Path Allocation in Backbone Networks: Project Report

Markus Fidler
Department of Computer Science
RWTH Aachen
fidler@i4.informatik.rwth-aachen.de

Wojciech Klimala
Central Institute for Applied Mathematics
Forschungszentrum Jülich
w.klimala@fz-juelich.de

Volker Sander
Central Institute for Applied Mathematics
Forschungszentrum Jülich
v.sander@fz-juelich.de

June 30, 2004

Contents

Chapter 1

Introduction

There are two major frameworks for providing service guarantees in IP-based networks: The Integrated Services (IS) architecture and the Differentiated Services (DS) architecture. While the IS framework provides service guarantees using a flow-based packet differentiation, the DS architecture differentiates the treatment of aggregates.

Though IS is able to provide strong Quality of Service (QoS) guarantees, its practical use suffers from significant scalability problems. The DS framework addresses this issue through its concept of aggregation. Packets are identified by simple markings that indicate the respective class. In the core of the network, routers do not need to determine to which flow a packet belongs, only which aggregate behavior has to be applied. Edge routers mark packets by setting the DS Code-Point (DSCP) and indicate whether they are within profile. If they are out of profile they might even be discarded by a dropper at the edge router. A particular marking on a packet indicates a so-called Per Hop Behavior (PHB) that has to be applied for forwarding the packet.

The price for building aggregates is, however, a weaker control over the provisioned service parameters. The derivation of delay bounds remains a challenge, when providing a Premium Service. In [14] such bounds are derived for a general topology and a maximum load. However, these bounds can be improved, when additional information concerning the current load and the special topology of the DS domain is available.

In [54], a central resource management for DS domains called Bandwidth Broker is presented. A Bandwidth Broker is a middleware service which controls and facilitates the dynamic access to network services of a particular administrative domain [31]. The task of a Bandwidth Broker in a DS domain is to perform a careful admission control, and to set up the appropriate configuration of the domain's edge routers, whereas the configuration of core routers remains static to allow for scalability. While doing so, the Bandwidth Broker knows about all service requests. Besides it can easily learn about the DS domains topology, either statically or by implementing a listener for the domains routing protocol. Thus, a Bandwidth Broker can have access to all information that is required, in order to base its admission control on delay bounds that are derived for the current load, and the special topology of the administrated domain.

The particular challenge that arises here is to derive a set of formulas that can effectively be used by the admission control procedures to decide about an additional service request based on the composition of already accepted service requests. The reservation-based approach used by the admission control procedures of the Integrated Services architecture is a foundation for this task. However, its assumptions do not apply to the scenario of aggregate scheduling.

In this context, traffic engineering capabilities offer the opportunity to control the composition of aggregates and can thus be used to provide stronger service guarantees. The Multi-Protocol Label Switching (MPLS) architecture [59, 60] is a good candidate to extend the ability to build reasonable end-to-end services based on the DS framework.

The MPLS architecture is a label switching technique which is based on a functional decomposition of the network layer into a control component and a forwarding component. Flows of packets are grouped into disjoint subsets which, from a forwarding point of view, are treated by

the routers in the same way. This logical grouping of traffic with a common destination leads to the construction of Forwarding Equivalence Classes. The members of these classes are identified by a common label. In contrast to other switching techniques such as Asynchronous Transfer Mode (ATM), MPLS does not rely on specific capabilities of the underlying link layer. Instead, it introduces the ability to add an additional label on top of the data link layer. It is therefore possible to deploy MPLS in heterogeneous environments which are built on different data link layer technologies.

The Path Allocation in Backbone Networks (PAB) was funded by the German Research Network (DFN) to explore advanced mechanisms for providing service guarantees under the particular constraints of an IP-based G-WiN-backbone. The project was performed by two contractual partners:

- Central Institute for Applied Mathematics, Forschungszentrum Jülich GmbH
- Department of Computer Science, RWTH Aachen

Additional support was provided by Cisco Systems. Cisco collaborated with the project partners and supported the project through its University Research Programme.

The evaluation was performed in two major workflows:

- Comprehensive experimental analysis of the Multi-Protocol Label Switching architecture and its potential with respect to support QoS using a Cisco-based testbed.
- Analytical modelling of service guarantees including the validation of applicability using simulation methods.

The work itself was organized in four work packages:

- Setup, validation, and operation of the testbed
- Configuration of Label Switched Paths
- Evaluation of Traffic Engineering algorithms
- Analysis of MPLS in the context of end-to-end Quality of Service

This report presents the results of the comprehensive evaluation and the software development performed within the PAB-project. For readability, it is structured to be both, a primer to the underlying methods and technology, and a documentation of the innovations achieved during the project.

Chapter 2

Background: Service Differentiation in the Internet

Emerging Grid applications exhibit complex mixes of data flows, with a wide range of data rates and widely differing latency requirements. Applications with these characteristics arise in areas as remote visualization, analysis of scientific databases, and teleimmersion. Table 2.1 lists the various flows of a future teleimmersion application together with their networking demand [22]. Characteristics such as these place substantial demands on networks which cannot be fulfilled by today's Best-Effort (BE) Internet.

Table 2.1: Flows and requirements of teleimmersion applications

	Latency	Bandwidth	Reliability	Dynamic QoS
Control	< 30 ms	64 Kb/s	Yes	Low
Text	< 100 ms	64 Kb/s	Yes	Low
Audio	< 30 ms	128 Kb/s	No	Medium
Video	< 100 ms	5 Mb/s	No	Medium
Tracking	< 10 ms	128 Kb/s	No	Medium
Database	<100 ms	> 1 Gb/s	Yes	High
Simulation	< 30 ms	> 1 Gb/s	Mixed	High
Haptic	< 10 ms	> 1 Mb/s	Mixed	High
Rendering	< 30 ms	> 1 Gb/s	No	Medium

Now consider a situation in which several teleimmersion sessions are in operation simultaneously, while other groups are concurrently attempting to perform high-speed bulk-data transfers over the same network infrastructure, perhaps to stage data required for an experiment later in the day. We receive a complex and dynamic mixture of network service requirements.

There are two basic approaches to providing network Quality of Service (QoS): reservation-based and adaptation-based. Applications using the reservation-based approach usually rely on specific capabilities of the underlying network infrastructure. They specify their QoS requirements during connection establishment and typically do not change the demand subsequently; the QoS system in turn guarantees that the reservation will not change during the lifetime of the application. From the reservation-based approach, network QoS refers to the ability of the network to handle specific packets in such a way that the related flows receive a specific type of Guaranteed Service, i.e. leaving the current Best-Effort Service.

On the other hand, applications that use the adaptation-based approach do not make a reservation but adapt to the network conditions at hand by responding to some form of feedback, whether explicit (notification of network conditions) or implicit (noticing that the achieved bandwidth is low). Instrumentation and notification are the key issues in this approach. Adaptation may occur when the application detects a problem or when the application is notified that a problem may exist. From the adaptation-based approach, network QoS refers to the ability of an application

to adapt its network resource usage to the actual state of the network in such a way that the fundamental functions of the application are still performed.

The PAB-project evaluated new mechanisms for implementing a reservation-based approach that relies on the ability of the network to provide a specific type of service. Service levels can be characterized by the following QoS-related parameters:

Bandwidth The rate at which packets of a flow are carried by the network. The specification of bandwidth guarantees includes the assurance of a peak data rate, a sustained data rate, and a minimum data rate. The latter is important, as it assures the ability to use this capacity at any time. The sustained date rate can either be a statistical guarantee, or, in the more common case, equal to the minimum date rate. The peak data rate is often specified in term of bursts relative to the sustained data rate.

Delay An upper boundary for the time it takes to send a Maximum Transmission Unit (MTU)-sized packet (see below) to traverse the network from the sender to the receiver (end-to-end delay) [1]. A more detailed definition of the Type-P-One-way-Delay can be found in [4]. For some applications it is useful to limit the delay for a full round-trip.

Jitter The variation of delay. Its formal representation is currently discussed by the Internet Engineering Task Force (IETF) as the "instantaneous packet delay variation" [23] and is the difference of the delay experienced by subsequent packets on a one-way transit from source to destination.

Reliability This parameter specifies the percentage of lost packets, errors in the network due to non-failure events, the mean-time between failures, the mean-time to repair, and the availability as percent of the uptime.

Maximum Transfer Unit Because network traffic is typically not continuous but separated in discrete packets of a maximum size, the maximum transfer unit is a service relevant parameter. First, it denotes a bound for the packet header overhead. Second, it indicates possible fragmentation delays for the application. Finally, it is relevant for packet scheduling as packets in transit are typically non-preemptive.

Today, the BE Service is the only service that is supported throughout the whole Internet. Though the BE class comprises a variety of different flows, special care has to be taken considering their co-existence. The majority of the Internet applications so far use the Transmission Control Protocol (TCP), which is responsive to network congestion by means of halving the sending data rate in case of packet loss. In contrast, if packets are successfully transmitted the data rate is increased by a constant [3]. This congestion control mechanism is supported in the network by Random Early Detection (RED) [29]. In case of an upcoming congestion RED starts early to randomly discard packets in order to force TCP connections to reduce their sending data rate and thus to avoid situations of persistent congestion, during which all incoming packets would have to be dropped. Problems arise especially, if unresponsive Real-Time Transport Protocol (RTP)/User Datagram Protocol (UDP) flows share the BE class with TCP streams, since neither RTP, nor UDP implements congestion control. Furthermore, there are often no performance incentives for implementing congestion control at application level. Thus RTP/UDP flows easily tend to steal capacity from TCP streams and can lead to an unfair distribution of the available capacity in case of congestion. Besides also greedy, and high data rate sources that apply parallel TCP sockets, like GridFTP [2], and also current Web browser implementations, can by these means lead to unfairness.

