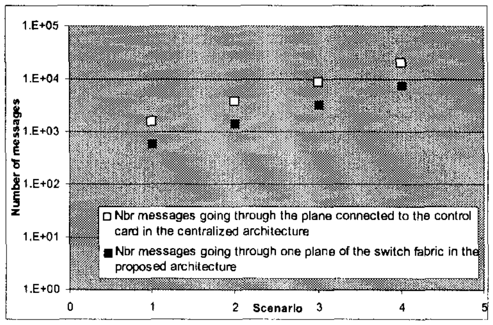
a) CPU resource consumption on control card and line card in the centralized architecture



b) CPU resource consumption on control card and line card in the proposed distributed architecture



c) Total number of messages going through the switch fabric in the centralized and proposed distributed architectures



d) Number of messages going through one plane of the switch fabric in the centralized and proposed distributed architectures

Figure 6-16: LDP Performance in the Centralized and Proposed Architectures

In the centralized architecture, the CPU utilization on line cards is low because it is required only for sending and receiving messages. In the distributed architecture, the CPU requirement on line cards increases in order to handle additional tasks. The number of messages traveling the switch fabric generated by the proposed distributed architecture increases (Figure 6-16 (c)), but not as much, due to the exchanges among line cards, however, it can be easily fulfilled with the large capacity and load balancing (e.g., using multiple planes) of the switch fabric. Indeed, the parallel processing capability of the switch fabric is not efficiently exploited in the centralized architecture because all messages have to go through the plane connecting to the control card. Figure 6-16 (d) shows the utilization of each plane on the proposed architecture in case of router configuration having 4 planes. If we assume that each SF plane serves the same number of line cards and the total number of LSPs is equally distributed on all line cards, the number of messages going through each plane is almost the same and it is less than the number of messages going through the SF plane connecting to the control card in the centralized architecture.

6.6 Chapter Conclusions

This chapter has presented the distributed architecture for MPLS/LDP resulting from the general distributed framework for signaling protocols described in Chapter 4. MPLS support is one of the emergent requirements for the next generation routers. We have reviewed the centralized architecture of MPLS/LDP and then investigated thoroughly the distribution of its functions. The distributed architecture we proposed allows the MPLS signaling to be achieved entirely on the line cards. The result is a significantly increase of the scalability of the router.

We have discussed the new challenges for a distributed architecture of MPLS/LDP, including the synchronization, label provision and table recovery. Some solutions have been proposed to overcome these challenges. In particular, we have proposed a resiliency mechanism that can be achieved at the line card level and that can be deployed efficiently in order to save backup memory on the control card. This chapter also provided an evaluation of the proposed distributed architecture in comparison with the centralized one, in terms of CPU consumption and number of exchanged messages.

The distributed architecture for LDP we presented is a good sample for distributed modules required for next generation routers. With little modification, a similar architecture can also be developed for RSVP-TE in order to complete the MPLS framework, where the synchronization, and recovery mechanisms as well as table management may be reused [Neri07]. The methodology of LDP message processing is useful for RSVP-TE, too. The most important extra requirement for such a distributed RSVP-TE architecture, related to the CSPF computation, will be discussed in the next chapter.

Chapter 7 Distributed RTM Architectures

In this chapter, we present the distributed architecture we propose for the Routing Table Manager (RTM). The first requirement for a RTM distributed architecture comes from the RSVP-TE module. As with LDP, the RSVP-TE module needs also to be distributed on line cards in order to increase the scalability and resiliency and to reduce the load on the control card. Unlike the LDP, the RSVP-TE needs to compute paths over the network based on user-specific requirements, such as QoS. This is done with the help of routing protocols, like OSPF or BGP, through interfaces with the RTM. Therefore, a distributed architecture for the RTM is required to implement a RSVP-TE distributed architecture. In addition, the distributed architecture of RTM may save the CPU resource and memory on the control card(s) that are used to compute the best routes.

This chapter begins with the distributed architecture for RSVP-TE, followed by the proposed distributed schemes for the RTM. The performance evaluation and comparison of the proposed schemes are provided, in terms of the number of exchanged messages, CPU cycles and memory consumption. We also discuss where to deploy the distributed schemes depending on the type of routers (i.e., their hardware capacity). The implementation architecture of the selected distributed RTM scheme is also provided, where we focus on the use of such a distributed RTM for CSPF computations.

7.1 Distributed architecture for RSVP-TE

Routing protocols, such as OSPF [Moy98], IS-IS [ISO02] or BGP [Rekh95] require path computations in order to produce best routes. The LDP protocol presented in the previous chapter (Chapter 6) is a signaling protocol for MPLS networks which simply performs the message exchanges between LSRs to establish the LSPs. With LDP, LSPs are built without taking into account traffic engineering parameters. Therefore, the path computation is not involved in the LDP process. RSVP-TE [Awdu01] can be used alternatively as another signaling protocol for MPLS networks that provides additional traffic engineering features. In traffic engineering we are concerned with establishing LSPs which do not necessarily follow the IP best routes from the ingress to the egress computed by normal routing protocols like OSPF or IS-IS. This allows data to be sent by alternative routes to reduce bottlenecks and congestion, to increase the utilization of network-resources, and to avoid planned faults. The Resource Reservation Protocol (RSVP) [Zhan02] was originally designed as a signaling protocol for the Integrated Services (IntServ) model, wherein a host requests a specific QoS from the network for a particular flow. RSVP-TE is an extension of RSVP that has been adapted to support traffic engineering within the MPLS network.

The architecture of RSVP-TE is basically similar to LDP, described in Chapter 6. Both signaling protocols are running on line cards and can be used alternatively by the MPLS Signaling module to establish the LSPs with the peer routers. Their concurrent access to the LIB table is controlled to avoid data inconsistency. The labels provided by the L-LAT are given to RSVP-TE and LDP according to their requirements. LDP establishes the LSPs for the IP best routes defined in the FIT managed by the IP stack, while RSVP-TE builds the TE-based routes. LDP runs on top of TCP and RSVP-TE uses the raw sockets provided by the IP stack. In Figure 7-1, there is only one RTM running on the control card that contains all the routes of the system and manages the interfaces of the routing protocols. Therefore, all path computation requests from RSVP-TE must go to the control card. The distributed MPLS architecture is not deployed efficiently. In addition, the control card can be overloaded by the large number of requests when the number of line cards is increasing.

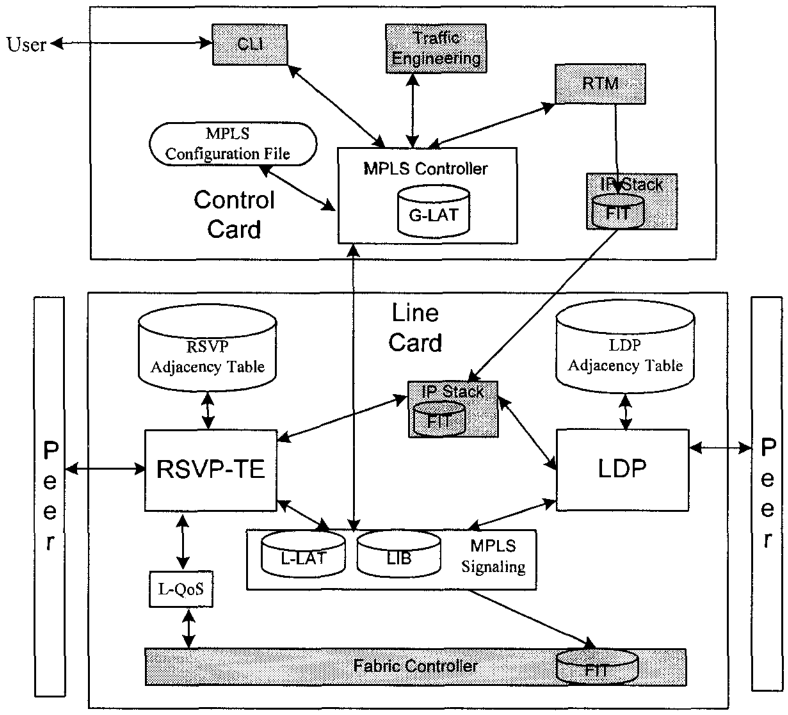


Figure 7-1: Distributed MPLS Architecture with a Centralized RTM

This issue can be dealt with by using a distributed RTM, where each line card has a RTM instance. The distributed RTM on each line card may contain the available routes of the routers, or all the routes which directly relate to the local line card, out of the best routes. RSVP-TE path computation requests may therefore be addressed to the local RTM instance running on the same line card instead of the control card. Based on its routing table, the local RTM will be able to call the appropriate routing protocol to compute the routes that satisfy user QoS requirements.

With a distributed RTM architecture, a distributed RSVP-TE process on line cards will be able to consult the local routing database in order to obtain routes. Routing protocols determine where packets get forwarded; RSVP-TE is only concerned with the QoS of those forwarded packets. RSVP-TE sessions are launched according to user requests from the application level. A host uses the RSVP-TE protocol to request specific QoS from a network for particular application data streams or flows. Routers use RSVP-TE to deliver QoS requests to all nodes along the paths of the flows and to establish and maintain a state to provide the requested services. RSVP-TE requests generally result in resources being reserved at each node in the data path.

Quality of service is implemented for a particular data flow by mechanisms collectively called "Traffic Control" which include:

- Packet scheduling (or some other Layer 2-dependent mechanism to determine when particular packets are to be forwarded),
- Admission control,
- Policy control.

For each outgoing interface, a packet scheduler (or other link-layer-dependent mechanism) is required to achieve the promised QoS. Traffic Control implements QoS service models defined by the Integrated Services Working Group [Bake97]. During reservation setup, an RSVP-TE request is passed to admission control and policy control. Admission control determines whether the node has sufficient available resources to supply the requested QoS. Policy control determines whether the user has administrative permission to make the reservation. If both checks succeed, parameters are set in the forwarding engine (through Traffic Manager chipsets) to obtain the desired QoS. If one of the checks fails, the RSVP-TE process returns an error notification to the application process which originated the request.

7.2 Current RTM Architecture

We now describe RTM architectures to support such RSVP-TE modules.

7.2.1 Overview of RTM

The main task of the RTM is to build the Forwarding Information Table (FIT) from the routing database coming from different routing and signaling protocols [Zini02]. All routes learnt by different routing protocols are stored in the routing database; including the best and the non-best routes. For a set of routes having the same destination prefix, only one route is deemed the best, which is based on a pre-configured preference value assigned to each routing protocol. For example, if static routes have a high preference value and OSPF routes have a low preference value, and if a route entry having the same destination prefix was recorded by each protocol, the static route is considered to be the best route and is added to the FIT (Figure 7-2). However, some services, such as RSVP, can use non-best routes to forward data with respect to user-defined parameters. Therefore, the RTM has to keep all routes and allows users or requested modules to access the route database and make routing decisions based on:

- Request Next Hop and Explicit Route resolution.
- Notify any change in the routing tables generated by the underlying routing protocols (e.g., RIP, OSPF, IS-IS, BGP).
- Alert the routing protocols about the current state of physical links, such as the up/down status, available bandwidth, etc. in order to manage associated link

states, and indirectly route status. This information helps the routing protocols when flooding QoS-related information to the routing domain or building QoS forwarding tables.

- Communicate with the policy manager module for making route filtering decisions for routing protocols (e.g., OSPF or BGP).
- Alert the routing protocol about resource reservation failures.

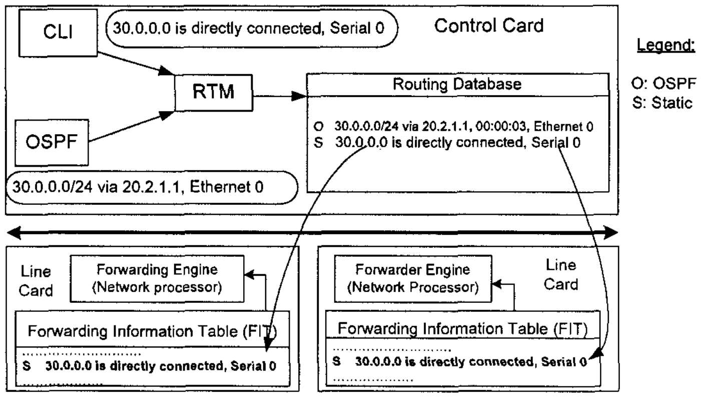

CLI
30.0.0.0 is directly connected, Serial 0
Control Card
RTM
Routing Database
OSPF
O 30.0.0.0/24 via 20.2.1.1, 00:00:03, Ethernet 0
S 30.0.0.0 is directly connected, Serial 0
30.0.0.0/24 via 20.2.1.1, Ethernet 0
Legend:
O: OSPF
S: Static
Line Card
Forwarding Engine (Network processor)
Forwarding Information Table (FIT)
S 30.0.0.0 is directly connected, Serial 0
Forwarder Engine (Network Processor)
Line Card
Forwarding Information Table (FIT)
S 30.0.0.0 is directly connected, Serial 0

Figure 7-2: Update Routing Database and Select Best Routes

The traditional architecture of the RTM module, so called centralized architecture, is neither distributed nor scalable (Figure 7-3). These legacy routers consist principally of a RTM located on the control card [Zini02], which processes information from all routing protocols and every network the router connects to. Indeed, there is no RTM module running on any line card. However, it supports resiliency at the control card level. It manages software and hardware failures of the control card on which it is running by having a backup instance running on another control card that will take over the preceding primary instance in case of failures.

In such an architecture, routing protocols, such as ISIS, OSPF and BGP are all running on the control card. Each protocol computes the best routes for all domains it connects to.

The results are sent to the RTM, which then selects the overall best routes among all routing protocols and updates the FIT through the services provided by the IP stack.

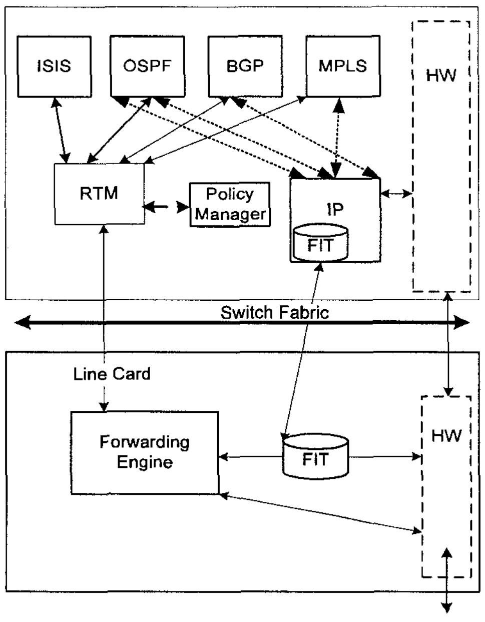


Figure 7-3: RTM in a Non-Distributed Routing Architecture

The RTM is also responsible for managing routing policies for the routing protocols. An external policy module is therefore required to provide policy information (i.e., QoS or traffic engineering information) to the RTM. The policy module allows system administrators to configure the filtering policies and inter-working between routing protocols (i.e., OSPF-BGP inter-working), and to modify path attributes according to specific pre-configured policies. The content of the policy module consists of the policy rules. Each rule is an association of conditions and actions. When the conditions are met, the router is required to trigger the specific actions. A condition is a *boolean* expression that is evaluated to be either true or false. An action represents a concrete treatment that should be taken to enforce a policy rule if the conditions are evaluated to be true. Policy actions may result in executing one or more operations to affect and/or configure network traffic and resources.

We will not go into details of the routing policy implementation architecture in this thesis. It is hence considered as an independent module which has only an interface with the RTM. The protocols, like MPLS, use its services through the RTM in order to filter the control packets out of the data packets and to map the routes as required.

7.2.2 Current RTM Architecture for Next Generation Routers

The RTM architecture used for recent router products, as described in [Ngu07a], consists of a Global RTM (G-RTM) module that manages the routing table for the whole system and several smaller RTMs, each devoted to a given protocol and therefore denoted by IGP-RTM or EGP-RTM, i.e., for each protocol. Each IGP/EGP-RTM is responsible for managing routes computed by a specific routing protocol (i.e., OSPF, BGP, MPLS), as shown in Figure 7-4. The G-RTM is a stand-alone process located on any control card in the system. It interfaces only with the IGP/EGP-RTMs and IP. It receives all the routes learned by the different IGP/EGP-RTMs and performs the selection of the overall best routes. Then it updates IP and the IGP/EGP-RTMs and finally performs route redistribution between protocols. The G-RTM also manages the configuration of static routes configured by users (through an external routing policy module) or traffic engineering based routes. The interface configuration and up/down status are handled by the G-RTM and broadcast to the routing protocol modules through the IGP/EGP-RTMs.

Each IGP/EGP-RTM is physically linked with a given protocol and gives the impressions to the protocol that it is a complete RTM. It contains all best routes of the protocol and the overall best routes of the system computed by the G-RTM used to update the protocol. It offers the same API, and keeps all pertinent information to the protocol for fast access and sustained performance.

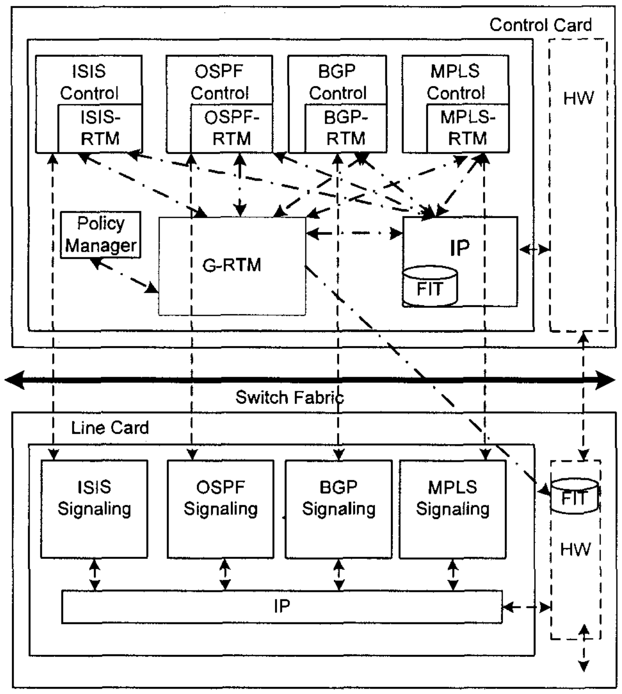


Figure 7-4: Current RTM Architecture: Distribution on Protocol Basis

Such an architecture allows the routing protocols to have flexible access to the routing tables managed by the G-RTM, without being queued. The architecture does not require much modification of routing protocol modules and the G-RTM regarding the traditional architecture (Figure 7-3). Each IGP/EGPRTM manages only the subset of the routing table that is related to a specific routing protocol. When a routing protocol receives a route update notification message through the corresponding signaling component on a line card, the control component located on the control card re-computes the best routes and updates its local IGP/EGP-RTM. The G-RTM is also notified through the link with the IGP/EGP-RTM. The overall best routes of the system are selected among those provided by different protocols. The route update is advertised by the G-RTM to other routing protocols in order to notify the neighbors. Finally, the overall best routes are updated to the FIT on the line cards through the connection with the G-RTM.

The architecture does not reduce the number of messages sent to the control card, in other words, the congestion still remains. However, the resiliency and scalability are improved at the control card level because the routing protocol can still use the IGP/EGP-RTMs when the G-RTM temporary fails. The lookup operation is faster because each IGP/EGP-RTM contains only a portion of the global routing table. The main advantage of this scheme is the simplicity since not much modification is needed. Therefore, it can be suitable for the short-term migration from the current to the next generation routers, where only the control card needs to be upgraded. However, there are some critical issues:

- Although the IGP/EGP-RTMs are distributed on a per protocol basis, they are basically independent processes running on the same control card. This leads to quite heavy resource consumption and to some overloading of the control card as the number of routes increases.
- Additional computing and memory resource on the line cards are not efficiently exploited to run the best route computations or to perform the route management tasks.
- In the context described in Chapter 5 where the routing protocols will be distributed on the line cards in order to improve the scalability and fully exploit the available memory and CPU resource of the line cards [Ngu07a], the IGP/EGP-RTM modules need also to be migrated to the line cards.

- It is not very efficient to perform the FIT update operations at the control card level by G-RTM as the FITs are hosted by the line cards.

In order to deal with these issues, in the following sections, new architectures for RTM will be proposed. We aim at a full distribution of RTMs on line cards, where each line card has an RTM instance responsible for the local routing protocols.

7.3 Proposals for Scalable and Distributed RTM Architectures

Taking into account the necessity of sharing the processing between control cards and line cards, we conducted a study where the aim was the migration of some control operations from the control card to line cards. Candidates for the migration included the three following items.

- *Protocol processing*: This is achieved by the IGP/EGP routing protocol modules, such as OSPF, IS-IS or BGP. Most of these processes can indeed be moved onto the line cards.
- *Route maintenance*: This is achieved by RTM. This operation can be shared between line cards and control cards.
- *Router management and user interface*: This is achieved by appropriate modules such as CLI. These operations should remain at the control card level in order to ease the management.

Therefore, this section focuses on the distribution schemes for the first two operations. We consider and evaluate four distribution schemes that we next describe.

7.3.1 Scheme 1: Basic Distribution of RTM

In order to ease the CSPF computations and to reduce the load of the control card due to the large number of requests addressed to the G-RTM, ones think about the migration of G-RTM functions to the line cards. In the simplest distribution scheme, a copy of G-RTM is implemented on each line card of the router. Thus, each line card has a so called LC-RTM which plays the role of the G-RTM of the control card (Figure 7-5). The best route computation for each routing protocol and the selection of overall best routes of all routing protocols are still achieved on the control card, respectively by control components of the routing protocols and by the G-RTM. Once new routes are added to the routing database, the G-RTM will update the LC-RTMs on the line cards. The LC-RTMs are responsible for recording the overall best routes to the FIT of the line card and for serving the CSPF computation requests. Since the LC-RTM contains all available routes of the router, as well as the parameters of all links, it is able to achieve the CSPF request sent by the RSVP-TE module running locally on the same line card.

Advantages

This architecture may reduce the load on the control card because CSPF computation requests are served by the LC-RTM running on the same line card with the RSVP-TE module. Recording overall best routes to the FIT is also accelerated with the interface between the LC-RTM and the Fabric Controller on line cards.

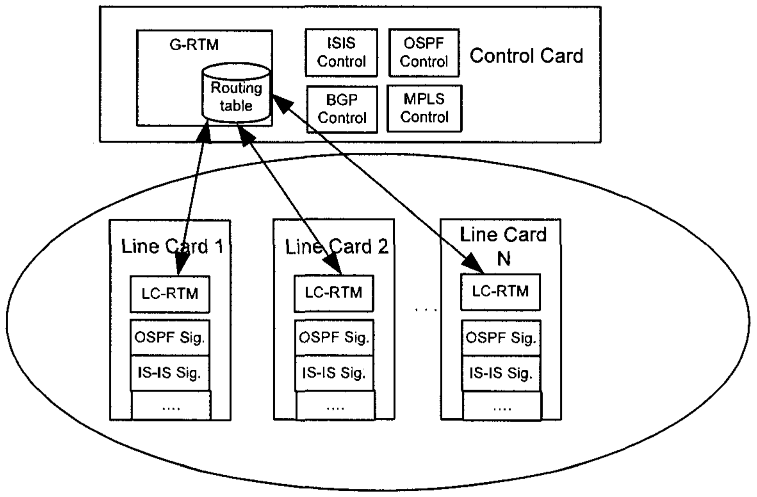


Figure 7-5: Basic Distribution of RTMs

Disadvantages

This architecture requires a large amount of memory on the line cards to save the whole routing table. It does not help to reduce the best route computations, which are achieved by routing protocols on the control card. The LC-RTM may contain many routes that are never used for the line card so the lookup operation is not optimal.

7.3.2 Scheme 2: Distribution of Routing Protocols

In this scheme, the IGP/EGP routing protocols are distributed on the line cards and the best route computation is entirely done at the line card level. The IGP/EGP-RTMs associated with each routing protocol (i.e., in the centralized architecture) are removed because whenever the best routes are determined, they are sent directly to the G-RTM located on the control card.

In the centralized architecture, the control component of each routing protocol running on the control card performs the best route computation for the corresponding protocol. It collects the link state notifications received by all the line cards and builds the network topology. Taking into account the fact that routers are usually grouped together in domains or sub-networks according to the routing protocol policies (e.g., OSPF, IS-IS, etc.), we propose an improvement in the distribution scheme which migrates the computation from the control card to the line cards in order to reduce the load of the control card and to increase the performance and the scalability of the whole router. In this architecture, line cards are grouped into clusters according to their physical interconnection domains and link state notification messages are limited in each cluster. A line card that is out of a cluster will not receive the link state notifications of that cluster, hence is not involved in the best route computation process. Each domain has a "proxy" line card for each routing protocol, which takes care of the path computation for the domain as described in Chapter 5.

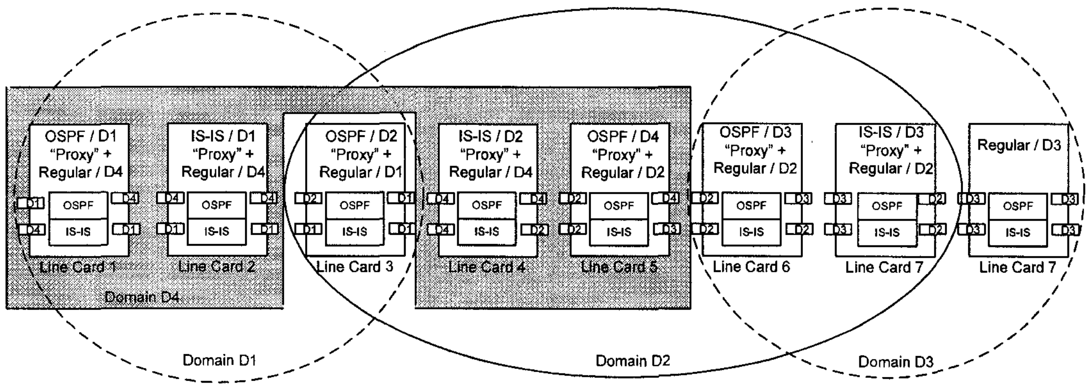

OSPF / D1 "Proxy" + Regular / D4
IS-IS / D1 "Proxy" + Regular / D4
OSPF / D2 "Proxy" + Regular / D1
IS-IS / D2 "Proxy" + Regular / D4
OSPF / D4 "Proxy" + Regular / D2
OSPF / D3 "Proxy" + Regular / D2
IS-IS / D3 "Proxy" + Regular / D2
Regular / D3
OSPF
IS-IS
Line Card 1
Line Card 2
Line Card 3
Line Card 4
Line Card 5
Line Card 6
Line Card 7
Line Card 7
Domain D4
Domain D1
Domain D2
Domain D3

Figure 7-6: Domains and Clusters of Line Cards

Let us illustrate this concept on Figure 7-6 where we consider 8 line cards with 4 ports each, the use of 2 protocols (OSPF and IS-IS) and 4 domains (D1, D2, D3, D4). Consider the line card #1 (denoted by LC1): it has two ports linked to domain D1 and two ports linked to domain D2. Therefore it belongs to two clusters, the first one associated with D1, and the second one associated with D2. The cluster associated with D1 contains 3 line cards, LC1, LC2, LC3. While LC3 is a regular line card, the other two line cards serve as proxy ones: LC1 for OSPF and LC2 for IS-IS.