[1] Listing the upper boundary for an MTU-sized packet gives an upper delay boundary for all packets of this connection

2.1 The Integrated Services Framework

The Internet Engineering Task Force (IETF) has specified the Integrated Services (IS) Framework with the goal to provide end-to-end QoS to applications. The basic framework is comprised of two elements:

- An extended service model which is also called the IS model.
- A reference implementation framework.

The extended service model consists of two particular real-time services which are built on specific capabilities provided by all networking devices. Namely, the model assumes that each network node is capable of differentiating packets based on their service-class and of a particular preferred treatment of those packets to influence the time-of-delivery under all conditions.

The reference implementation framework states that all routers capable of supporting the IS architecture should offer three components that implement the required traffic control within the network devices:

- *The Admission Control* procedure of a router decides whether to accept or reject the requested QoS of a new flow. It is important to note that the admission control test does not provide any specific QoS, instead it declines unacceptable request and polices whether a granted request is conforming to its specification.
- *The Packet Classifier* identifies the service class and maps it into the related treatment, i.e. to the related output queue of the *Packet Scheduler*. The service specific treatment includes accounting.
- *The Packet Scheduler* manages the packet forwarding as part of the output driver of a router. It uses a set of queues and perhaps other mechanisms such as timers to schedule the order of outgoing packets.

The essence is a QoS infrastructure which offers services for individual flows which have applied for this in advance. Each router performs admission control to ensure that they only accept reservation requests when they do have sufficient local resources. Once an appropriate reservation has been installed in each router along a path of a flow, the particular treatment of the packets assures the desired level of service during the life-time of the reservation. Note that the IS framework is a flow oriented mechanism.

2.2 The Differentiated Services Architecture

The DS architecture [10] is a QoS framework that provides scalability by defining the behavior of aggregates. Packets are identified by simple markings that indicate according to which aggregate behavior they should be treated. In the core of the network, routers need not determine to which flow a packet belongs, only which aggregate behavior should be used. Edge routers mark packets and indicate whether they are within profile or if they are out of profile, in which case they might even be discarded by a dropper at the edge router. A particular marking on a packet indicates a Per-Hop Behavior (PHB) that has to be applied for forwarding of the packet. Currently, the Expedited Forwarding (EF) PHB [21] and the Assured Forwarding (AF) PHB group [38] have been specified. Though the DS architecture allows for the definition of additional PHBs by setting the 6-bit Differentiated Services Code-Point (DSCP), end-to-end guarantees require the support of a certain PHB in all pertaining DS domains, thus making an end-to-end deployment with only a few PHBs more feasible.

The intention of specifying Per-Hop Behaviors for aggregates is always the establishment of services. Because the concept of DS is bound to domains in which the specific PHBs are applied, services are established by domains and are provided within domain boundaries. Recently, Nichols and Carpenter [53] formulated this particular property by the definition of a Per-Domain Behavior (PDB).

Definition 1 *The term Per-Domain Behavior (PDB) describes the expected treatment that an identifiable or target group of packets will receive from "edge-to-edge" of a DS domain. A particular Per-Hop Behavior (PHB) (or, if applicable, list of PHBs) and traffic conditioning requirements are associated with each PDB.*

In the following, the two standardized PHBs are described in more detail:

2.2.1 Expedited Forwarding

The EF PHB [21] is intended for building a service that offers low loss, low delay, and low delay jitter, namely a Premium Service. In addition, the EF PHB can be used to implement a Guaranteed Rate Service. However, if flows with different delay requirements and especially, if bursty flows share the EF PHB, traffic shaping of these bursts at the edge of the DS domain is likely to be required. To avoid a starvation of other traffic classes, EF traffic has to be strictly policed at the ingress router of a DS domain and excess EF traffic has to be discarded [21]. The specification of the EF PHB was recently redefined to allow for a more exact and quantifiable definition [15].

2.2.2 Assured Forwarding

Besides the Premium Service a so called Olympic Service [38] is proposed by the IETF to be based on the AF PHB group by extending it by means of a class based over-provisioning. Three of the four currently defined classes of the AF PHB group are used for such an Olympic Service [63]. The service differentiation between the three classes Gold, Silver, and Bronze is proposed to be performed by the means of admission control, assigning only a light load to the Gold class, a medium load to the Silver class, and a high load to the Bronze class. Drop rates can be configured independently within the three Olympic classes. Each AF class consists of a differentiation into up to three levels of drop precedence, namely Green, Yellow, and Red. The different levels of drop precedence can be supported in the network by Multiple Random Early Detection (MRED), which allows applying multiple instances of RED [29] with independent parameters to implement different levels of drop precedence. Like the EF PHB, the AF PHB can be applied to build a Guaranteed Rate Service for conforming traffic that is marked Green. An advantage of an AF based Guaranteed Rate Service is the isolation of the pertaining traffic from the EF based Premium Service. Doing so significantly reduces the interdependencies of the two aggregate classes, however, this distinction is not necessarily required [64].

2.2.3 Classifying, Policing and Packet Marking

A fundamental requirement to implement a Per-Hop Behavior is to support specific well known features in each network device. Edge routers must classify, police, mark, and optionally shape the traffic. Core routers have to provide specific congestion avoidance and management mechanisms to establish the desired per-hop behavior. The term "policing" denotes the ability to check whether a specific set of timely ordered incoming packets is not exceeding a given traffic profile. A common technique for specifying a traffic profile uses the "Token Bucket Model". A token bucket is a non-negative counter which accumulates tokens at a constant rate r until the counter reaches a maximum capacity b, the token bucket depth. Upon packet arrival, the packet size in Bytes is checked against the amount of tokens in the bucket. If this amount exceeds the packet size, the packet is treated as conforming and the actual amount of tokens is reduced by the size of the packet. Whenever the increase of tokens was stopped due to a full token bucket, the filling is initiated again. If there is not a sufficient amount of tokens in the bucket the packet is treated as exceeding its traffic profile. To clarify the use of a token bucket (r, b), consider that it is used for policing a flow in such a way that exceeding packets were dropped. Using this token bucket, one could assure that for any given time interval $[t_0, t_1]$ of length $T = t_1 - t_0$ the amount of data passing this policing function is not exceeding $rT + b$ Bytes.

An implementation of the policing function requires to actually classify incoming packets based on their characteristics, such as source and destination IP-address, Port numbers, Type of Service field of the IP-header, or DSCP. Once the decision whether a packet is relevant or not has been made, it is policed against the related traffic profile. Depending on the question whether the packet was conforming or not conforming, it receives a different treatment.

A common packet treatment used in this context is to assign a service specific DSCP value to the related IP packet. The action for packets exceeding the reserved rate could be to either transmit them with a different marking, or to drop them explicitly.

Packet classification facilitates the transition from flows to aggregates. Policing is used for controlling access to aggregates, i.e. for admission control, and marking is the entry point to aggregates.

2.2.4 Congestion Management

While edge routers police and mark the incoming traffic, core routers implement the related aggregate behavior. Two major concepts exist to fulfill this tasks.

Congestion Management is the ability of a network device to perform a specific packet schedule when the output link is congested. The ideal approach is to associate a relative weight (or precedence) with each individual traffic flow, and at every router, segment each traffic flow into an individual First-In First-Out (FIFO) queue, and configure the scheduler to serve all queues in a bit-wise weighted round robin fashion. This is an instance of a Generalized Processor Sharing (GPS) [57] discipline.

A popular approximation to GPS is Weighted Fair Queuing (WFQ). WFQ offers dynamic, fair queuing that divides bandwidth across a set of queues of traffic based on weights. Given the weight of the queues, WFQ calculates the time the packet finishes service under the GPS scheme and ensures that packets are served in the order of the time they would finish their service in a GPS environment.

Weighted queues can be established based on a value of the DSCP. That is, class-based WFQ is able to detect higher priority packets marked with a related DSCP and can schedule them in a well-defined manner. Class-based WFQ is conceptually very well suited for scheduling traffic with a marked precedence.

In periods of congestion each WFQ class is allocated a percentage of the output bandwidth equal to the weight of the related queue. For example, if a DSCP class is assigned a weight of 30, packets from this class will allocate at least 30 percent of the outgoing bandwidth during periods of congestion. It is important to note that WFQ only has an effect when there is congestion. When the interface is not congested, the treatment of packets is independent from their classification.

Since DS offers services that are of higher quality than the BE Service, an admission control is required, which controls incoming traffic and further on allows for billing. Such an admission control can be implemented by an automated DS network management based on a centralized entity a so-called Bandwidth Broker.

2.3 The Concept of a Bandwidth Broker

The task of a Bandwidth Broker is to facilitate and control the access to network services for applications. Here, the Bandwidth Broker performs admission control, resource provisioning and other policy decisions. From the Differentiated Services (DS) architecture point of view, a Bandwidth Broker expands the Per-Hop Behavior (PHB) of aggregates to a Per-Domain Behavior (PDB) [53].

Definition 2 *A Bandwidth Broker is a middleware service which controls and facilitates the dynamic access to network services of a particular administrative domain. Bandwidth Brokers are also viewed as the Policy Decision Point (PDP) of the controlled domain.*

Each PDB has measurable, quantifiable attributes that can be used to describe the offered service characteristics. While the associated PHBs do have a great impact on the achievable PDB attributes, the topology and the entering traffic profile influence the service as well.

It is very unlikely that a single Bandwidth Broker will control more than one administrative domain. Instead, each administrative domain wishes to have control over its resources and will operate its own policy decision point. A network reservation for traffic traversing multiple domains must therefore obtain multiple network reservations. Whenever service guarantees have to be obtained in multiple domains, a specific contract between peered domains comes into place. This contract is called a Service Level Agreement (SLA). It regulates details about available service levels by specifying the related parameters, access policies, and associated costs. Hence, the SLA regulates the acceptance and the constraints of a given traffic profile. Note that service associated costs and access rules are also listed in this contract. In contrast to the formal character of an SLA, Service Level Specifications (SLS) are used to technically describe the appropriate QoS parameters [71] that an SLA demands. According to [53] the definition of an SLS is as follows:

Definition 3 *A Service Level Specification (SLS) is a set of parameters and their values which together define the service offered to a traffic stream by a DS domain. It is expected to include specific values or bounds for Per-Domain Behavior parameters*

SLSs are used by Bandwidth Brokers as input information for their admission control procedures. End-to-end guarantees can then be built by a chain of SLSs, i.e. by the transitivity of all SLSs.

An example SLS could list domain related boundaries for delay and jitter for traffic of this type of service. It could also specify a minimum bandwidth available for requesters, depending on either their source and/or destination address, and a threshold for the available bandwidth. In this context, the service requester would be able to claim the demand of bandwidth dynamically. Whenever SLSs allow a service provision independently from the actual egress point, the granularity of control is quite limited:

- The maximum available bandwidth for this type of service does only depend on the minimum link capacity and not on the actual path in the network
- Multiple ingress domains might interfere and thus might reduce the amount of available bandwidth again
- Delay and jitter boundaries used in the SLA have to be calculated based on worst-case assumptions for their use of network links

Because of these limitations, we propose an architecture where an SLS lists the service parameters between each pair of ingress and egress point. In this environment, a Bandwidth Broker dynamically maps service requests to their traversing path. A service request is only granted if the Bandwidth Broker is able to validate the service availability on all traversed links. Each service request must therefore contain a destination address which allows the derivation of the related destination domain. Note that this model also applies to environments in which a network operation center manually performs the tasks of a Bandwidth Broker.

The desired granularity of service control is different for transit domains than for end-domains. While end-domains typically perform their admission control on a per-request basis, transit domains are focusing on their contractual conformance. This leads to the abstraction of a core tunnel:

Definition 4 *A core tunnel is an aggregated unidirectional reservation between two end-domains. It connects the egress router of the source-domain with the ingress router of the destination-domain applying the parameters of the service request. Any subject authorized to use this tunnel may then request portions of this aggregate bandwidth by contacting just the Bandwidth Brokers of relevant end-domains. Intermediate domains do not need to be contacted as long as the total bandwidth remains less than the size of the tunnel.*

A core tunnel instantiation of a single transit domain nicely relates to the concept of Per-Domain Behavior, and fits well to the proposal to incorporate traffic engineering mechanisms during its allocation. In some sense, the PDB can be viewed as a core tunnel with particular service parameters. While the term core tunnel is used to model aggregated reservations, the abstraction of a traffic trunk [6] is frequently used to model the realization of a core tunnel. Core tunnels traversing multiple domains can then be seen as the concatenation of the PDBs of all transit domains.

The abstraction of a core tunnel allows an entity to request an aggregate end-to-end reservation. Users authorized to use a particular core tunnel can then request portions of this aggregate bandwidth by contacting just the two end domains. The intermediate domains do not need to be contacted as long as the total bandwidth remains less than the size of the tunnel.

Whenever traffic engineering mechanisms are incorporated into the Bandwidth Broker of a transit domain, the question arises how these mechanisms are instantiated at the edge router. In Multi-Protocol Label Switching (MPLS) [59] networks, the allocation of a core tunnel respective a traffic trunk can be associated with the placement of a Label Switched Path (LSP). However, traffic designated to use the core tunnel must somehow be routed to the related LSP. Solutions for this problem will be presented in the following sections.

Chapter 3

Testbed and Experimental Configuration

The PAB-project evaluated the concepts of service provisioning in an MPLS-domain by an experimental analysis that was performed with widely deployed router hardware. Here, the project was able to benefit from a collaboration with Cisco Systems through the University Research Program and was therefore enabled to explore the proposed technologies in a hardware environment that is closely related to that deployed in the current G-WiN. Further on, the Command-Line Interface (CLI) of the operating system is similar to that interface offered by other hardware vendors, particularly by emerging popular Layer2/3-Switch products.

3.1 Testbed

Our experimental configuration comprises a laboratory testbed at the Research Centre Jülich, donated by Cisco Systems. The testbed allows controlled experimentation with basic DS mechanisms. Four Cisco Systems 7200 series routers were used for all experiments. These are either connected by OC3 ATM connections, by Packet Over SONET (POS), by Fast Ethernet, or by Gigabit Ethernet connections. End-system computers are connected to routers by switched Fast Ethernet connections. Hence, the minimum MTU size of our testbed is that of the end-systems: 1500 Bytes. Figure 3.1 illustrates the configuration used during the basic service evaluation. Note that the actual configuration during experiments varied.

Various Cisco Internetwork Operating System Software releases have been used during the lifetime of the project. The focus was on releases that did offer the newest functionality, i.e. early deployment versions (IOS E releases) and so-called T-releases. All of the described functions were available with the special early deployment Version 12.2(14)S5 and later versions.

3.2 Evaluation Tools

We performed several experiments demonstrating the performance of high-end TCP applications [65] like a guaranteed rate file transfer with a fixed deadline and typical UDP applications like video streaming [28] or videoconferencing. The following major tools have been used and enhanced within the project:

- *gen_send/gen_recv – BE UDP traffic generator.* This traffic generator was applied to generate BE UDP traffic with a mean rate of 50 Mbps and different burst characteristics. These UDP flows do not aim to model any specific applications, but we assume that the applied burst characteristics reflect effects that occur in today's and in the future Internet. TCP streams are initially bursty, UDP based real-time applications are emerging, which create bursts, for

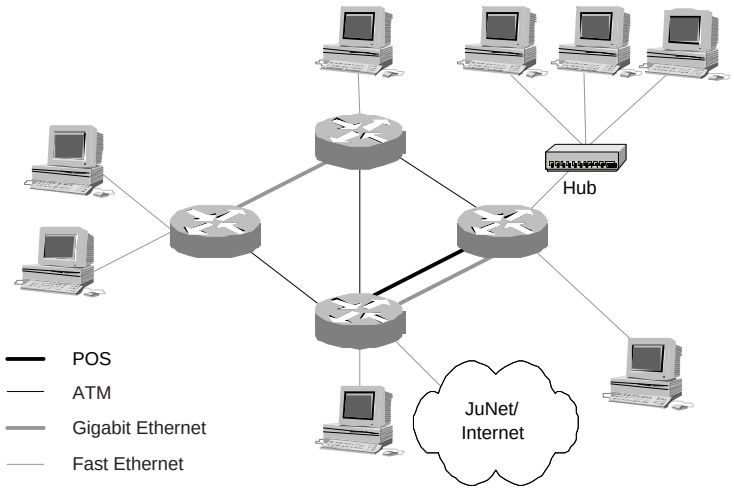

Figure 3.1: Basic Testbed Configuration

example by intra-coded frames in a video sequence. Further on burst sizes increase in the network, due to aggregation and multiplexing [14].

- *rude/crude – Delay-sensitive UDP traffic generator.* This traffic generator allows to measure the one-way delay and delay jitter. In several experiments we used real-time traffic patterns from script files, which we created from publicly available video traces [28]. We applied IP fragmentation for the transmission of frames that exceed the MTU, which we consider as being allowed here, since we configured the DS classes to prevent from dropping fragments. Note that IP fragmentation is a potential problem for a DS domain since packet classification is working differently for fragments. Hence, it typically requires host-based packet marking. The sequence, which we applied for the experimental results shown in this paper, is a television news sequence produced by the ARD. The sequence is MPEG-4 encoded with a minimum frame size of 123 Byte, a maximum frame size of 17.055 KB, a mean rate of 0.722 Mbps and a peak rate of 3.411 Mbps. The Hurst parameter is about 0.5 and decays with an increasing aggregation level. Figure 3.2 illustrates the traffic profile of the sequence.

- *ttcp – TCP stream generator.* We used the widely known TCP benchmark ttcp to generate TCP load. In the experiments reported in this paper we selected an end-system which was not capable of generating a rate of more than 1.8 MB/s = 14.4 Mb/s and if not stated otherwise we applied a socket buffer corresponding to a maximum window size of about 15 MTU.

- *Videoconferencing Software.* We used the RWTH-Aachen development VEXPHONE and RAXLET. Additionally, we applied the MBONE-tools VIC/RAT.

- *Bandwidth Broker.* We used the General-purpose Architecture for Reservation and Allocation (GARA). GARA was developed as a next generation resource management prototype of the Globus project and is used within the EU-projects DataGrid and DataTAG. GARA is a modular and extensible QoS system architecture that integrates different QoS mechanisms. There exist prototype implementations for several resource types, including a Bandwidth Broker for the DS architecture. A comprehensive description of GARA can be found in [61]. Further details about the use of GARA will be listed in Chapter 7. Note that the results of the EU-funded Grid Interoperability Grid Project (GRIP) led by FZ-Jülich can be used as a basis for the incorporation of Globus services into a UNICORE-based Grid. This conveniently gives a roadmap for application initiated network services in both frameworks: UNICORE and Globus.

Clock synchronization was assured by the use of the Network Time Protocol (NTP).