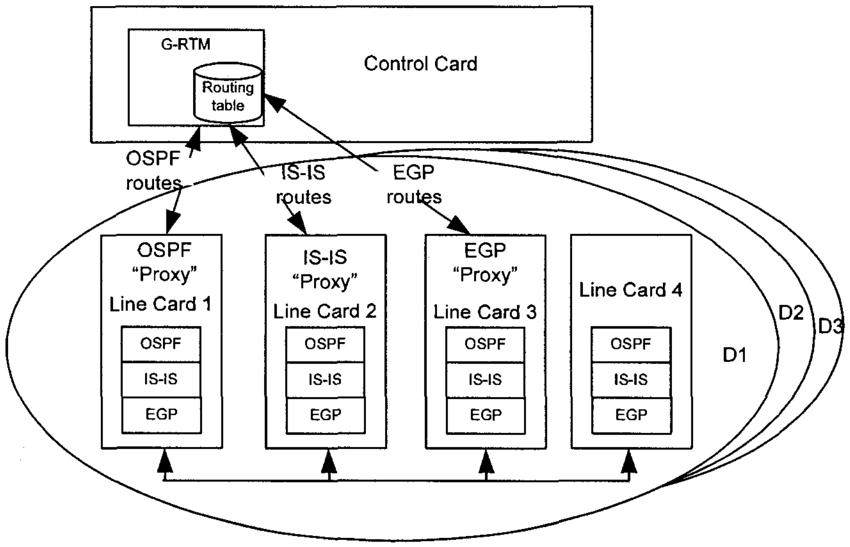


Figure 7-7: Distribution of Routing Protocols

An OSPF and an IS-IS module used to exchange protocol messages with other routers in the network run on each line card. In our example, each line card is connected to four different routers through its ports. If the state of any link in a domain is changed, a notification will be sent to the neighbor router pertaining to that domain. In return, each router in the domain will compute the best routes (e.g., using Djikstra algorithm for OSPF and IS-IS) to other routers in the domain following the reception of the link state update being flooded to all routers connected in the domain. Basically, all routers in a domain receive the same information. Therefore having a proxy line card per protocol for each domain avoids repeating the computation on each line card within that domain, plus ensures that a central instance will not be overloaded by having to calculate the best routes for both domains. In our example, OSPF best routes for the domain D1 are computed on LC1, IS-IS best routes for the domain D1 are computed on LC2. The LC3 is not involved in any best route computation of the domain D1, however, it performs the OSPF best route computation for the domain D2 as it has some ports connecting to D2.

The best routes computed by each proxy in each domain and for each routing protocol is sent to the G-RTM which then selects the overall best routes of the system based on the protocol preference value (e.g., administrative distance) or on the router configuration, as shown in Figure 7-7. The G-RTM updates these final results to the FIT on all line cards as the FIT is consulted to forward data packets. Each proxy in the proposed scheme maintains a database to keep the best routes of its domain. The G-RTM handles all best routes from all domains and routing protocols in the system. Any change in system interface configurations and up/down status are sent by the G-RTM to the proxies, which then notify the neighbor routers through the appropriate routing protocols.

We illustrate on Figure 7-8 the message sequence exchanged between a proxy line card and a regular line card belonging to the domain *D* managed by the proxy. Whenever a link state modification is received, each line card in domain *D* must send a report to the proxy. The proxy computes the best routes and advertises the result to all line cards in domain *D*.

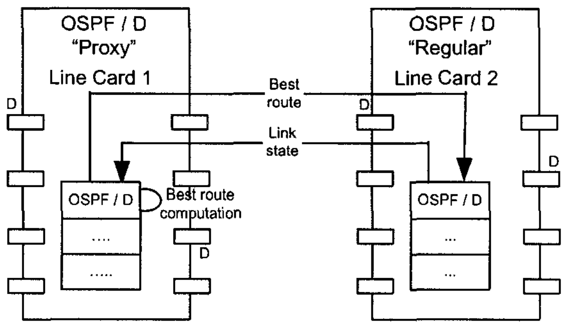


Figure 7-8: Messages Exchanged between a Proxy and a Regular Line Card for the Second Distribution Scheme (Distribution of Routing Protocols)

This scheme can be deployed in the case where only some ports of each line card of the router are connected to a routing protocol domain. Those can be medium scale routers with a limited number of line cards so the ports of a line card are shared among different domains. The scheme can also be preferred in case where ports are changed flexibly from a domain to other. Thus we can assume that the number of ports per domain is chosen randomly.

The architecture achieves the CSPF computation requests as follows. A CSPF computation request sent by a RSVP-TE module running on a given line card is forwarded to the G-RTM located on the control card as in the centralized architecture. However, the G-RTM does not compute the CSPF path on the control card. It looks in the routing database for the appropriate routing protocol and the domain where the CSPF needs to be computed. The G-RTM then forwards the request to the proxy assuming the route computation of that domain to achieve the CSPF, taking into account user-specific parameters.

Advantage

The main advantage of this scheme is the saving of the control card resources. It is also scalable because the load balancing among all line cards can be achieved by having a good proxy assignment mechanism, i.e., as shown in Figure 7-6, each line card can be proxy for a specific routing protocol. As the highest resource consumption is due to the best route calculation using Bellman-Ford or Dijkstra's algorithms, when this task is performed in parallel on several proxies, the overall performance of the router will be improved in comparison to the current RTM architecture.

Disadvantage

The drawback of this scheme is that the routing protocol modules need to be modified in order to enable message exchanges among line cards. This can lead to extra development costs. In addition, since all routes are still stored by G-RTM, the CSPF computation (e.g., for RSVP-TE paths) must be performed at the control card level, thus message exchanges between the line cards and the control card should be taken into account.

7.3.3 Scheme 3: Distribution of LC-RTMs on Line Cards

For highly scalable routers, all ports on each line card connect to only one routing domain. The cabling connections are maintained for a long time (e.g., several months or years). Thus each port is attached permanently to its routing domain. Several protocols can be executed alternatively to perform the routing on that domain (e.g., OSPF and IS-IS). If one protocol temporary fails, the routing can be resumed by other protocols.

In this context, our third proposed distributed scheme enhances the second one by having a "master" for each cluster of line cards in addition to the proxies (Figure 7-9). Each cluster is associated with a domain as in the previous scheme. The main difference between the second and the third scheme is that in the previous one, the proxy of a given domain and a routing protocol computes the best route then sends the result to the G-RTM located on the control card in order to select the overall best routes of all routing protocols of the system; while in the latter scheme, the master line card instead performs the overall selection of the best routes from all routing protocols in its domain. In addition, in the second scheme, all available routes are stored by the G-RTM and only a copy of the overall best routes is kept by line cards to perform the forwarding. Unlike the previous schemes, in the third scheme, available routes of all routing protocols of a domain are managed by the master of that domain. This allows the master to be able to perform further the CSPF calculation on user-specific parameters (e.g., administrative weights) according to special requests (e.g., from RSVP).

In this scheme, routes are managed at the line card level by a so called LC-RTM process running on each line card (Figure 6). The LC-RTM handles the overall best routes of the domain supplied by the master. If a line card is assigned as master for a domain, its LC-RTM contains all available routes of the domain. The FIT of each line card is updated by the local LC-RTM. The LC-RTM of each line card has interfaces with local routing protocols modules so that route notifications can be sent from a given protocol to another one in order to advertise neighbors.

Any CSPF computation request (i.e., sent by a RSVP-TE module) from any line card in a domain will be forwarded to the master line card of that domain. The master LC-RTM performs the CSPF computation based on its database and on the IGP topology of the domain.

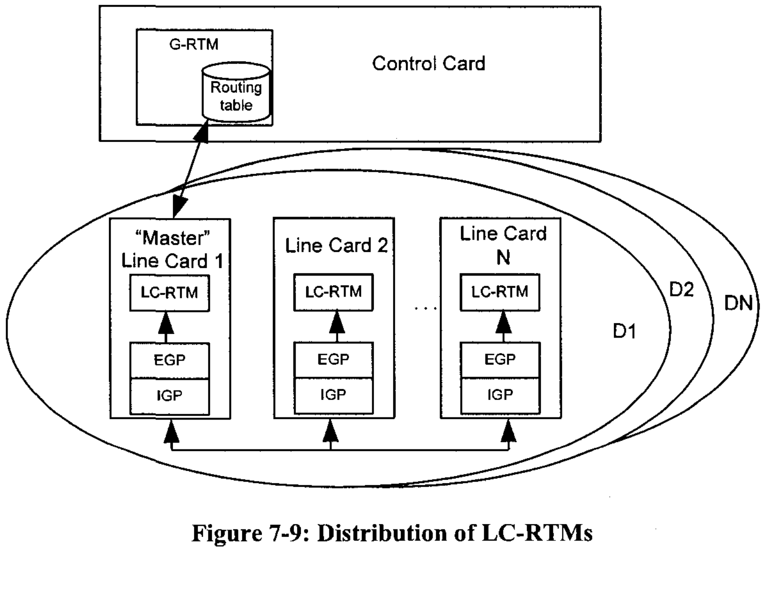


Figure 7-9: Distribution of LC-RTMs

Figure 7-10 shows the message sequence exchanged between a master line card and a regular line card belonging to the domain managed by the master. Whenever a link state notification is received, routing protocols will send a report to the LC-RTM. The LC-RTM forwards the notification to the master, which is responsible for computing and selecting best routes for all routing protocols.

The IGP/EGP-RTMs associated to each routing protocol as in the centralized architecture are no longer required in this scheme because the local LC-RTM of each line card has an interface with local routing protocol modules and the master LC-RTM contains all the best routes of the domain.

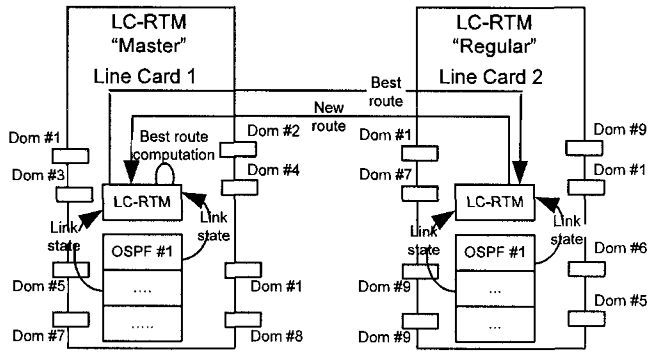


Figure 7-10: Message Exchange between the Master and a Regular Line Card for the Third Distribution Scheme (Distribution of LC-RTMs)

Advantages

The main advantage of this scheme is that route information is handled on a master line card for each domain. Thus it can be used for IGP/EGP or RSVP-TE protocols without having to go up to G-RTM. In this case, CSPF computation is conducted at the level of the master line card. The trade-off is that the master line card must reserve a large memory space for storing all routes and the IGP topology of the domain. In addition, the number of message exchanges between the master line card and other line cards in the domain can be large, i.e., a request should be sent to the master per LSP computation.

The resiliency of the system in the proposed scheme is also improved because when the control card temporary fails, routing can still be achieved at the line card level and the overall best route for each domain can be computed independently by the master line cards.

Disadvantage

A large memory resource is required at the master line card in order to contain all available routes in the domain. It can be therefore costly in terms of line card memory. In addition, the CSPF computation for every LSP in the domain requires message exchanges with the master: this results in an increase of the traffic among the line cards associated with the same domain.

7.3.4 Scheme 4: Distribution of Best Route Computation Algorithms.

The best route computation algorithm, such as Djikstra, needs to have the global topology of the network. Therefore, it must be performed in one router component, namely line cards or the control card. This scheme is proposed assuming we have an algorithm that is able to determine the shortest path from one node to other node in a network in a distributed way. On other words, parts of the best route computation algorithm can be performed on different router cards. The final result can be converged through the messages exchanged among the cards.

In this scheme (Figure 7-11), line cards are again grouped according to the domains they are associated with, but there is no proxy line card for each domain. Instead, the route computation algorithm can be implemented in a distributed way. Each line card performs a fragment of the computation algorithm and synthesis of the distributed computations is handled through the message exchanges among line cards. In such an architecture, each line card manages only the routes connecting to its neighbors. Each line card computes only some best routes of the domain and the final results are sent to the G-RTM in order to select the overall best routes.

However, a distributed algorithm for best route computation is out-of-scope of this thesis so we do not consider the implementation of this scheme.

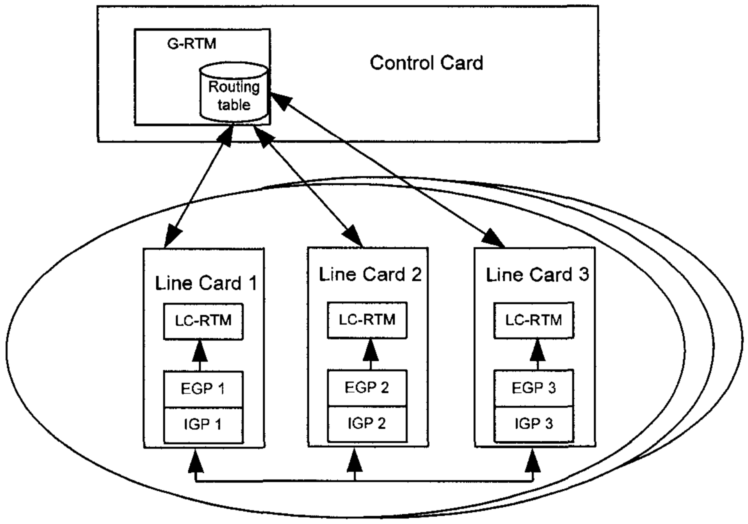


Figure 7-11: Distribution of Route Computation Algorithms

Advantage

This scheme is able to balance the computation task among line cards because each line card performs only a portion of the algorithm. Each line card maintains the available routes; hence the CSPF computation can be executed locally within the domain.

Disadvantage

There is currently no suitable distributed algorithm for the best route computation.

7.4 Performance Evaluation of the Proposed Distributed Schemes

The distribution schemes are proposed in order to fully exploit the hardware architecture of the next generation routers. In addition, they increase the scalability of the system in terms of the number of routes the router can support. We conduct a comparison of the performance achieved by each distribution scheme in order to select the most appropriate one for our requirements.

In this evaluation, we focus on the second and the third distribution scheme (i.e., distribution of routing protocols and distribution of LC-RTMs respectively). We do not consider the first scheme because it is too costly in practice. In addition, we will compare the performance of these two distribution schemes with the current centralized architecture.

We first set the notations, then evaluate the number of messages going through the switch fabric, the CPU and the memory consumptions before conducting a comparative performance evaluation in the last paragraph of this section.

Let:

N_{LC} : number of line cards in the router.

$\overline{N}_{Port}$: average number of ports (network interfaces) located on each line card. Usually, all line cards in a router have the same number of ports.

N_P: number of protocols currently running. We assume that all N_P protocols are activated in all line cards.

$\overline{N}_{Cmsg}$: average number of CPU cycles used to process a message on the line card or on the control card.

$\overline{N}_{Sel}$: average number of CPU cycles used to select an overall best route among N_P best routes of N_P routing protocols. We assume that best routes to a destination learnt by N_P protocols are computed at the same time.

M: required memory to store one route on the line card or on the control card

N_D^j : number of domains for the routing protocol j.

- For the second scheme, we assume that the domains of the routing protocol j are equally distributed on all line cards. Each domain has a proxy. So the number of proxies for the routing protocol j is also N_D^j.
- For the third scheme, since all ports connecting to a domain are included in a cluster and all protocols are activated in that domain, $N_D^j = N_D$ for all j, and the number of line cards used to connect to each domain is: N_{LC} / N_D

P_i^j : number of ports the router uses to connect to domain i of the routing protocol j.

- For the second scheme, we assume that these ports are distributed equally on all line cards and each line card has at maximum one port connecting to the domain i of the routing protocol j.
- For the third scheme, since all ports connecting to the domain i are included in a cluster and all protocols are activated in that domain, $P_i^j = P_i$ for all j, and $P_i = (N_{LC} / N_D) \times \overline{N}_{Port}$

$N_{R_i}^{j}$: number of routers in the domain i of the routing protocol j except the router in consideration. In practice, a router can be reached by several routing protocols.

- For the third scheme, since all protocols are activated on the same domain, $N_{R_i}^{j} = N_{R_i}$ for all j.

$N_{C_i}^{j}$: number of needed CPU cycles in order to run the route computation algorithm (either on the line card or on the control card) for domain i and routing protocol j.

Thus, the total number of domains for all protocols, as well as the total number of proxies for all routing protocols in the second scheme is $\sum_{j=1}^{Np} N_D^j$.

The average number of domains a line card can belong to, as well as the average number of proxies per line card is: $(\sum_{j=1}^{Np} N_D^j)/N_{LC}$

In the following calculation, we do not consider the messages used by a routing protocol to establish and maintain connections such as HELLO or KEEPALIVE because they can be processed entirely at the line card level thus are not involved in the path computation process performed by the RTMs.

7.4.1 Number of Messages Going Through the Switch Fabric

For a link state routing protocol j, when there is a link change in a domain i, all ports connecting to that domain are notified by a message. Thus the router receives $\sum_{i=1}^{N_D^j} P_i^j$ messages. This number is the same for all the proposed distributed schemes.

- In the centralized architecture, all $\sum_{i=1}^{N_D^j} P_i^j$ link notification messages received by all line cards are forwarded through the switch fabric to the protocols control components located on the control card in order to compute a new best route. Since there are N_P protocol modules, there are $\sum_{j=1}^{N_P}\sum_{i=1}^{N_D^j} P_i^j$ messages processed by the control card. The G-RTM updates the FITs on all line cards with the overall best routes. We assume that a message can contain all the overall best routes of a domain. Therefore, the number of update messages going through the switch fabric is $N_{LC} \times \sum_{j=1}^{N_P} N_D^j$.

- In the second scheme, $P_i^j - 1$ link notification messages received by non-proxy line cards of domain *i* of a given routing protocol *j* are forwarded to the proxy of their domain (the notification message received by the proxy is processed locally so it does not consume bandwidth of the switch fabric). Since there are N_D^j domains for the routing protocol *j*, there are $\sum_{i=1}^{N_D^j}(P_i^j - 1)$ messages sent by line cards to their proxies within the protocol *j*. And as there are N_P protocol modules, $\sum_{j=1}^{N_P}\sum_{i=1}^{N_D^j}(P_i^j - 1)$ messages are sent by line cards to their proxies. In addition, each proxy of a domain *i* and routing protocol *j* sends a message containing the best route of its domain to the G-RTM located on the control card. Therefore, the number of best route messages sent over the switch fabric is $\sum_{j=1}^{N_P} N_D^j$. Totally, the number of messages sent over the switch fabric in the distributed architecture is $\sum_{j=1}^{N_P} N_D^j + \sum_{j=1}^{N_P}\sum_{i=1}^{N_D^j}(P_i^j - 1) = \sum_{j=1}^{N_P}\sum_{i=1}^{N_D^j} P_i^j$, exactly as same as in the centralized architecture. Similarly, the G-RTM updates all line cards with the overall best routes. Thus the number of update messages going through the switch fabric is $N_{LC} \times \sum_{j=1}^{N_P} N_D^j$ as in the centralized architecture.

- In the third scheme, each line card receives $\overline{N}_{Port}$ messages from its $\overline{N}_{Port}$ because it uses all ports to connect to the domain *i*. These messages are the same for the protocol j. The routing protocol module forwards only one notification per protocol to the master of the domain. Since there are N_{LC} / N_D line cards in the domain, the number of messages forwarded through the switch fabric to the master is $(N_{LC} / N_D) - 1$. With N_P protocols, the total number of messages sent to the master of the domain i is $N_P \times ((N_{LC} / N_D) - 1)$. There are N_D domains in the system, so the number of message sent to the masters is $N_P \times (N_{LC} - N_D)$. The master updates the regular line cards in its domain with the overall best routes of the domain. Therefore, each master sends $(N_{LC} / N_D) - 1$ update messages. All masters send $N_{LC} - N_D$ update messages in total.

7.4.2 CPU Consumption

In order to compute best routes:

- In the centralized architecture, the protocols control component located on the control card uses $N_{C_i}^j$ CPU cycles to compute best route for the domain i for a given protocol j. Since there are N_D domains and N_P protocols running, the number of cycles needed on the control card is: $\sum_{j=1}^{N_P}\sum_{i=1}^{N_D^j} N_{C_i}^j$.

- In the second scheme, the computation is achieved by proxies. Each proxy performs the computation for a domain and a routing protocol. The number of CPU cycles consumed on the proxy for the path computation is therefore: $N_{C_i}^j$. The total number of CPU cycles used by the whole system is unchanged: $\sum_{j=1}^{N_P}\sum_{i=1}^{N_D^j} N_{C_i}^j$. However, since the proxies are equally distributed on line cards, the number of CPU cycles used for the best route computation on a line card on average is $(\sum_{j=1}^{N_P}\sum_{i=1}^{N_D^j} N_{C_i}^j)/N_{LC}$.

- In the third scheme, the computation is achieved by the proxies of protocols and the overall best routes are selected by the master line cards. Therefore CPU cycles consumed by each proxy is similar to the second scheme: $N_{C_i}^j$ cycles per proxy. In other words, each line card uses on average $(\sum_{j=1}^{N_P}\sum_{i=1}^{N_D^j} N_{C_i}^j)/N_{LC}$ CPU cycles.

In order to select the overall best routes:

- In the centralized architecture and the second scheme, the G-RTM on the control card chooses among best routes computed by all routing protocol modules. For each domain, it takes $\overline{N}_{Sel}$ CPU cycles. The number of common domains for all protocols is $(\sum_{j=1}^{Np} N_D^j)/N_P$. Therefore, the number of CPU cycles to select the overall best routes is: $(\overline{N}_{Sel} \times \sum_{j=1}^{Np} N_D^j)/N_P$.

- In the third scheme, the masters assume the overall best route selection. Therefore, each master uses $\overline{N}_{Sel}$ CPU cycles.

In addition, some computing resources are also required for processing the messages:

- In the centralized architecture, the control component of the protocol j located on the control card has to process $\sum_{i=1}^{N_D^j} P_i^j$ link notification messages coming from domains of the protocol j. Therefore, the number of CPU cycles consumed by all control components to process protocol messages is $\overline{N}_{Cmsg} \times \sum_{j=1}^{N_P} \sum_{i=1}^{N_D^j} P_i^j$.
- In the second scheme, the proxy of the domain i and the protocol j processes $P_i^j - 1$ link notification messages coming from all ports in the domain. Therefore, the number of CPU cycles used by that proxy is: $\overline{N}_{Cmsg} \times (P_i^j - 1)$. In total, the number of CPU cycles used by all proxies for message processing is $\overline{N}_{Cmsg} \times \sum_{j=1}^{N_P} \sum_{i=1}^{N_D^j} (P_i^j - 1)$. In other words, the number of CPU cycles used on average by each proxy for message processing is $(\overline{N}_{Cmsg} \times \sum_{j=1}^{N_P} \sum_{i=1}^{N_D^j} (P_i^j - 1))/N_{LC}$ since the proxies are equally distributed on all line cards. The control card receives a best route update per domain, therefore it uses $\overline{N}_{Cmsg} \times \sum_{j=1}^{N_P} N_D^j$ CPU cycles.

- In the third scheme, each line card sends only one link notification message per protocol to the proxy because all the ports on a line card are connected to the same domain. With N_{LC} / N_D line cards per domain each proxy uses $\overline{N}_{Cmsg} \times (N_{LC} / N_D - 1)$ CPU cycles. The master has to process a best route update message per proxy in its domain. Therefore, it uses $\overline{N}_{Cmsg} \times N_P$ CPU cycles. The control card receives an overall best route message per domain, thus it uses $\overline{N}_{Cmsg} \times N_D$ CPU cycles.

7.4.3 Memory Consumption

Memory is required to store routes.

- In the centralized architecture, the best routes of all domains of the routing protocol j are saved on the control card by the j-RTM (IGP/EGP-RTM). There is a best route to reach each router on a given domain i using the protocol j. Since there are $N_{R_i}^j$ routers in the domain i connected by the protocol j, there are $\sum_{i=1}^{N_D^j} N_{R_i}^j$ best routes. Hence, the amount of memory needed to save these routes is $M \times \sum_{i=1}^{N_D^j} N_{R_i}^j$. With N_P different protocols, the memory needed is $M \times \sum_{j=1}^{N_P} \sum_{i=1}^{N_D^j} N_{R_i}^j$. Best routes of all routing protocols and all domains are backed up by the G-RTM, which requires $M \times \sum_{j=1}^{N_P} \sum_{i=1}^{N_D^j} N_{R_i}^j$ memory.
- In the second scheme, the best routes of all routing protocols and all domains are saved by the G-RTM. The proxy of each domain and each protocol keeps the best

routes of that domain. There is a best route to reach each router on a given domain i using the protocol j. Since there are $N_{R_i}^j$ routers in the domain i connected by the protocol j, the number of best routes to all routers in the domain i using the protocol j is $N_{R_i}^j$. Since the proxies are equally distributed on line cards, the memory required to save best routes on each line card in average is $(M \times \sum_{j=1}^{N_P} \sum_{i=1}^{N_D^j} N_{R_i}^j) / N_{LC}$.

The memory needed on the control card for the G-RTM to save best routes is $M \times \sum_{j=1}^{N_P} \sum_{i=1}^{N_D^j} N_{R_i}^j$. The overall best routes are updated directly by the G-RTM to the FIT located on the iNP of each line cards, therefore, it does not require memory space of the line card.

- In the third scheme, the best routes of all routing protocols in a domain are handled at the master line card. Therefore, the memory needed on a master line card is $M \times N_P \times \sum_{i=1}^{N_D} N_{R_i}$. The control card keeps only the overall best routes of the system, therefore, the memory it needs is $M \times \sum_{i=1}^{N_D} N_{R_i}$.

Table 7-1 summarizes the overhead needed to compute new routes in the centralized architecture and our proposals. The numbers should be very similar for most IGP protocols.

7.4.4 Observations

The performance evaluation of the three proposed schemes based on the computation above is illustrated in Figure 7-13 and Figure 7-12 according to the following router configuration parameters:

	Centralized	Distribution of routing protocols	Distribution of LC-RTMs
Number of messages exchanged through switch fabric	$N_{LC} \times \sum_{j=1}^{N_P} N_D^j$	$N_{LC} \times \sum_{j=1}^{N_P} N_D^j$	$(N_P + 1) \times (N_{LC} - N_D)$
CPU Consumption			
CPU cycles on the control card	$(\sum_{j=1}^{N_P} \sum_{i=1}^{N_D^j} N_{C_i}^j) +$ $($ $(\overline{N}_{Sel} \times \sum_{j=1}^{Np} N_D^j) / N_P$ $) +$ $(\overline{N}_{Cmsg} \times \sum_{j=1}^{N_P} \sum_{i=1}^{N_D^j} P_i^j)$	$((\overline{N}_{Sel} \times \sum_{j=1}^{Np} N_D^j) / N_P) +$ $(\overline{N}_{Cmsg} \times \sum_{j=1}^{N_P} N_D^j)$	$\overline{N}_{Cmsg} \times N_D$
Average CPU cycles on proxy line card	N/A	$((\sum_{j=1}^{N_P} \sum_{i=1}^{N_D^j} N_{C_i}^j) / N_{LC}) +$ $($ $(\overline{N}_{Cmsg} \times \sum_{j=1}^{N_P} \sum_{i=1}^{N_D^j} (P_i^j - 1)) / N_{LC}$ $)$	$((\sum_{j=1}^{N_P} \sum_{i=1}^{N_D^j} N_{C_i}^j) / N_{LC})$ $+$ $($ $\overline{N}_{Cmsg} \times (N_{LC} / N_D - 1)$ $)$
CPU cycles on master line card	N/A	N/A	$\overline{N}_{Sel} + \overline{N}_{Cmsg} \times N_P$

Memory Requirements			
Memory required on control card	$2\times(M\times\sum_{j=1}^{N_P}\sum_{i=1}^{N_D^j}N_{R_i}^j)$	$M\times\sum_{j=1}^{N_P}\sum_{i=1}^{N_D^j}N_{R_i}^j$	$M\times\sum_{i=1}^{N_D}N_{R_i}$
Memory required on line card	N/A	$(M\times\sum_{j=1}^{N_P}\sum_{i=1}^{N_D^j}N_{R_i}^j)/N_{LC}$	$M\times\sum_{i=1}^{N_D}N_{R_i}$
Memory required on master line card	N/A	N/A	$M\times N_P\times\sum_{i=1}^{N_D}N_{R_i}$

Table 7-1: Performance Comparison of the Centralized and the Distributed RTM Architectures

- The number of line cards the router supports (N_{LC}). The more line cards are added, the higher connectivity the router has.
- The number of interfaces (ports) located on a line card ($\overline{N}_{Port}$). They are optical interfaces with high capacity (10-40Gbps). In our evaluation, we use the configuration of 10 ports per line card.
- The number of routing protocols (N_P) currently running on the router. In practice, a router may support RIP, OSPF, IS-IS, BGP, MPLS, LDP and RSVP.