3.3 Experimental Analysis of Service Differentiation

Applying the plain BE Service to the two example applications used throughout this paper we generate the baseline for our evaluation. Our configuration allocates the remaining capacity of the ATM bottleneck link, which is not used by any other class, to the BE class. In the following experiments no other class than BE is used, resulting in an assignment of 60 Mbps of the bottleneck ATM link to the BE class. The tx-ring-limit parameter on the ATM interface card that specifies the queue size, which is assigned to the applied ATM PVC, was set to 16 particles each of 512 B allowing to store up to four MTU on the ATM interface. This value is by far smaller than the default value, but it has to be applied to allow for an efficient QoS implementation [24]. The BE layer 3 queue was configured to hold at most 256 packets. We consider this queue size, which is a trade off between delay and loss rate, as being feasible for BE traffic, which is more sensitive to packet drops than to queuing delay in a range of a few tens of milliseconds.

In Figure 3.3 the delay measured when transmitting the news sequence in the BE class is shown. Congestion is generated by applying an UDP stream with two bursts, each of ten seconds duration. As can be seen from figure 3.3, the delay is bounded to about 42 ms, showing some minor effects on the measurements due to tail-drop in the router. The delay corresponds to an effective data rate on the ATM interface of about 48 Mbps after subtracting the ATM induced overhead. While this delay is acceptable for streaming video applications, it can be critical for real-time video applications like videoconferencing. A Weighted Fair Queuing (WFQ) environment

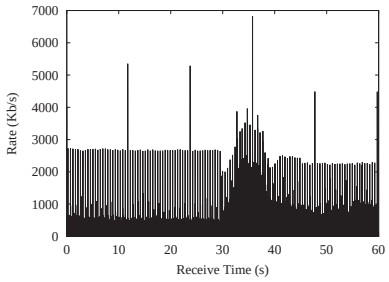

Figure 3.2: Data Rate UDP News Sequence.

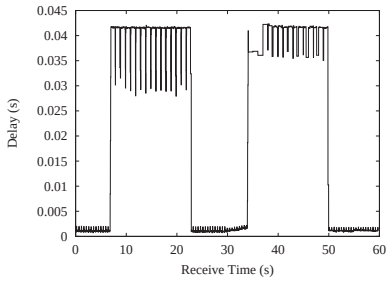

Figure 3.3: Delay BE UDP News Sequence.

is used for the implementation of the Olympic Service based on three AF PHB classes. The Olympic Service [38] proposed by the IETF is realized by admission control and a class based over-provisioning. We carried out experiments with the transmission of the news sequence in each of the Olympic classes, with the classes configured according to table 3.1. Within each of the

Table 3.1: Core Configuration of the Olympic Classes

Class	Percent	Gross Capacity	Approx Net Capacity	Over-Provisioning Factor
Bronze	5 %	3 Mb/s	2.4 Mb/s	≥ 1
Silver	10 %	6 Mb/s	4.8 Mb/s	≥ 2
Gold	15 %	9 Mb/s	7.2 Mb/s	≥ 3

Olympic classes a differentiation of the drop probability for differently marked excess traffic can be performed by applying Multiple Random Early Detection (MRED). Nevertheless, we consider excess traffic in an over-provisioned class as harmful for the BE class. Therefore we mark the conforming traffic Green and we drop excess traffic in the over-provisioned classes, whereas we allow a marking of excess traffic as Red in the Bronze class. The layer 3 queue size of each of the three Olympic classes was configured to 128 packets in the WFQ environment. Consequently, the ingress meter and marker is based on a token bucket with a Confirmed Information Rate

(CIR) of 2.4 Mbit/s for all Olympic classes thereby leading to the over-provisioning factors given in table 3.1. A Confirmed Burst Size (CBS) of 32 MTU is used at the ingress. This value is intentionally smaller than the queue size applied, to avoid packet drops in the Olympic classes within the network and also to avoid a high utilization of the queuing space and thus to reduce queuing delays. Besides it has to be noted that the WFQ queue sizes are configured in packets, which can be smaller than the MTU, whereas the confirmed burst size is configured in Bytes.

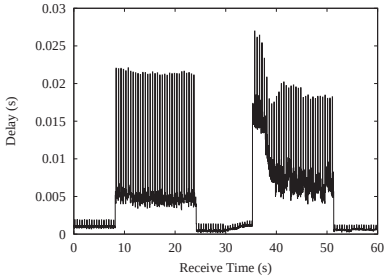

Figure 3.4: Delay Bronze UDP News Sequence.

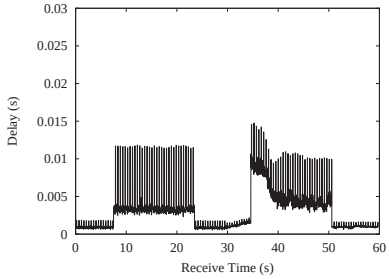

Figure 3.5: Delay Silver UDP News Sequence.

Figure 3.4 shows the measured delay for the news sequence in the Bronze class and the impacts of congestion in the BE class on the Bronze class. Compared to the transmission of the sequence within the BE class, which is shown in Figure 3.3, the delay is reduced significantly. Furthermore, packet drops did not occur in the Bronze class. Thereby AF based services can be applied as Guaranteed Rate Service without packet loss for conforming traffic.

The delay and delay jitter differentiation, which can be achieved in addition by the Olympic Service, is shown in figure 3.5 and 3.6 for the Silver and the Gold class respectively, compared to the Bronze class in figure 3.4.

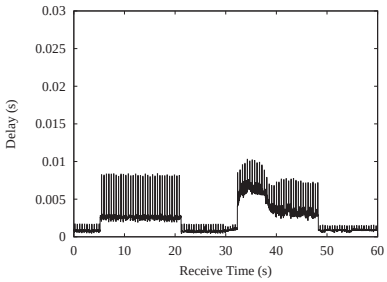

Figure 3.6: Delay Gold UDP News Sequence.

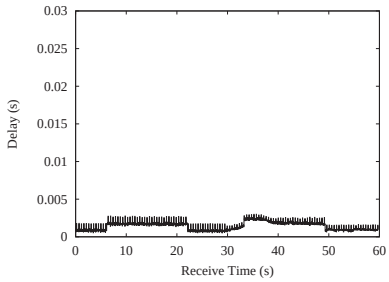

Figure 3.7: Delay Premium UDP News Sequence.

3.4 Implementation of a Guaranteed Rate Service and a Scavenger Service

In the following we report on an implementation of a Guaranteed Rate Service based on the different DS PHBs, the BE Service and possible Scavenger Services using commodity products. The traffic sources and sinks that were applied for the following evaluation of the different services are given in Figure 3.8.

Figure 3.9 shows the queuing and scheduling unit of a DS core node that supports the EF, and AF PHB, the BE Service, and the Scavenger Service. The EF PHB is based on Priority Queuing (PQ), AF, BE, and the Scavenger Service on Weighted Fair Queuing (WFQ). We concatenate two WFQ scheduler to form a hierarchical service policy, which separates the AF PHB from the group of BE Service and Scavenger Service. In case of queued packets in all of the classes, this hierarchical service policy could be replaced by a flat one, whereas, if any of the queues is empty, the hierarchical policy allows for a different redistribution of resources. The L2 queue that is

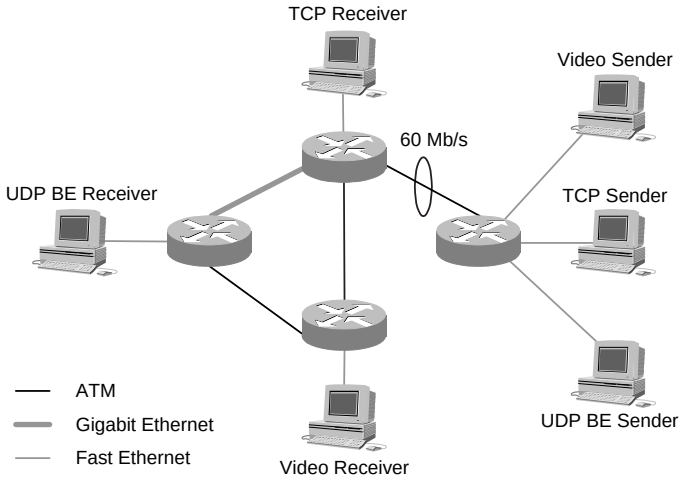

Figure 3.8: Testbed Configuration used for the Evaluation of DS-based Services

shown in Figure 3.9 is implementation dependent and usually required on the outgoing interface card of a router [63]. In the right of the figure the MRED configuration that is used within the AF classes is shown. An exponentially weighted moving average of the queue size s is used as an input parameter to derive a drop probability p for incoming packets. This drop probability is zero, as long as the average queue size does not exceed a minimum threshold $min_{g,r}$, and it is one, if it exceeds a maximum threshold $max_{g,r}$. Between the two thresholds p increases within the interval $[0, p_{g,r}]$. Within the AF classes independent RED curves apply to packets that are marked Green, or Red, such that Red marked packets are discarded earlier and with a higher probability. Yellow marked packets are not supported in accordance with the AF PHB specification [38].

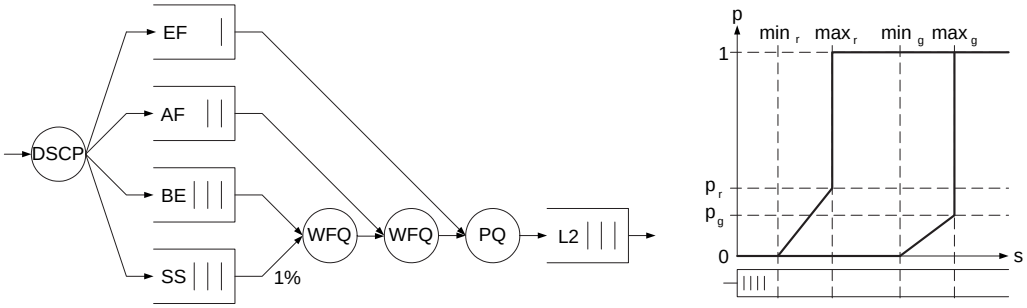

Figure 3.9: Differentiated Services Queuing and Scheduling and Assured Forwarding Multiple Random Early Detection

3.4.1 Guaranteed Rate Service

A Guaranteed Rate Service can be implemented in a DS network either by applying the EF PHB or the AF PHB. Since we implement both PHBs in our testbed, either of these two can be chosen.

In detail the EF PHB is implemented in the testbed based on PQ. A token bucket mechanism is attached to the priority queue in order to bound the amount of EF traffic to 10 % of the bottleneck link capacity and to allow a maximum EF burst size of 48 kB. The same parameters are used at the ingress side of the DS domain for metering and marking incoming EF traffic. Excess traffic is discarded as is required by the standard [21].

The AF PHB [38] is implemented based on Weighted Fair Queuing. In the following experiments only one AF class is used to form the basis of a Guaranteed Rate Service. Here the DS domains ingress configuration allows to match the reserved DS domains core capacity with Green marked packets that correspond to a low drop precedence. Excess traffic is marked Red and is mapped to a high drop precedence implemented by MRED. A pre-coloring of streams by upstream domains, or DS aware hosts is supported.

3.4.2 Low Loss Alternative Scavenger Service

The Scavenger Service proposed in [77] is designed for applications that are both loss and delay tolerant. However, we apply the fundamental idea of the Alternative Best-Effort (ABE) Service [40] and propose an Alternative Scavenger Service, which provides a low loss probability to loss-sensitive but responsive flows such as TCP streams. TCP implements two main sources of congestion control [3]. These are a self-clocking by the Round-Trip-Time (RTT) and a congestion window size adaptation in case of packet loss. Whereas an increase of the RTT is tolerable and can often be addressed by the use of a larger socket buffer size that allows for a larger congestion window, frame loss leads to a significant performance degradation. In detail the self-clocking of TCP can be described by equation 3.1 with R denoting the data rate and W denoting the window size in Bytes.

$$R = W/RTT \tag{3.1}$$

Equation (3.1) expresses that with the window flow control applied by TCP a maximum of W unacknowledged Bytes may be outstanding. The acknowledgement of these W Bytes is received after one RTT, thus allowing the sender to transmit another W Bytes, leading to a rate of W Bytes per RTT. Further on TCP implements an Additive Increase Multiplicative Decrease (AIMD) technique that increments the congestion window by one Maximum Segment Size whenever a complete window of data is acknowledged successfully, whereas the congestion window is at least halved in case of a packet loss.

To implement an efficient Scavenger Service routers require an amount of queued data in the Scavenger class ideally at any time, to be able to transmit this data, whenever there is no data of the BE class available. Thereby it becomes obvious that rather big TCP congestion windows are required, which are only offered by TCP, if packets are not lost. Thus, applying the Scavenger Service, TCP performance is heavily influenced by packet loss, which we address by a Low Loss Alternative Scavenger Service (LLASS). This service is implemented by Weighted Fair Queuing with a separate queue and a minimum weight of 1 %. In Figure 3.9 the queue that is applied for the LLASS is labelled with SS. Our configuration applies two WFQ scheduler that are combined hierarchically, such that unused BE capacity is at first assigned to the LLASS class. Only if the BE and the SS queue are empty, the unused BE capacity is redistributed to the AF PHB. We propose to assign a large buffer capacity to the LLASS class. In our implementation, a buffer capacity of 256 packets is used for the LLASS class compared to 128 packets for the BE class, in order to allow TCP streams to use comparably big windows and the window scale option without generating packet loss in case of congestion.

3.4.3 Low Delay Alternative Scavenger Service

In addition to loss sensitive but delay tolerant TCP streams, the Internet has to cope with an increasing number of real-time video and audio applications that are delay sensitive, but considered to be loss-tolerant to some extend. Thus, these flows require significantly different means of adaptation to network load and congestion. A Scavenger Service if applicable has to consider these properties.

Our implementation of such a Low Delay Alternative Scavenger Service (LDASS) is build on the MRED configuration of the AF PHB, but it could also be implemented by similar means in an ABE class. The relevant AF class offers a Guaranteed Rate Service for packets that are marked Green, whereas it in addition allows to oversubscribe the AF class with packets that are marked Red.

These Red packets are mapped to an aggressive RED curve, which in case of network congestion drops Red packets very early and rigourously, thus limiting the delay increase imposed by such Red packets on Green ones. The MRED configuration implemented in our testbed differentiates between the two drop precedence levels Green and Red. It uses the thresholds $min_r = 16$, $max_r = 32$, $min_g = max_g = 64$ and the maximum probability $p_r = 0.1$. Thereby it is ensured that Red packets are dropped comparably early and further more that Red packets are dropped with a probability of 1.0 before Green packets are discarded. On the other hand, if no Green marked packets are available, Red marked packets are able to scavenge the unused AF capacity, and, if neither BE nor LLASS traffic is queued, also the configured BE capacity. We disable the RED mechanism for Green packets to implement a lossless Guaranteed Rate Service by these means.

3.5 Multi-class Applications

In this section we report on two example applications that we implemented for a utilization of a Guaranteed Rate Service and a Scavenger Service in parallel. Instead of the Guaranteed Rate Service a Best-Effort Service or Alternative Best-Effort Service can be applied at the cost of not being able to give any performance guarantees as it can be done if a DS based Guaranteed Rate Service is used. The two example applications we present are significantly different in that the first one – a deadline file transfer based on a parallel FTP – is TCP based, whereas the second one – the transmission of hierarchical video streams – is RTP/UDP based. In the reported experiments we create congestion with a very aggressive and unresponsive BE UDP flow, to be able to clearly show the qualitative impacts of BE congestion on both the Guaranteed Rate Service and on the Scavenger Service.

3.5.1 Multi-class Grid FTP

Grid FTP [2] was proposed in the context of Grid computing to achieve an extremely high throughput. It is based on parallel streams and implemented by striped TCP sockets, as also applied by many Web browsers. Using several TCP streams in parallel circumvents TCP congestion control to some extend. Each of the individual TCP connections performs congestion control by adapting its data rate. Thus each TCP connection on its own can be considered to be TCP-friendly, whereas the sum of the rates achieved with parallel TCP connections is higher than the rate of a single connection. In order to increase fairness towards applications that use only a single TCP stream, Grid FTP should ideally apply a Low Loss Alternative Scavenger Service. On the other hand Grid computing often requires that data is delivered fulfilling certain deadlines. Thus a reservation of network capacities is required, which can be addressed with a Guaranteed Rate Service for which we apply the EF PHB. A drawback in using the EF PHB is that a deadline file transfer can send data at most with the guaranteed rate. Probably unused capacity cannot be used by excess traffic, since when applying the EF PBH, excess traffic has to be discarded at the DS domains ingress. Similar problems arise, if the required Guaranteed Rate Service is implemented based on the AF PHB. Here excess traffic is not discarded at the ingress of the domain, but marked as Red and probably discarded in the domains core and can thus also decrease TCP performance significantly.

To address these shortcomings, we developed a multi-class Grid FTP as an example for a high data rate TCP based application. The multi-class FTP is configured to apply a Guaranteed Rate Service and a Low Loss Alternative Scavenger Service in parallel. Its performance is shown in Figure 3.10 for an example file transfer of 280 MB with a deadline at 200 s after the file transfer start. The file transfer application initially requests 12 Mb/s of EF capacity from the DS domains Bandwidth Broker to ensure that the file transfer deadline is met by applying the EF PHB for a Guaranteed Rate Service. It starts sending data at the requested rate by applying a rate shaping to 12 Mb/s at application level. Thereby the sending rate conforms to the requested rate and packet drops due to policing at the ingress router of the DS domain are avoided. A similar functionality can be achieved by implementing holding queues at the ingress router of the DS domain that allow

to temporarily buffer excess traffic [64]. In parallel data is transmitted at the maximum possible rate applying the Low Loss Alternative Scavenger Service. After succeeding in transmitting an additional configurable amount of 25 MB of data by applying the Scavenger Service, the application recomputes the required rate to meet the deadline and updates its request for EF capacity. In Figure 3.10 this happens for a first time after about 8 s and leads to a new required rate of about 10 Mb/s. In the time interval from 10 s to 20 s, congestion is created in the Best-Effort class by an aggressive UDP flow. As can be seen from Figure 3.10, this BE congestion does not have any impact on the Guaranteed Rate Service, but it cuts off the transmission of data by the Scavenger Service. The same process repeats until the complete file of 280 MB is transmitted after about 90 s instead of the target deadline of 200 s, which would have been reached, if only the Guaranteed Rate Service would have been used. Thus besides the reduced transmission time, our multi-class

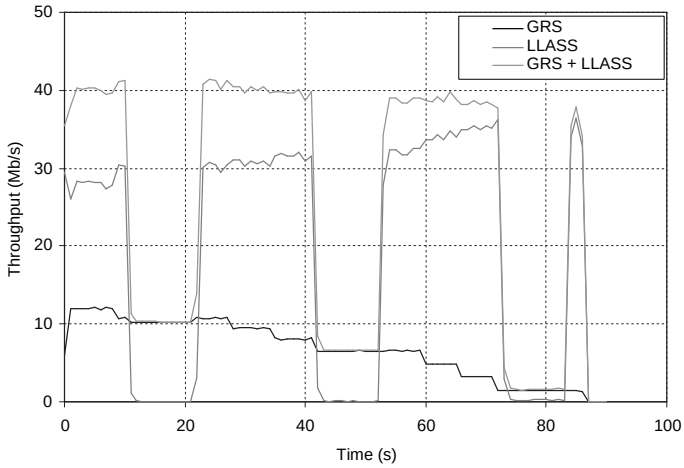

Figure 3.10: Multi-class Grid FTP combining Guaranteed Rate Service and Low Loss Alternative Scavenger Service

Grid FTP increases the utilization of the available network capacity without impacting the BE class and it reduces the required EF capacity to meet the deadline. We receive an economic model of a deadline file transfer which uses potentially cheap resources in the less than BE class to reduce the deadline assuring EF reservation. In addition to the cost-effectiveness for the user, this model also ensures that the scarce resource of a Guaranteed Rate Service is available to other requesters.