The number of messages going through the switch fabric is almost the same on the centralized architecture and the second schemes. In the centralized architecture, link notification messages received by all line cards are forwarded to the control component on the control card through the switch fabric. In the second scheme they are forwarded to the proxy of each domain and only the best routes are sent to the G-RTM on the control card. The number of messages traveling the switch fabric in the third scheme is smaller because each line card uses all of its ports to connect to a domain therefore the routing protocol module on a line card sends only one link notification message to the master.

In the centralized architecture, no CPU and memory resource is required on the line cards for route management, because the RTMs are implemented only on the control card. Therefore we compare in Figure 7-13 only the CPU and memory used for route management on the control card (CC) in the centralized architecture with the control card and the line cards (LC) in the second scheme and with the control card and the master line cards in the third scheme. There are respectively 2 and 4 protocols running on the system.

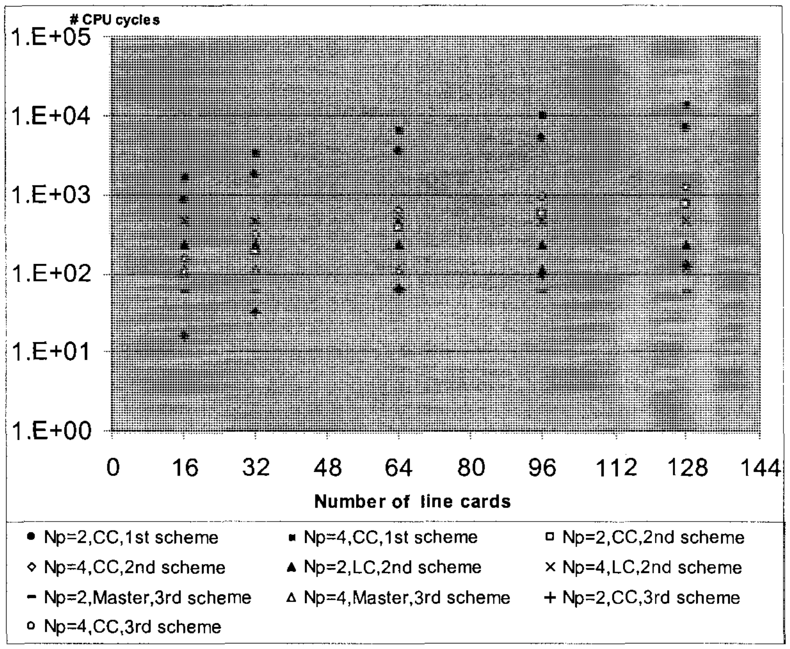


Figure 7-12: Comparison of CPU Consumption between the Centralized and the Proposed Distributed Architectures

Figure 7-13 shows that the memory needed on the control card in the centralized architecture is largest, followed by the second and the third scheme. It is due to the fact that the centralized architecture saves two copies of all best routes: one is handled by the

G-RTM, the other is distributed on IGP/EGP-RTMs. The second scheme balances the memory utilization on all line cards by a good proxy distribution. The third scheme transfers the memory consumption from the control card to the masters so with a small number of line cards, the memory used in the masters is more than in the control card. With a large number of line cards, memory used for the route management can be shared equally between control and line cards.

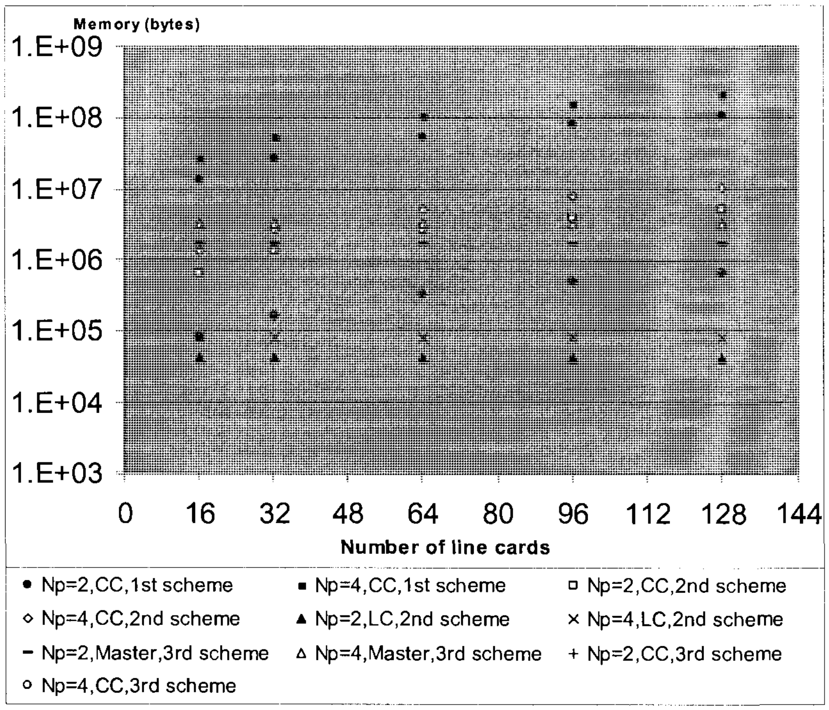


Figure 7-13: Memory Requirements in the Centralized and the Proposed Distributed Architectures

We also observe the following:

- As the number of line cards increases, the resource requirements increase in the control card for all three architectures.
- As the number of protocols increases, the resource requirements increase in the control card for the centralized architecture and Scheme 2. It is not the case, however, for the third scheme because the control card is not involved in the

best route computation and the overall best routes selection. In the third scheme, different protocols share the same domain so the number of overall best routes does not change when more protocols are activated.

- As the number of line cards increases, the resource requirements on the line cards in the second and third scheme do not increase if we have a good proxy selection mechanism where a proxy serves an unchanged number of line cards.
- As the number of protocols increases, the resource requirement in the line cards increases for schemes 2 and 3.

Since the ultimate goal of a scalable RTM distribution scheme is to support the CSPF computation of RSVP-TE routes, Scheme 3 is the best candidate for our requirement because it allows the reduction of the number of messages exchanged between line cards and the control card. Hence in the following sections, we will present an implementation model for Scheme 3.

7.5 RTM Distribution and Added Values for Path Computation

Traditionally, the RTM is handled by control cards where it can obtain routing information (e.g., link state, route exchange) directly from routing protocols such as OSPF and BGP. Best routes are computed and recorded into the Forwarding Information Table (FIT) located on line cards through services provided by the IP stack. Towards a distributed approach where routing protocols can be distributed between line cards and control cards, the RTM should be distributed between line cards and control cards in order to reduce the load on the control card and the number of messages exchanged between control cards and line cards. In addition, a distributed architecture for RTM allows the forwarding process to be achieved independently on the line cards. A local RTM (LC-RTM) located on each line card can update routes instantly to the FIT used by the local IP stack and local Network Processors (NP) located on the same line card. LC-RTM contains appropriate information used for routing protocols, such as OSPF or IS-IS, and signaling protocols, such as RSVP-TE, running on the same line card. Path computation can be performed at the line card level, making the distributed process to be highly scalable and robust.

However, such a distributed approach has to take into account the following challenges.

- The number of message exchanges between line cards and control cards can be large: in order to update routing tables and to compute best routes, RTMs (the G-RTM and LC-RTMs) need to exchange routing information. In the traditional architecture (e.g., with only G-RTM located at the control card level), there are only communications between the set of line cards and control cards. According to the distributed architecture, communications among the line cards are required. This can be done with the control card used as a relay point. However, this solution will lead to additional load on the control card. The solution we are considering is to have a special communication channel for line cards which shares the bandwidth on the switch fabric with data flows.
- Resource consumption at line card level (i.e., memory and processing resource): memory resource is required to contain LC-RTM tables (i.e., link state databases). Some CPU cycles are also needed to be reserved for path computation and access/update operations.

- The consistency among the LC-RTMs: it is required that LC-RTM contents must be consistent in some given context (i.e., for line cards belonging to a same domain). This can be done through appropriate synchronization mechanisms, but they entail some resource consumption, which needs to be evaluated.

In addition, a distributed architecture of the RTM has to perform the following tasks.

- Receive route update notifications from routing protocols. Routing protocols are responsible for computing best routes using appropriate algorithms and message exchange mechanisms. Best routes are then sent to RTM in order to update the IP forwarding table (FIT).
- Send new route advertisements to routing protocols. Routing protocols must be aware about new computed routes in order to update their databases and to inform neighbors. Therefore after having computed the new routes, the RTM has to advertise the appropriate routing protocols.
- Path computation (best route or TE-based path). There are basically two computations done by the RTM. The first is best route computation where the RTM has to select the best routes coming from different protocols in order to update the FIT. The selection criterion is usually based on the administrative distance. The second computation deals with traffic engineering (TE) paths where the RTM has to determine the appropriate LSPs based on QoS parameters of the links, such as administrative weights.
- QoS/TE route definition. The RTM is responsible for getting static route configurations from users, as well as managing route policy.

where the path computation is the most important and complex task. In the following paragraph, we will investigate the ability of distributing the path computation function to line cards with the LC-RTM. The other tasks will be discussed together with the proposed architecture for RTM.

On the line card, the LC-RTM can be designed hence as a Path Computation Element (PCE) [Farr06] which currently serves the RSVP-TE path computation requirement but also can be extended to other protocols such as OSPF or BGP. Since the current signaling protocols (RSVP-TE and LDP) are able to run independently on line cards, the LC-RTM module consists of two main components (Figure 7-14):

- Traffic engineering database (TED): contains the topology and resource information of the domain. The TED may be fed by an IGP protocol instance running on the same line card or on the control cards.
- Path computation element (PCE): achieves the path computation based on a network graph and applies computational constraints during the computation. We investigate the distributed path computation model in the inter-domain, intra-domain and inter-layer context.
    - Inter-domain path computation may involve the association of topology, routing and policy information from multiple domains. This can be done at the LC-RTM level.
    - Intra-domain path computation deals with the routing information coming from a single domain. This is achieved by routing protocols running on the line cards, such as OSPF or IS-IS.

- Inter-layer path computation aims at performing the path computation at one or multiple layers while taking into account topology and resource information at these layers. This is achieved by the LC-RTM and local QoS (L-QoS) modules.

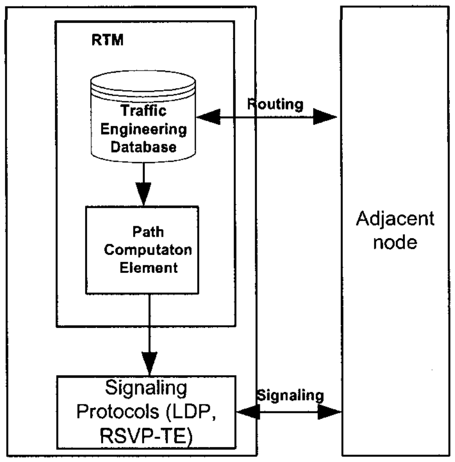

RTM
Traffic Engineering Database
Routing
Path Computaton Element
Adjacent node
Signaling Protocols (LDP, RSVP-TE)
Signaling

Figure 7-14: Overview of the Proposed RTM Architecture

The path computation can be done with the help of a distributed RTM architecture, which consists of a G-RTM located on the control card and the LC-RTMs running on line cards. LC-RTM contains information of a domain while G-RTM keeps information of all routes going through the router. The LC-RTMs can exchange link state information in order to establish traffic engineering routes. The process can be described as follows.

- The RSVP-TE module on the ingress line card of the router receives a PATH message from the upstream router.
- The RSVP-TE module on the ingress line card checks the admission status (grant/deny) for the new request based on information in the TED.
- The LC-RTM computes the next hop (downstream) router using the PCE and the traffic engineering database

- In case of inter-domain path computation, the request is sent to the master of the domain which is able to build the inter-domain topology with other domains.
- In case of intra-domain path computation, routing protocol modules running on the same line card are invoked.
- In case of inter-layer path computation, the PCE uses information contained in the traffic engineering database.

- The egress line card connecting to the downstream router is contacted in order to forward the PATH message.

7.6 Implementation of the Distributed RTM Architecture

Our distributed model for RTM has two main components (Figure 7-15):

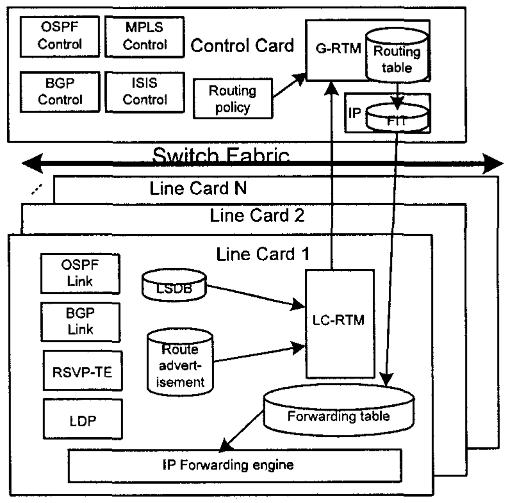


Figure 7-15: Distributed RTM Architecture

- Each line card handles a LC-RTM process. The LC-RTM obtains link state update information from routing protocols which are running on the same line card, and computes the best routes for each network domain of the line card. This task can be achieved by exchanging information among the line cards and it may need the help of the control card in order to make routing decisions at the platform level (e.g., selection of routes based on user decisions). For example, some routing protocols need a global view of the network topology provided by the control card in order to compute the best route.