Scalability issues limit the approach of signalling required guaranteed rate updates to high bandwidth and long lived flows. However, since the core configuration of a DS domain is intended to be static, the signalling can as an option be applied at the source domain only. To address these concerns, the requirement of application level rate shaping can be replaced by updating the shaping configuration in the ingress router. Figure 3.11 illustrates the pacing property of an updated traffic shaping configuration. It is therefore possible to handle these updates within the Bandwidth Broker by a single policy propagation which includes an updated traffic shaping configuration.

3.5.2 Multi-class Hierarchical Video

An increasing number of RTP/UDP audio and video streams are transmitted across the Internet. Especially for high data rate video an application of a Low Delay Alternative Scavenger Service can be advantageous, too. As already mentioned real-time audio and video applications are delay sensitive, but to some extend loss tolerant. Concerning packet loss, video frames can be classified by their importance on the decoding process and on the quality of the decoded video stream. These differences in terms of importance can be efficiently accommodated by applying priority dropping schemes that protect more important video frames and drop less important ones with

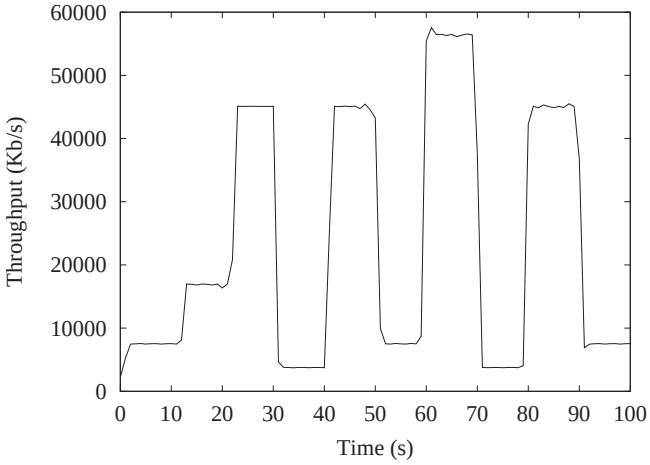

Figure 3.11: TCP flow paced by a varying traffic shaping configuration on the ingress router.

priority, if necessary [9]. One means of generating such a video stream with a number of layers with decreasing importance is the option of hierarchical video encoding, as it is offered by the H.263+ standard [17] or by the MPEG-4 standard. In detail the H.263+ standard gives three options of hierarchical video encoding:

- Temporal scalability is achieved deploying the disposable nature of bidirectionally predicted frames.
- Signal-to-Noise Ratio (SNR) scalability permits the refinement of encoded video frames and thereby modifies the SNR.
- Spatial scalability is achieved by encoding the frames with a low spatial resolution for the base layer and with a higher resolution for the enhancement layers.

An hierarchical encoding of video data leads due to packetization overhead to an increased data rate. However, as shown in [17] some options of layered encoding even allow for a higher compression ratio and thus for a reduced resulting data rate, compared to flat video encoding.

The realization of different drop probabilities that can be applied as priority dropping scheme for the transmission of the different layers can be based on a differentiated queuing and scheduling. It can be efficiently implemented by aggregate scheduling and the service classes offered by a DS network [25].

Figure 3.12 shows the performance of an example multi-class video transmission. The video application implements the option of SNR scalability of the H.263+ standard to generate a video stream consisting of two layers. A base layer, which contains the complete encoded video stream with a minimum requested quality, is transmitted applying a Guaranteed Rate Service. The Guaranteed Rate Service is implemented based on the AF PHB using Green marked packets. The enhancement layer, which in a joint decoding with the base layer increases the video quality, but which is not required for decoding of the base layer, is mapped to a Low Delay Alternative Scavenger Service. This service is also implemented by the same AF class, but the relevant packets are marked Red and are mapped to an aggressive RED curve. Further on the Low Delay Alternative Scavenger Service packets are allowed to exceed the configured AF capacity. The video application performs the relevant marking of packets to be either Green or Red, such that the ingress router of the DS domain already receives a pre-colored stream. Figure 3.12 shows the quality of the decoded video stream in terms of the Peak Signal-to-Noise Ratio (PSNR). In case of a transmission of the video base layer with the Guaranteed Rate Service, the PSNR oscillates in a range of 35-36 dB. If in addition the enhancement layer is transmitted using the Low Delay

Alternative Scavenger Service, an increase of the quality of the decoded stream to about 38 dB can be noticed. Further on the effects of BE congestion are shown in the interval from frame number 100 to 400. It can be seen, that BE congestion again has no impact on the Guaranteed Rate Service, which ensures the transmission of the video base layer. In contrast enhancement layer packets transmitted by means of the Low Delay Alternative Scavenger Service are dropped during congestion. This occurs due to the fact, that the rate at which Green packets are sent matches the configured capacity of the AF class. Thus, Red packets exceed this capacity and are dropped with priority by MRED, if both the AF class and the BE or LDASS class are loaded.

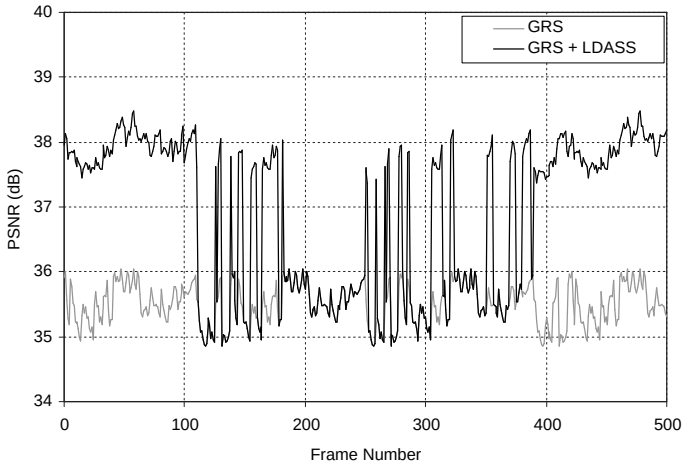

Figure 3.12: Multi-class Hierarchical Video Transmission combining Guaranteed Rate Service and Low Delay Alternative Scavenger Service

Generally, if either AF or BE capacity is unused, it can be collected by the Scavenger Services. Due to the hierarchical WFQ scheduler, unused BE capacity is at first assigned to the Low Loss Alternative Scavenger Service, whereas unused AF capacity is at first assigned to the Low Delay Alternative Scavenger Service. We assume that for BE and AF capacity billing is applied for example by charging a flat rate. Thus, Scavenger Services in both of the classes are able to scavenge unused capacity, which is already paid for, at a very low additional cost, but also at the risk of being heavily penalized in case of congestion.

Chapter 4

Multi-Protocol Label Switching

4.1 The Multi-Protocol Label Switching Architecture

In conventional packet switching, a given router forwards packets according to their IP destination addresses. For that purpose it maintains a forwarding table which specifies to which peered router and through which interface a packet with a given destination address should be forwarded. Any time a new packet arrives, a network part of its destination address is extracted. The router uses the longest match procedure to check in the forwarding table which destination is matched and what is the next hop for the packet.

The Multi-Protocol Label Switching (MPLS) architecture defines a new way of forwarding packets between routers[1]. It uses an additional header placed between the OSI layer 2 and 3 headers (or a part of an ATM header) containing a 20-bit field called an MPLS label. Note that MPLS is not bound to a particular link-layer or layer-3 protocol. Instead, it defines a particular encoding for major link-layer protocols including Ethernet indicating that the MPLS forwarding component should be used instead of the traditional IP-component. Then the next hop is determined not as usual by applying the packet's IP destination address to the longest match procedure but by the MPLS label. Contrary to IP addresses, MPLS labels have no global meaning since the router usually changes the label of a packet before it is forwarded. This leads to the demand for some signaling mechanism which establishes a consistent path of label pairs between ingress and egress routers.

When a packet enters an MPLS domain, its IP header is processed by the ingress router that decides about the insertion of the MPLS-header. Figure 4.1 illustrates this. Within the MPLS-domain, routers forward packets based on the incoming label and are not anymore required to support IP's longest match procedure. Taking this new functionality into account, the forwarding network elements within an MPLS-domain are called Label Switched Routers (LSRs). For readability, we will use the term router in the remaining text. Since labels do not have a global meaning, the outgoing label might be different than the incoming. Hence, the initial choice of the label by the MPLS ingress router defines the path towards the edge of the MPLS domain. This path is called Label Switched Path (LSP) and can be viewed as a tunnel which connects the ingress with the egress router. At the tail end of an LSP, the MPLS header is removed by the egress router. As a consequence, any packet leaving an MPLS-domain is forwarded further in its conventional way.

In addition to the 20-bit "Label" value, the MPLS header, as shown in Figure 4.1, contains 3 bits of the "Exp" field (usually used to request some QoS parameters), one "S" bit indicating whether further MPLS headers were following (label stacking is an important feature of MPLS, particularly in the context of recovery and VPN support), and the "TTL" (Time To Live) field used to limit the life time of the packet (used by loop detection mechanisms).

[1] MPLS does not replace the conventional forwarding but rather it is used as an additional switching technique, which is very useful in traffic engineering.

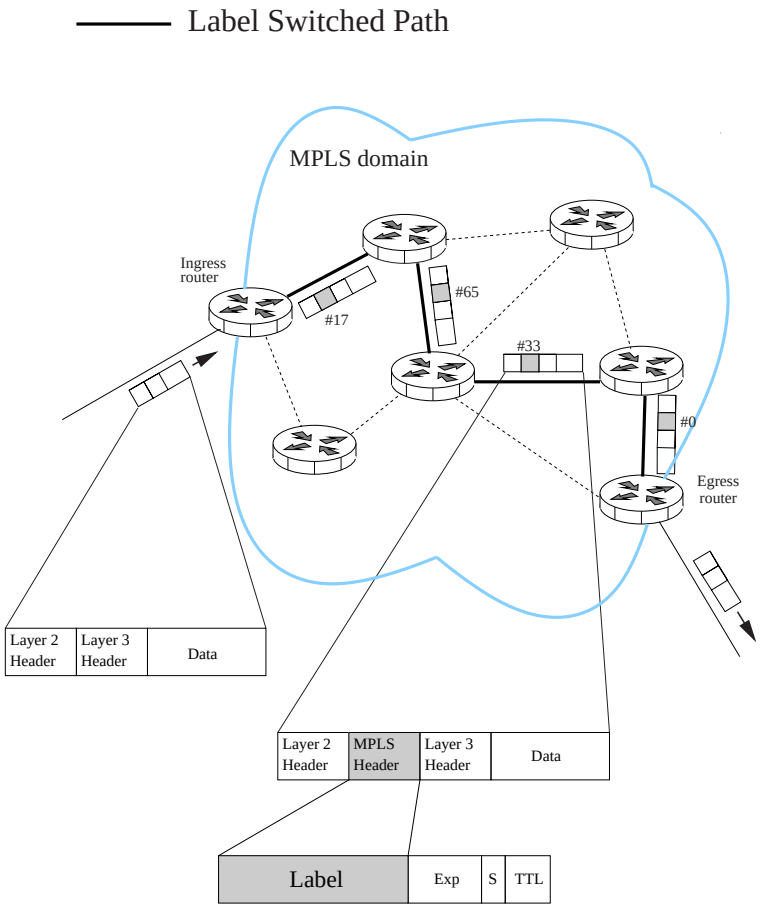

Figure 4.1: The example of packet forwarding through an MPLS domain. The ingress router classifies every incoming packet into one of the predefined Forwarding Equivalence Classes (FEC) and adds an MPLS header to it. The header contains the label value associated with the matched FEC. All the MPLS routers along the path traversed by the packet use their label switching forwarding tables to lookup for the next hop and the outgoing label to be used. The initial label is therefore updated along its path through the domain. In the illustrated scenario, the initial label "17" is changed to "65", then to "33" and finally to "0". The path determined by this procedure is called Label Switched Path. The label value of "0" is reserved for a particular purpose. When a router receives an MPLS packet marked with this label, like the egress router in the example does, it has to forward the packet by means of conventional IP forwarding instead of MPLS.

Label switching, similarly to network layer routing, consists of two functional components (see Figure 4.2):

- Control Component
  It specifies the way in which the routers distribute labels in order to establish an LSP. Finally, this component builds in each MPLS router a label switching forwarding table used by the Forwarding Component.

- Forwarding Component
  It forwards MPLS packets based on their labels and the information from the label switching forwarding table.

It is important to note that the main motivation for MPLS is not the improvement of the

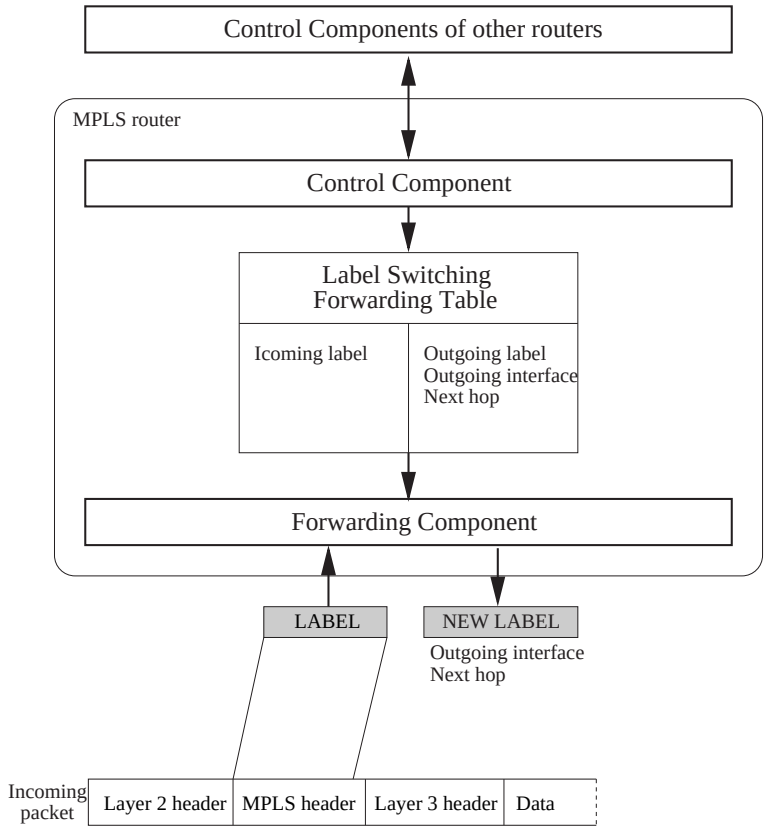

Figure 4.2: The figure shows the two functional components of an MPLS router. Control Components of the routers in the MPLS domain control the creation of LSPs by exchanging label information with other routers. The main task of the Control Component is to create the label switching forwarding table which is then used by the Forwarding Component to switch incoming packets to the proper outgoing interfaces. The arrows denote the flow of information.

forwarding throughput, nor the provision of new QoS mechanisms. Using MPLS encoded labels avoids to switch to the IP-layer processing routines of the router and, if implemented in hardware, can therefore improve the throughput significantly. However, the fundamental design goal of MPLS is the ability to influence the operation of the control component. Traffic engineering (TE) is one of the major driving factors for MPLS.

4.2 Label Distribution in MPLS

Routers using label distribution protocols exchange information by means of messages. Each message contains three fields: the number representing the type of the message, the length of the message and the data. This way of encoding, called Type-Length-Value (TLV), allows to easily extend the set of messages of a given protocol and thus to extend its functionality.

4.2.1 Classification of the Label Distribution

Upstream vs. downstream mode

Each label used in any LSP sector [2] is always referred as outgoing for the sending node and as incoming for the receiving node. It is therefore clear that only one from the two nodes should make a binding of the label to the associated FEC. Once the node has bound the label, it should advertise it to the second node. Hence, there are two possible modes concerning the label binding[3]:

- Upstream mode
  The node which is upstream with respect to the data flow makes the binding of the given FEC to the outgoing label and advertises it to the node which is downstream in respect to the data flow. In this way the downstream node learns his incoming label for a given FEC.

- Downstream mode
  The node which is downstream with respect to the data flow makes the binding of the given FEC to the incoming label and advertises it to the node which is upstream with respect to the data flow. In this way the upstream node learns his outgoing label for a given FEC.

The remaining text will assume that the downstream mode is chosen.

Downstream on-demand vs. downstream unsolicited

The label bindings can be requested by another node or sent unsolicited without any request. This implies the two possible modes for the label distribution:

- On-demand mode
  It is assumed that the incoming label binding is advertised to the upstream neighbor on request only.

- Unsolicited mode
  In this mode the incoming label binding is advertised to the neighbors unsolicited without waiting for any request from their side.

Figure 4.3 illustrates both procedures.

Conservative vs. liberal label retention

This classification refers to the unsolicited mode described in the previous section. Since any node working in this mode can send the incoming label advertisements to all peered routers, the node itself can receive the label binding for the same FEC from more than one node. In this case, the node must decide which binding is accepted and actually used. The output label bindings which were not chosen are treated differently in the two modes:

- Conservative label retention mode
  The rejected outgoing label bindings were not kept in memory for future use.

- Liberal label retention mode
  The rejected outgoing label bindings were kept in memory for future use. In case of fault conditions, such as a link failure, which invalidate the selected path, the actual binding will be replaced by one of the alternative bindings.

The example of the liberal label retention in a downstream label distribution mode is presented in Figure 4.4.

[2] An LSP sector refers to an LSP between two Label Switched Routers

[3] The term “downstream” is used to specify direction which is determined by the data flow while the term “upstream” is used to specify the direction which is opposite to the data flow.