- G-RTM to run on a control card and be responsible for getting routing information from LC-RTMs and updating the forwarding table of the router. The G-RTM is also the interface with the user (through an external routing policy module) by which the user can define static routes, TE or QoS-based routers. Additional control cards can be added to share processing tasks or to save backup information of the G-RTM used for resiliency purpose. However, load balancing and G-RTM resiliency are not taken into account in this thesis. We suppose also that there is a routing policy module located on the control card that allows users to configure route filtering policies and OSPF/BGP inter-working, and to modify path attributes for BGP routing protocols according to specific pre-configured policies.

We investigate the ability of using a distributed model proposed for RTM based on the following aspects:

Receiving route update notification from routing protocols. The RTMs, i.e., G-RTM and all LC-RTMs, must be notified of the changes in the routing information generated by the underlying routing protocols (i.e., RIP, OSPF, IS-IS, BGP) or by the user. In our proposal, we assume that LC-RTMs are updated through exchanges with G-RTM. Best routes and/or TE/QoS-based routes can be re-computed and the forwarding table can be updated.

Sending new best route advertisements to routing protocols. The RTMs must send an alert message to the routing protocols about the current state of physical links, such as the up/down status, the available bandwidth, etc. This information helps the routing protocols when flooding QoS-related information to the routing domain or building QoS forwarding tables

Path computation. Best or TE/QoS-based routes are computed based on information collected from routing protocols and users. When the RTM is distributed on the line cards, the results given by each process must converge for the whole platform, i.e., information provided by the set of processes is consistent or otherwise inconsistencies must be addressed and resolved.

QoS and traffic engineering. QoS and TE-based routes are established for specific connections and replace the existing best routes. These routes can be defined by the user through a command line interface running on the control card or by QoS-enabled protocols such as RSVP-TE or LDP-CR.

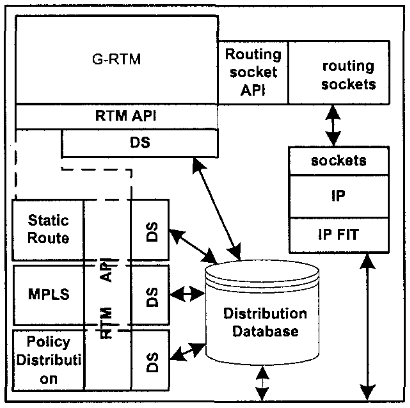


Figure 7-16: Architecture of G-RTM on the Control Card

Figure 7-16 and Figure 7-17 present the architectures we propose for G-RTM and LC-RTM respectively. The communication between LC-RTMs located on line cards and the

G-RTM located on the control card or among LC-RTMs is achieved by the Distribution Services (DS) channel. The G-RTM has an interface with routing socket by which it is able to record the forwarding information table (FIT) through services provided by the IP stack. Route update information can be received from neighbor routers through interfaces between LC-RTM and routing protocols running on the same line card (Figure 7-16). Also, route advertisement can be sent out to the external world using the same interface.

The functions provided by the G-RTM and the LC-RTMs are implemented as APIs. They include the store, access, lookup, list, remove, update and backup functions. Each function is represented by a Type-Length-Value (TLV) structure. A module, e.g., MPLS, can execute an RTM function by sending a message containing this data structure to the G-RTM or LC-RTM. The Type field is the name of the operation, followed by the length of the structure; the Value field contains additional information for the function, such as the parameters to be processed.

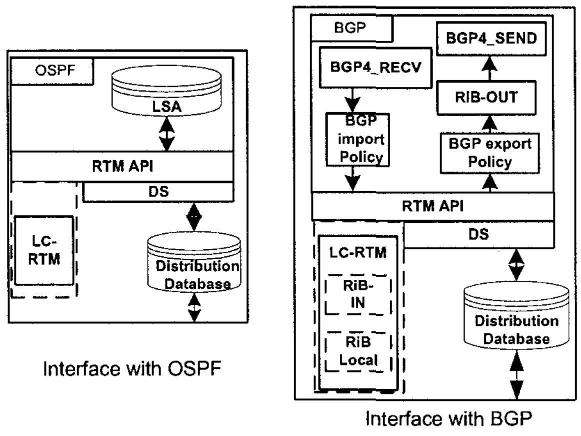


Figure 7-17: Architecture of LC-RTM on a Line Card

Figure 7-17 shows the interface between the LC-RTM on line card with the routing protocols, namely OSPF and BGP. For the BGP, the LC-RTM has two tables. The RIB-IN (*Routing Information Base – Input*) handles the routes advertised by BGP neighbor routers (so called BGP speakers). The RIB-LOC (*Routing Information Base – Local*) contains the routes the router discovers by itself (e.g., physical links of the line card or the routes learnt by other protocols like OSPF). By combining these two tables, the LC-RTM determines the best routes for BGP, which are stored in the RIB-OUT (*Routing Information Base – Output*) table, taking into account the additional user policy configurations. The RIB-OUT table is then advertised to the BGP neighbor routers.

The LC-RTM has also access to the Link State Database (LSDB) managed by OSPF as shown in Figure 7-17 (a). This allows the OSPF to be updated with the route changes and the link status information managed by the LC-RTM. The OSPF best route computation is achieved by the OSPF module so the LC-RTM is not involved in this process. However, the final results will be stored in the routing table through the RTM API services.

Basically, the distributed architecture we propose achieves the essential functions as follows.

Receive route update from routing protocols

In the proposed architecture, link state databases (LSDB) are stored on line cards, made locally available to be accessed by the requesting processes such as IGPs, LC-RTM or RSVP. Recall that the link state database is domain specific or network specific. In the centralized model, the link state database is handled by the control card, hence synchronization is not required. In the proposed distributed model, we have to make sure that all line cards connecting to a domain or network maintain the same link state database. This can be achieved by having a line card acting as a "proxy" assuming the path computation for each domain or network. When a line card in the given domain receives a link state notification message, it has to forward the message to the corresponding proxy line card of the domain. The proxy updates its database and then synchronizes the associated line cards. An appropriate election mechanism for the proxy line card is required for each domain or sub-network the router connects to. In order to simplify the architecture, we can assign the first line card on which the routing protocol is activated as the proxy for that domain. After having computed the best routes, the protocol module has to send update information to the LC-RTM located on the same line card. The LC-RTM will update its best route table with this information and advertise to other line cards in the same domain.

Advertisement

The LC-RTM has an interface with the IP stack, by which it is able to detect changes on physical links of the line card. Local routing protocols are then updated to perform the further processing (i.e., re-compute the paths or inform neighbors). Also, when the LC-RTM is updated by a routing protocol, it has to re-compute the overall best routes and update the other protocols running on the same domain.

Path computation (best route and TE-based route)

The path computation is processed on routing protocol basis. According to a distributed architecture of routing protocols, best routes can be computed in a distributed way. For link-state based protocols (e.g., OSPF, IS-IS), path computation can be performed by the "proxy" line card of the domain or sub-network. On the other hand, distance vector based protocols have to send the route update information they obtain from neighbors to the master line card of each cluster in order to perform the computation. Basically, the path computation process proceeds as follow:

- The routing protocol module receives update information from neighbors or detects local link modifications by itself.
- Based on the protocol identification, it decides to send the notification to the master line card (i.e., in case of distance vector based protocols) or forward this information to the corresponding proxy line card (in case of link state protocols).
- The PCE of the master line card or an appropriate "proxy" line card runs specific algorithms (e.g., Dijkstra for link-state based protocols or Bellman-Ford for distance vector based protocols) to build the network topology and produce the best routes.
- New route or update route will be registered in the forwarding tables located on each line in the cluster through the local LC-RTMs.

New best routes are then also advertised to all routing protocols in order to update their tables.

In practice, administrative weights are specified by users through the interface between the G-RTM and the Routing Policy module.

QoS and traffic engineering specification

The model we propose provides user QoS and traffic engineering specification functions through an interface between the routing policy module and the G-RTM located both on the control card. QoS and TE-based routes can also be established using specific protocols like RSVP-TE or CR-LDP. In that case, specific parameters will be updated first to LC-RTM, then the computed routes will be updated to G-RTM.

Migrating some processing tasks from the control card to line cards helps to reduce potential bottlenecks experienced on the control card when the number of requests increases following the increase of the number of line cards and the number of routes the core router has to support. The LC-RTM is also able to react rapidly to the physical link modification and exploit efficiently additional resources available on line cards of next generation routers. In addition, the model we propose has the following advantages:

- *Scalability*: it balances the path computation load between the control card and line cards. RTM functions are distributed as far as possible, allowing the control card to be available for more complicated tasks, such as router management and user interaction.
- *High availability*: since route information and link state databases have a back up on line cards, we provide a high redundancy level for RTMs. Also, problems arising on the control card will not slow down the processes running on the line cards.
- *Robustness*: in our architecture, the path computation is performed on the domain or sub-network instead of the whole router, which leads to a rapid convergence in case of topology changes. Routing information and notification can also arrive faster and more efficiently to the needed modules because they can be provided directly by LC-RTMs. Communication between routing protocols and RTMs is also more efficient and bandwidth of the switch fabric can be saved.

Table 7-2 presents a qualitative comparison between the centralized and distributed architecture for RTM.

Parameter	Centralized architecture	Distributed architecture
Hardware writing latency	Poor	Good
Interface with routing protocols	Good	Good
Path computing performance	Poor	Good
User management	Good	Good
Memory consumption (Control Card)	High	Low
Memory consumption (Line Card)	Low	High
Link state notification reception	Some delay	Fast
Route advertisement speed	Some delay	Fast
QoS/TE specification	Good	Good

Table 7-2: Qualitative Comparison between the Centralized and Proposed Architectures for RTM

7.7 Chapter Conclusions

In this chapter, we have presented the distributed architectures for the Routing Table Manager (RTM), which is essentially the most important component of a router. The RTM plays a decisive role for routing performance and connectivity of the network. The distribution of RTM was firstly required to serve the CSPF computation requests of a distributed RSVP-TE on line cards. Such distributed RTM architecture may also considerably reduce the load of the control card since parts of the path computation can be achieved on the line cards. We have proposed three distributed schemes to implement the RTM in the next generation router platform, including the basic distribution, the distribution of routing protocols on the line cards, and the distribution of RTMs on line cards, which are suitable for upgrading the control card, medium scale routers and very highly scalable routers respectively. The architectures we propose are able to exploit additional computing and memory resources which are available in line cards and the very high speed communication channel among line cards. The robustness, availability and resiliency of the router can also be considerably improved.

We have also performed some evaluation in order to compare the performance achieved by the distributed architectures to the centralized one where the number of CPU cycles, the memory requirements and the number of messages exchanged are taken into account.

This chapter also described the implementation architecture of the LC-RTM on the line cards and the use of such distributed architectures to compute CSPF path as RSVP-TE module requires. We have provided the designs of interfaces of LC-RTMs with routing protocols, both on control and line cards. The communication among LC-RTMs and between LC-RTM and G-RTM is also discussed.

With distributed RTM architectures, the PhD thesis has covered the distribution of principal components of the control plane of the routes: signaling protocols, routing protocols, route management and MPLS. Particularly, we have provided the details of the implementation of RTM and MPLS/LDP. The methodology to design distributed software architectures has been presented, which can be applied for other protocol modules, such as OSPF or BGP, that will be considered in our future research.

Conclusions

In this PhD thesis, we investigated the need for new architectures in order to provide QoS for distributed applications regarding the evolution of applications and hardware components. QoS architectures can be implemented on the application layer in order to provide services to end users with expected quality levels. The underlying layer supporting these QoS architectures consists of QoS-enabled devices. We therefore conducted research on the QoS provisioning architectures on both layers: application layer and underlying layer.

The first part of our two-fold research deals with the QoS provisioning at the application level where one of the main issues is the diversity and heterogeneity of QoS information on the various layers, software and hardware components. We have pointed out that QoS management architectures have to take into account all QoS information in order to make optimal QoS decisions. As the diversity and heterogeneity of service components increase over time, the proposed QDD (Quality-Driven Delivery) framework deals with the large amount of QoS information. In order to support QDD, an extensive and flexible QoS information management system has been developed to provide available QoS information adequately to needed QoS activities. The QoS information management system is based on the information models specifically designed for each service component. It allows users and providers to specify the QoS dimensions for any component that contributes to service provisioning in a distributed system. Mapping rules are built to represent relationships between user specifications and QoS dimensions of service components or among QoS dimensions of service components. The QoS management architecture we proposed in the context of QDD consists of a QoS information base that contains available QoS information and mapping rules for a given service, a QoS information manager that provides access to the QoS information base, and a QoS decision engine that computes efficient QoS decisions based on information from the QoS information manager. A user QoS requirement is achieved by one or many service components through different stages. All of this is expressed by a set of mapping rules. An efficient QoS decision can be made by walking through all available mapping rules, taking into account the deployment cost of the involved service components.

The main benefit of the approach is to avoid the *hard-coded* QoS information management that is found in the existing QoS management architectures. Mapping rules are also built in a flexible way and represented by formulas or tables, allowing the description of the QoS relationships of all service components. Since all QoS information and mapping are considered, QoS decisions are then improved, resulting in higher quality service provisioning and cost-effective distributed systems.

The second part of our PhD thesis is devoted to the QoS enabled devices, where we investigate particularly the core routers, considered as the most critical part of distributed systems. The main issue is to design routers with high scalability, resiliency and robustness. To address these requirements, the proposed distributed architecture exploits the new hardware features of the next generation routers through the distribution of control functions on all router hardware components. It results in the distribution of the specific functions of the router control plane, namely the routing, signaling and routing table management functions on router control and line cards. The starting point was the proposal of an overall generic distributed architecture for the control plane. We pushed further the distribution and developed fully distributed architectures for MPLS/LDP and RTM. We also outlined a distributed OSPF architecture. For each architecture, we reviewed the current centralized architecture, analyzed its drawbacks, and then proposed the fully distributed architecture taking advantage of the new hardware features, i.e., line cards with additional memory and computing resources and the very high switching capacity being *petabits* per second. For the MPLS/LDP, we are interested in improving the resiliency of protocol, the task sharing among line cards, and the scalability of the processing. For the RTM, we focus on the distributed computing of best routes, the storage, and the scalability of routing domains. Performance evaluation was conducted in order to compare the proposed architectures to the current ones. It showed that our proposed architectures can reduce the resource consumption, in terms of CPU cycles and memory, on the control card hence avoid potential congestions. Load balancing is also achieved among line cards.

The main benefit of this approach is the ability to fully exploit the distributed hardware architecture of the next generation routers The resiliency is significantly improved because message processing is achieved at the line card level so that the failures of the control card can be recovered transparently. The load on the control card is reduced, so the scalability of the router increases.

Contributions of the Thesis

In the first part of the thesis, we have presented a new QoS information management system based on information models. Steps to provide the QoS regarding the contribution of all service components are identified within the QDD framework. We have proposed a new methodology to specify the QoS of a distributed system based on the qualitative and quantitative QoS information. A novel classification of information models is presented where *User Models* and *Actor Models* are essential. Other QoS information models are derived from these basic models and include specific QoS information for each service component.

We have also established a methodology to determine the QoS mapping rules among services components. In summary, the proposed methodology is to (i) install the agent on specific components, (ii) collect and store the QoS information from agents, (iii) analyze the QoS information, (iv) generate the mapping rules using data mining techniques.

We have demonstrated that efficient QoS decisions can be made only if all QoS information and mapping rules of all service components are taken into account. We also proposed a case study with a video delivery system in order to illustrate and validate the proposed QoS information system. It leads to a video application that is able to provide service with quality in a flexible way, taking into account the utilization cost of various system components.

In the second part of the thesis, our contributions include a general distributed and scalable framework for the control plane of next generation routers; a distributed architecture for MPLS/LDP; and three distributed architectures for RTM. Based on our proposal of a general distributed control plane architecture, we have defined mechanisms allowing routing and signaling protocols to be processed in a distributed manner. A novel approach of distributed processing based on the routing domain configuration has been proposed. A distributed OSPF architecture has also been outlined.

The distributed architecture for MPLS/LDP fully complies with the RFC specifications. In addition, the protocol processing is achieved totally in a distributed platform. Processing mechanisms have been completely redefined in order to deal with the message synchronization between line cards, synchronization between routing tables and MPLS tables, distributed LSP storage, distributed label provision, distributed table access and update, LSP establishment with multiple sources and data recovery in case of failures. The result is not only a novel distributed MPLS/LDP implementation architecture that is able to exploit the next generation router platform, but also involves a new approach to design distributed signaling protocols. Both data and control planes are considered, allowing significant improvement of the scalability and resiliency of routers.

The new proposed distributed architectures for RTM have been designed to do route computation efficiently in very large scale routers, where best routes of each autonomous system are computed independently on different locations of the router. Different distributed schemes have been explored and compared. In particular, we discussed and identified for each scheme the best network/traffic environment where it should be deployed. We also provided the description for interfaces between the RTM and different routing protocols, allowing the proposed distributed architectures to fit in the existing software platforms. Comparative performance evaluations have been conducted for the centralized and proposed distributed architectures where comparisons have been made in terms of the number of CPU cycles, memory requirements and messages exchanged.

Lessons Learned

For the QoS provisioning at the application layer:

- *Current distributed system components and user requirements, especially those related to QoS, are evolving over time, thus QoS management architectures should be built in an extensible manner.* For example, in current QoS management architectures, the QoS information management is defined in a *hard-coded* manner so it is not able to dynamically update the evolution of the system. Our research has been conducted to address this issue.
- *A modeling approach can be used to deal with extensible systems such as distributed multimedia systems.* In this thesis, we have used a modeling approach to better manage QoS information. Generic models help to give an overview of the system and specific models provide details about system components. In fact, it is difficult to build specific QoS information models for all service components of an extended system. However, the more information models we have, the better QoS decision can be made.
- *It is hard to build a mapping rule among different QoS dimensions using mathematical formulas, particularly for user-defined dimensions or new dimensions*. Mapping rules can be defined from statistical information using the tabular tool.
- *In cases where the service quality is degraded, the QoS manager should seek alternative solutions to maintain the most important user requirements while keeping the cost unchanged.* Current service providers are more interested in the

fast or easy-to-implement solutions (e.g., increase the bandwidth) than cost-effective solutions (e.g., change the codecs). Therefore, users are often asked to upgrade their systems.

For the architecture of next generation routers:

- *A software architecture should be able to fully exploit the capacity of the hardware platform*. This helps to increase the overall performance of the router. Several router software architectures have been developed in general-purpose environments. Therefore they do not take full advantage of the new advanced features of router platforms.
- *Due to the evolution of processor and memory manufacturing industries, the main issues of the current routers are moving towards scalability and reliability instead of performance*. According to many studies in this field [Chao07], current core routers will be replaced in few years by more powerful ones. However, keeping them functional with zero-downtime is still a big challenge for engineers. Due to additional processors and memory chipsets, routers are provided with more cards, raising problems related to current software design and management.
- *Routers should be designed in a modular manner, so that failures can be isolated.* A good design has to make sure that failures at the control plane level do not lead to the shutdown of the data plane. Additional backup cards should be used in order to increase the resiliency.

- *Task sharing should be used as much as possible to reduce congestion.* The next generation routers are expected to have thousands of line cards and to be designed in a distributed manner. This is a good suggestion for task-sharing based architectures. Since the volume of data to be processed is very large, congestion can be experienced in different locations of a router. Thus centralized processing architectures cannot be used.
- *Router functions, such as routing, signaling, routing table management or data forwarding, may be distributed in different ways.* There are different possible distributed architectures, leading to different scalable and resiliency levels. We have to investigate the nature of each function in order to propose an appropriate distributed architecture. Resource consumption, complexity of the architectures and development costs need to be taken into account.

Future Work

For the QoS provisioning at the application layer:

- *Developing the information models for a wider range of service components.* In this thesis, we demonstrated the usefulness of a QoS information management approach for a video streaming application with a limited number of service components, namely streaming servers, video provider and clients. A commercial system may include more components running on different platforms. In the future, we would like to investigate the problem on a peer-to-peer video streaming system with hundred of nodes where each node can be considered as a service component with various configurations.

- *Investigating the algorithms to make optimal QoS decision, taking into account a larger number of mapping rules and QoS dimensions.* Since the QoS provisioning for contemporary services should be achieved dynamically, the QoS decision is expected to be made in the shortest possible time. We may therefore consider techniques producing solutions using the dynamic programming, approximation or distributed computing.

For the distributed and scalable software architecture of routers:

- *Proposing distributed architectures for other protocols, such as OSPF or BGP.* This thesis proposed new distributed architectures for MPLS/LDP and RTM and outlined one for OSPF. A complete distributed control plane for core routers should also consider IS-IS, BGP and RSVP as essential protocols. We can also apply the general distributed architecture for these protocols regarding their specific functions. For OSPF, we should consider proxy selection mechanisms, neighbor management, designated router and backup designated router elections, physical link management and synchronization among LSDB (Link State Database) of line cards within a domain. For BGP, we are interested in balancing the load between active and backup control cards, and restoring information of BGP sessions from the TCP stack of the router.
- *Developing resource reservation mechanisms for MPLS.* Actually, the LDP architecture presented in this thesis does not deal with the resource reservation issue. Bandwidth and other QoS parameters are allocated using RSVP-TE, which is still centralized in most products in the market. One of the next

objectives of our research is to develop the distributed architectures for RSVP-TE and CR-LDP (*Constrained Routing LDP*). One of the main challenges for this work is the design of a local QoS module on each line card, which is able to interact with network processor (NP) and traffic manager (TM) chipsets. Sharing QoS information between ingress and egress line cards should also be taken into account. Mapping different IP traffic to appropriate QoS levels is also an issue. Additional mechanisms should be defined to allow MPLS to deploy LDP and RSVP-TE protocols alternatively.

- *Investigating algorithms to group the line cards into domains and to assign the proxy or master line card in an optimal way, so that the load on the line cards can be balanced.* Indeed, if the size of a routing domain is too large, two or more proxies can be deployed to share the computing task. On the other hand, a proxy can serve several domains if their sizes are small. We consider also developing distributed algorithms to compute the best routes of a domain in parallel in all line cards belonging to that domain.
- *Implementing some prototypes of distributed software for commercial routers of the next generation.* This process should be done with the help of industrial equipment so we can observe the behaviors of software architectures in the context of specific hardware platforms. Several challenges should be addressed to implement the proposed architecture according to industrial development standards. Our goal is to develop a set of distributed prototypes for OSPF, BGP and MPLS modules, which are able to run on next generation router products.

The performance test in working environments with a large number of network nodes and large traffic volume can also be considered.

Finally, we would be interested in exploring the combination of two research topics, where QoS information models and mapping rules of the first research topic can be used similarly in the second one in order to manage the tables and to map different traffic, such as MPLS and IP.

Bibliography

[Abde99] Abdelzaher, T.F., Bhatti, N., "Web Content Adaptation to Improve Server Overload Behavior", Computer Networks, Vol. 31, No. 1116, pp. 1563-1577, May 1999.

[Abde03] Abdelzaher, T.F., Shin, K.G., Bhatti, N., "User-level QoS-adaptive Resource Management in Server End-Systems," IEEE Transactions on Computers, Vol. 52, Issue 5, pp. 678- 685, May 2003.

[Agar03] Agarwal, D., González, J., Jin, G., Tierney, B., "An Infrastructure for Passive Network Monitoring of Application Data Streams", PAM, April 2003.

[Ande01] Andersson, L., Doolan, P., Feldman, N., Fredette, A., Thomas, B., "LDP Specification," RFC3036, IETF - Network Working Group, January 2001.

[Aol07] AOL Video on demand, http://video.aol.com (Last access: 31/08/2007)

[Aurr98] Aurrecoechea, C., Campbell, A.T., Hauw, L., "A Survey of QoS Architectures," Journal of Multimedia Systems, Vol. 6, No. 3, pp. 138-151, May 1998.

[Avic06] Avici Systems Inc., "The Avici TSR®," 2006. http://www.avici.com/products/tsr.shtml (Last access: 31/08/2007)

[Awdu01] Awduche, D., Berger, L., Gan, D., Li, T., Srinivasan, V., Swallow, G., "RSVP-TE: Extensions to RSVP for LSP tunnels," RFC 3209, IETF - Network Working Group, December 2001.

[ATIS01] Alliance for Telecommunications Industry Solutions (ATIS), "Objective Video Quality Measurement using a Peak Signal-to-Noise Ratio (PSNR)

Full Reference Technique", ATIS Technical Report T1.TR.74 – 2001, October 2001.

[Awey01] Aweya, J., "IP Router Architectures: An Overview," International Journal of Communication Systems, Vol. 14, Part 5, pp. 447-476, 2001.

[Bake97] Baker, F., Krawczyk, J., Sastry, A., "Integrated Services Management Information Base Guaranteed Service Extensions using SMIv2," RFC 2214, IETF - Network Working Group, September 1997.

[Bu04] Bu, T., Gao, L., Towsley, D., "On Characterizing BGP Routing Table Growth," Computer Networks, Vol. 45, Issue 1, pp. 45-54, 2004.

[Casa05] Casale, G., "Combining Queueing Networks and Web Usage Mining Techniques for Web Performance Analysis," Proceeding of ACM Symposium on Applied Computing, pp. 1699-1703, March 2005.

[Cbc07] CBC Video on demand, http://www.cbc.ca/cbcondemand/index2.html (Last access: 31/08/2007).

[Chao02] Chao, H.J., "Next Generation Router," IEEE Communication Magazine, Vol. 90, No. 9, pp. 1518-1558, September 2002.

[Chao07] Chao, H.J., Liu B., "High Performance Switches and Routers," Wiley-Interscience, USA, 2007.

[Cisc05] Cisco Systems, "Cisco 12000 Series Internet Router Architecture," http://www.cisco.com (Last access: 31/08/2007)

[Corm01] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein C., "Introduction to Algorithms, Second Edition," The MIT Press, 2001.

[Deca00] Decasper, D., Dittia, Z., Parulkar, G., Plattner, B., “Router Plugins: A Software Architecture for Next-Generation Routers,” IEEE/ACM Transactions on Networking, Vol. 8, Issue 1, pp. 2-15, February 2000.

[Deva03] Deval, M., Khosravi, H., Muralidhar, R., Ahmed, S., Bakshi, S., Yavatkar, R., “Distributed Control Plane Architecture for Network Elements”, Intel Technology Journal, Vol. 7, Issue 4, pp. 51-62, November 2003.

[DMTF07] DMTF Inc, “CIM Schema: Version 2.15,” April 2007.

[Dori07] Doria, A., Haas, R., Salim, J.H., Khosravi, H., Wang, W. M., “ForCES Protocol Specification”, IETF Draft, Work in Progress, IETF - Network Working Group, July 2007.

[Dupl05] Duplaix, J., “Routing Software Architecture in the R1.0 PBR-1280 Router,” Internal Document, HyperChip Inc., October 2005.

[Duza04] Duzan, G., Loyall, J., Schantz, R., Shapiro, R., Zinky, J., “Building Adaptive Distributed Applications with Middleware and Aspects,” Proceeding of The 3rd International Conference on Aspect-Oriented Software Development, pp. 66-73, UK, 2004.

[Farr06] Farrel, A., Vasseur, J.-P., Ash, J., “A Path Computation Element (PCE)-Based Architecture,” RFC 4655, IETF - Network Working Group, August 2006.

[Fost01] Foster, I., Roy, A., Sander, V., “A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation,” Proceeding of the 8th International Workshop on Quality of Service - IWQOS, pp.181-188, May 2001.

[Frol98] Frolund, S., Koisten, J. , "QML: A Language for Quality of Service Specification," HP Technical Repport, 1998.

[Gerb03] Gerbé O., Kerhervé B., Srinivasan U., "Model Operations for Quality-Driven Multimedia Delivery," In the Proceeding of ICCS 2003, International Conference on Conceptual Structures (ICCS2003), Germany, July 2003.

[Gu01] Gu, X., Nahrstedt, K., Yuan, W., Wichadakul, D., Xu, D., "An XML-based QoS Enabling Language for the Web," Journal of Visual Language and Computing (Special Issue on Multimedia Languages for the Web), Vol. 13, No. 1, pp.61-95, 2001.

[Gill05] Gill, C., Gossett, J.M., Corman, D., Loyall, J.P., Schantz, R.E., Atighetchi, M., Schmidt, D.C., "Integrated Adaptive QoS Management in Middleware: An Empirical Case Study," Journal of Real-time Systems, Vol.29, No. 2-3, pp. 101-130, 2005.

[Gu05] Gu, X., Nahrstedt, K., "Distributed Multimedia Service Composition with Statistical QoS Assurances," IEEE Transactions on Multimedia, Vol. 8, Issue 1, pp. 141- 151, 2005.

[Hafi99] Hafid, A., Bochmann, G., "An Approach to QoS Management in Distributed Multimedia Applications: Design and an Implementation," Multimedia Tools and Applications, Vol. 9, No. 2, 1999.

[Hala05] Halabi, S., "Internet Routing Architecture," Second Edition, Cisco Press, 2005

[Hand05] Handley, M., Kohler, E., Ghosh, A., Hodson, O., Radoslavov, P., "Designing Extensible IP Router Software," Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2005.

[Harr99] Harrington, D., Presuhn, R., Wijnen B., "An Architecture for Describing SNMP Management Frameworks," RFC 2571, IETF - Network Working Group, April 1999.

[Hide06] Hidell, M., "Decentralized Modular Router Architecture," Doctoral Thesis, KTH – Royal Institue of Technology, TRITA-IT R 96:07, Sweden, September 2006.

[Huar97] Huard J.F., Lazar A.A, "On QOS Mapping in Multimedia Networks," Proceeding of COMPSAC '97 - 21st International Computer Software and Applications Conference, pp. 312-323, May 1997.

[Hype04] Hyperchip Inc., "PBR-1280 Release 1.0 Performance Characterization System Engineering," Technical Report, 2004.

[ICS00] ICS Inc., "Enterprise Process Management Framework – A definition of Service Level Management," White paper, Intelligent Communication Software, 2000.

[ISO02] ISO, "Intermediate System to Intermediate System Routing Information Exchange Protocol for use in Conjunction with the Protocol for Providing the Connectionless-mode Network Service," ISO 8473, ISO/IEC 10589:2002, Second Edition.

[Jin04] Jin, J., Nahrstedt, K., “QoS Specification Languages for Distributed Multimedia Applications: A Survey and Taxonomy,” IEEE Multimedia Magazine, Vol. 11, Issue 3, July 2004.

[Juni07] Juniper Networks Inc., “Juniper Networks Next Generation Network Core Solution for Service Providers,” Solution Brochure, June 2007.

[Kapl02] Kaplan, H. “Non-Stop Routing Technology,” White Paper, Avici Systems Inc., 2002.

[Kerh06] Kerhervé, B., Nguyen, K.K, Gerbé, O., Jaumard, B., “A Framework for Quality Driven Delivery in Distributed Multimedia Systems”, Proceeding of ICIW’06, France, pp. 195-202, February 2006.

[Koli02] Koliver, C., Nahrstedt, K., Farines, J.M., Fraga, J.S., Sandri, S.A., “Specification, Mapping and Control for QoS Adaptation,” Journal of Real-Time Systems, Vol. 23, No 1-2, pp.143-174 , July 2002.

[Lee99] Lee, C., Lehoczky, J., Siewiorek, D., Rajkumar, R., Hansen, J., “A Scalable Solution to the Multi-Resource QoS Problem,” Proceeding of the 20th IEEE Real-Time Systems Symposium (RTSS '99), pp. 315-326, Phoenix, AZ, December 1999.

[Lima04] Lima, S., Carvalho, P., Freitas, V., “Distributed Admission Control for QoS and SLS Management”, Journal of Network and Systems Management - Special Issue on Distributed Management, Vol. 12, No. 3, pp. 397-426, September 2004.

[Liu04] Liu, Y., Pisharath, J., Liao, W., Memiki, G., Choudhary, A., Dubey, P., “Performance Evaluation and Characterization of Scalable Data Mining

Algorithms," Proceeding of 16th IASTED International Conference on Parallel and Distributed Computing and Systems, MIT Cambridge, MA, November 2004.

[Medh07] Medhi, D., Ramasamy, K., "Network Routing – Algorithms, Protocols, and Architectures," Morgan Kaufmann, Elsevier Edition, 2007.

[Mira02] Miras, D., "Network QoS Needs of Advanced Internet Applications", Working Document of Internet 2 - QoS Working Group, November 2002.

[Moy98] Moy, J., "OSPF Version 2," RFC 2328, IETF - Network Working Group, April 1998.

[Nahr99] Nahrstedt, K., "Quality of Service Guarantees in Networked Multimedia Systems," Handbook on Multimedia Computing, Borko Fuhrt Edition, CRC Press, 1999.

[Nahr00] Nahrstedt, K., Wichadakul, D., Xu, D., "Distributed QoS Compilation and Runtime Instantiation," IEEE/IFIP International Workshop on Quality of Service, pp.198–207, June 2000.

[Nahr01] Nahrstedt, K., Xu, D., Wichadakul, D., Li, B. "QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments," IEEE Communication Magazine, Vol. 39, Issue 11, pp. 140-148, November 2001.

[Neri07] Neri, S., "Distributed and Scalable RSVP-TE Architecture for Next Generation Routers," M.Sc. Thesis, Department of Electrical and Computer Engineering, Concordia University, September 2007.

[Nguy03] Nguyen, K.K., Kerhervé, B., Jaumard, B., "QoS Information Modelling and Transformation", Poster presented at MITACS'03, Montréal, February 2003.

[Nguy04] Nguyen, K.K., Kerhervé, B., Jaumard, B., "Modélisation et Transformation d'Informations de Qualité de Service : Une Revue de la Littérature", In Proceedings of RIVF'04, Vietnam, February 2004.

[Nguy05] Nguyen, K.K., Kerhervé, B., Jaumard, B., "Quality Driven Delivery for Video Streaming Application: Implementation and Evaluation", In Proceedings of RIVF'05, Vietnam, February 2005.

[Nguy06] Nguyen, K.K., Kerhervé, B., Jaumard, B., "QoS Mapping Rule Builder: A Model", In Proceedings of CCECE'06, Ottawa, Canada, May 2006.

[Nguy06b] Nguyen, K.K., "Incremental Architecture Description for LDP", Internal Document, HyperChip Inc., December 2006.

[Nguy07a] Nguyen, K.K., Mahkoum, H., Jaumard, B., Assi, C., Lanoue, M., "Towards a Distributed Control Plane Architecture for Next Generation Routers," In Proceedings of ECUMN'2007, pp. 173-182, France, February 2007.

[Nguy07b] Nguyen, K.K., Jaumard, B., "A Distributed Model for Next Generation Router Software," In Proceedings of HPSR'07, pp. 1-6, Brooklyn, NY, May 2007.

[Nguy07c] Nguyen, K.K., Neri, S., Jaumard, B., Agarwal, A., "Distributed and Scalable Architecture for Routing Table Maintenance," In Proceedings of DFMA'07, France, July 2007.

[Nguy07d] Nguyen, K.K., Jaumard, B., Agarwal, A., "A MPLS/LDP Distributed Architecture for Next Generation Routers," Submitted for publication to Journal of Computer Networks, December 2007.

[Nguy07e] Nguyen, K.K., Jaumard, B., Agarwal, A., "Three Distributed Schemes for Route Maintenance in Next Generation Routers," Submitted for publication to IEEE Communication Magazine, December 2007.

[Nguy07f] Nguyen, K.K., Jaumard, B., Agarwal, A., "A Distributed and Scalable Routing Table Manager for Next Generation IP Router," To appear in IEEE Network, *Special Issue: Internet Scalability: Properties and Evolution*, Mars 2008.

[Phan03] Phanse, K.S., DaSilva, L.A., "Addressing the Requirements of QoS Management for Wireless Ad Hoc Networks," Journal of Computer Communications, Vol. 26, Issue 12, pp. 1263-1273, July 2003.

[Pyun02] Pyun, J.-Y., Shim, J.-J., Ko, S.-J., Park, S.-H., "Packet loss resilience for video stream over the Internet," IEEE Transactions on Consumer Electronics, Vol. 48, Issue 3, pp. 556-563, August 2002.

[Rekh95] Rekhter, Y., Li, T., "A Border Gateway Protocol 4 (BGP-4)," RFC1771, IETF - Network Working Group, March 1995.

[Rekh01] Rekhter, Y., Rosen, E., "Carrying Label Information in BGP-4," RFC3107, IETF – Network Working Group, May 2001.

[Robu03] Robustelli, L., Loreto, S., Fresa, A., Longo, M., Spinelli, D., "Prototype of an Adaptive Voice Coder for IP Telephony," Proceeding of International Conference on Software, Telecommunications and Computer Networks –

SoftCom 2003, pp.859-863, Split Dubrovnik (Croatia), Ancona, Venice (Italy), October 2003.

[Rose01] Rosen, E., Viswanathan, A., Callon, R., "MultiProtocol Label Switching Architecture," RFC 3031, IETF - Network Working Group, January 2001.

[Scha03] Schantz, R.E., Loyall, J.P., Rodrigues, C., Schmidt, D.C., Krishnamurthy, Y., Pyarali, I., "Flexible and Adaptive QoS Control for Distributed Real-time and Embedded Middleware," ACM/IFIP/USENIX International Middleware Conference on Middleware, Brazil, June 2003.

[Schm99] Schmidt, D., Levine, D., Cleeland, C., "Architectures and Patterns for High- Performance, Real-time CORBA Object Request Brokers," Advances in Computers, Marvin Zelkowitz, Ed., Academic Press, Vol. 48, pp. 1-118, 1999.

[Schm02] Schmidt, D., Gokhale, A., Gill, C., "Patterns and Performance of Real-time and Data Parallel CORBA for High-Performance Embedded Computing Applications," Proceeding of the 6th Annual Workshop on High Performance Embedded Computing, USA, September 2002.

[Serh05] Serhani, M.A, Dssouli, R, Hafid, A, Sahraoui, H, "A QoS broker based architecture for efficient web services selection," Proceeding of IEEE International Conference on Web Services (ICWS'05), pp.113-120, July 2005.

[Shai06] Shaikh, A., Dube, R., Varma, A., "Avoiding Instability During Graceful Shutdown of Multiple OSPF Routers," IEEE/ACM Transactions on Networking, Vol. 14, Issue 3, June 2006.

[Shah01] Shaha, N., Dessai, A., Parashar, M., "Multimedia Content Adaptation for QoS Management over Heterogeneous Networks," In Proceeding of International Conference on Internet Computing (IC 2001), USA, pp. 642-648, June 2001.

[Sun07] Sun MicoSystems, Inc. "Java Media Framework API (JMF)," http://java.sun.com/products/java-media/jmf/index.jsp (Last access: 31/08/2007)

[Tse04] Tse, H.E.S., "Switch Fabric Architecture Analysis for a Scalable Bi-Directionally Reconfigurable IP Router," Journal of Systems Architecture, Vol. 50, Issue 1, pp. 35-60, January 2004.

[Vid07] Vidéotron:Television Video-on-demand, (Last access: 31/08/2007) http://www.videotron.com/services/en/television/vod-video-sur-demande-vsd.jsp

[Wang00] Wang, P.Y., Yemini, Y., Florissi, D., Florissi, P., "QoSME: Toward QoS Management and Guarantees," Proceeding of WCC-ICCT 2000 International Conference on Communication Technology, pp. 868-875, China, August 2000.

[Wang04] Wang, N., Gill, C., Schmidt, D., Gokhale, A., Natarajan, B., Loyall, J., Schantz, R., Rodrigues, C., "QoS-enabled Middleware," Chapter in Middleware for Communications, Wiley, July 2004.

[Wu01] Wu, D., Hou, T., Zhu, W., Zhang, Y.Q., Peha, J.M., "Streaming Video over the Internet: Approaches and Directions," IEEE Transaction on

Circuits and Systems for Video Technology, Vol. 11, Issue 3, pp. 282-300, February 2001.

[Yang04] Yang, L., Dantu, R., Anderson T., Gopal R., "Forwarding and Control Element Separation (ForCES) Framework," RFC 3706, IETF - Network Working Group, April 2004.

[Ye03] Ye, H., Kerhervé, B., Bochmann, G., Oria, V., "Pushing Quality of Service Information and Requirements into Global Query Optimization," Proceeding of International Database Engineering and Applications Symposium (IDEAS), pp.170-179, Hong Kong, July 2003.

[Yuan06] Yuan, W, Nahrstedt, K, Adve, S., Jones, D., Kravets, R., "GRACE: Cross-Layer Adaptation for Multimedia Quality and Battery Energy," IEEE Transactions on Mobile Computing, Vol. 5, Issue 7, pp. 79–815, 2006.

[Zhan02] Zhang L., Deering, S., Estrin, D., Shenker, S., Zappala, D., "RSVP: A New Resource Reservation Protocol," IEEE Communication Magazine, Vol. 40, Issue 5, pp. 116-127, May 2002.

[Zini02] Zini, A., "Cisco IP Routing," pp. 80-111, Addition-Wesley, 2002.

[Zink97] Zinky, J., Bakken, D., Schantz, R., "Architecture Support for Quality of Service for CORBA Objects," Journal of Theory and Practice of Object Systems, Vol. 3, No.1, pp. 1-20, January 1997.

Pending Patents

1. Nguyen, K.K., Jaumard, B., Lanoue, M., "Distributed Architecture for MPLS/LDP", Submitted in Canada, 2007.
2. Nguyen, K.K., Jaumard, B., Lanoue, M., "Distributed RTM Architecture on the Control Card", Submitted in Canada, 2007.
3. Nguyen, K.K., Jaumard, B., Agarwal, A., "Distributed Best Route Computation and Management Architecture on Line Cards", Submitted in Canada, 2007.
4. Nguyen, K.K., Jaumard, B., Agarwal, A., "Distributed Architecture of RTM on Line Cards", Submitted in Canada, 2007.
5. Nguyen, K.K., Jaumard, B., Agarwal, A., Neri, S., Lanoue, M., "Distributed Architecture for RSVPTE", Submitted in Canada, 2007.