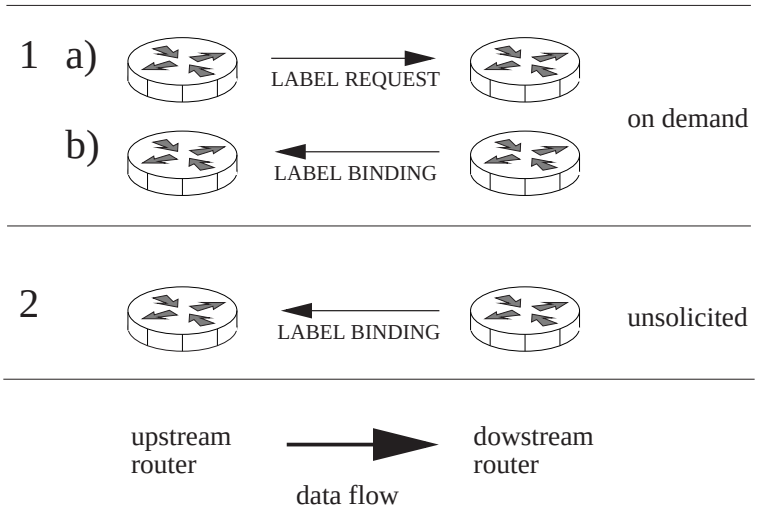

Figure 4.3: The difference between the downstream on demand (**1**) and the downstream unsolicited (**2**) label distribution modes. In the first case the label binding has to be requested (**a**) before it is advertised (**b**). In the second case no label request is necessary to advertise the binding.

Independent control vs. ordered control

Each node in the MPLS domain binds the labels to a given FEC independently from the binding performed by other nodes. However, the advertising process can be synchronized along the LSP. One of two following modes can be chosen here:

- Independent control mode
  In this mode, a node advertises the incoming label binding for a given FEC even when it has not yet learned the out-going label binding for this given FEC.

- Ordered control mode
  In this mode, a node advertises the incoming label binding for a given FEC only after it learns the out-going label binding for this given FEC.

The example illustrating downstream unsolicited independent and ordered control modes is shown in Figure 4.5.

The MPLS architecture does not specify how labels are actually distributed among the routers and how LSPs are then maintained. All modes for the label distribution presented in the previous section can be implemented.

The functionality of the label distribution protocol is also out of the scope of the proposed MPLS standard. In the simplest case, the MPLS forwarding table may be configured in such a way that it is equivalent to the conventional IP destination based one. However, the MPLS routing can be configured to fulfill any specific network engineering requirements.

4.2.2 Label Distribution Protocol (LDP)

The basic documents describing this protocol are RFC3036 (LDP Specification) and RFC 3037 (LDP Applicability). LDP was designed particularly for the purpose of distributing labels in an MPLS domain relying on the information gathered by some Interior Gateway Protocol (IGP), like OSPF or IS-IS. Note that LDP does not support explicit routing nor reservations. However, its extension described in the next section does.

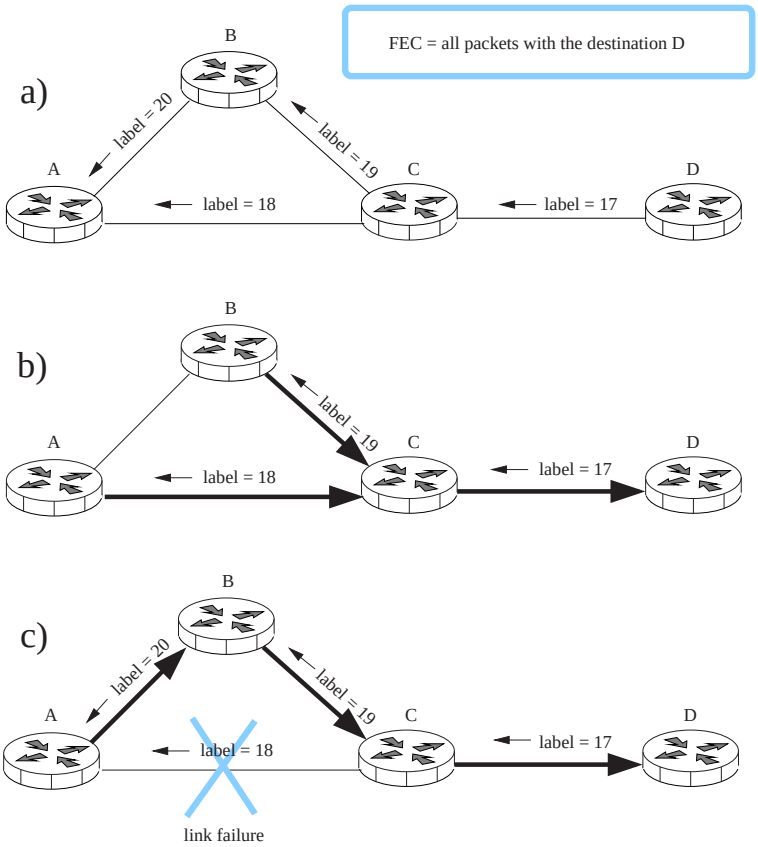

Figure 4.4: This Figure shows a simple example of the liberal label retention mode. **a)** The LSPs are going to be established for the FEC grouping all packets with the destination address of the router D. The label bindings for this FEC are distributed as shown on the picture. The router A receives two label bindings but it chooses only one label, the label "18". The label "20" is stored as an alternative label for future use. **b)** The LSPs are established as marked by the thick arrows. **c)** After the link failure the label "18" can not be used by the router A. The router restores the alternative label "20" and the new LSP to the destination router D is established.

General description

LDP works closely with the installed IP routing protocols. In a steady state, MPLS packets are routed exactly in the same way as in conventional destination based routing. However, some differences might appear for relatively short times when e.g. the network topology changes. Since the label switching is potentially faster than the conventional IP forwarding, an MPLS domain can benefit from the smaller delivery times of packets.

LDP is a set of specifications defining the way in which peer LDP nodes discover each other and exchange the necessary information. The communication, except during the initial stage, is based on TCP ensuring reliability of the message propagation. Since messages were specified in the TLV form, the protocol is easily extendible by adding new message types to the protocol specification.

The course of the LDP session can be divided into the following phases, illustrated in Figure 4.6:

- **Discovery**
  The aim of this stage is to find out which peered routers are LDP enabled routers. For that purpose the router periodically transmits a UDP broadcast message (**Hello** message) to all

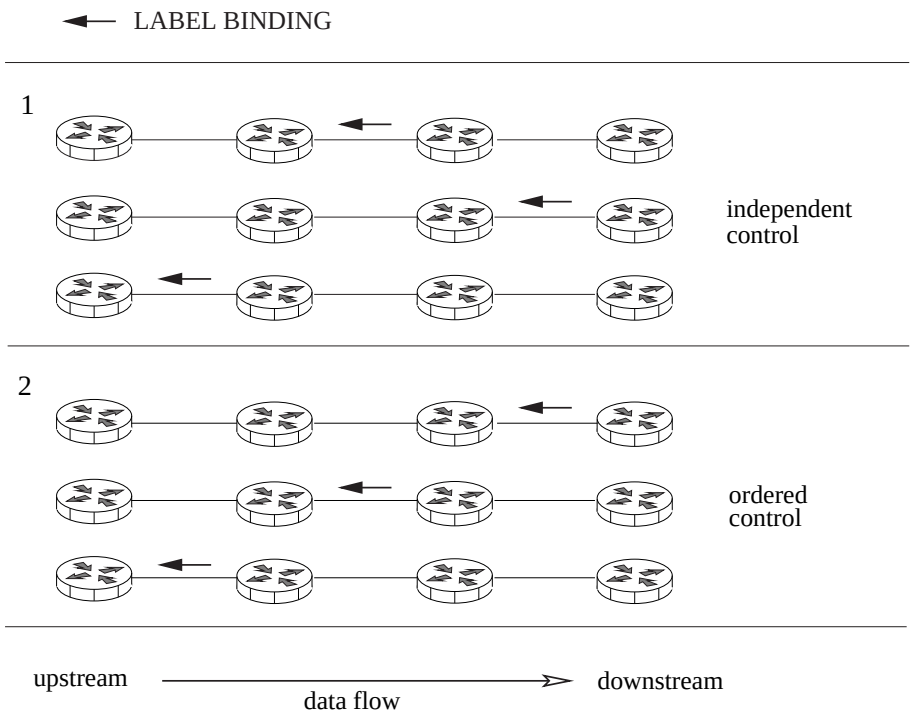

Figure 4.5: Illustration of the two label binding advertisement modes. **(1)** In the downstream unsolicited with independent control mode, the label bindings were advertised independently, without any synchronization. **(2)** In the downstream unsolicited with ordered control mode, any upstream node is not allowed to advertise its incoming label binding before it gets the outgoing label binding for the given FEC from the peered downstream node.

peered routers[4]. On the other hand, it receives the **Hello** messages from its neighbors.

- **Adjacency**
  The task of this step is to initiate the TCP session between two peered routers. After the discovery phase, a router knows each of its LDP neighbors. It uses this information to initialize a long-term TCP session with each of its neighbors. This is the purpose of the **Initialize** message. Note that the source of the **Initialize** message is typically selected based on the IP-addresses. If the parameters included in this message (e.g. LDP version, the method of label distribution, timer values, ...) are accepted by the neighbor the latter sends in response a **Keepalive** message (if not then the **Error** message is generated). At the end of the **Adjacency** phase, peer routers send one to another **Address** messages to advertise their interface addresses.

- **Label advertisement**
  In this phase, the bidirectional TCP session between peered routers is established. Its purpose is to facilitate the exchange of label bindings. The **Label_Mapping** message containing the bindings of the labels to the FECs is sent either unsolicited or on-demand (**Label_Request**). Additional messages were introduced to support the management of labels. A **Label_Request** message can be cancelled when it is followed by the **Label_Request_Abort** message. On the other hand, the router which advertises the label bindings can withdraw the advertised binding by sending the **Label_Withdraw**. A router

[4]The specification of LDP allows to establish an LDP session between routers which have no direct connection. In this case the address of the peer host should be known and the **Hello** is sent as uni-cast UDP message.

which is not going to use a received binding uses the **Label_Release** message to inform the peered node about its decision.

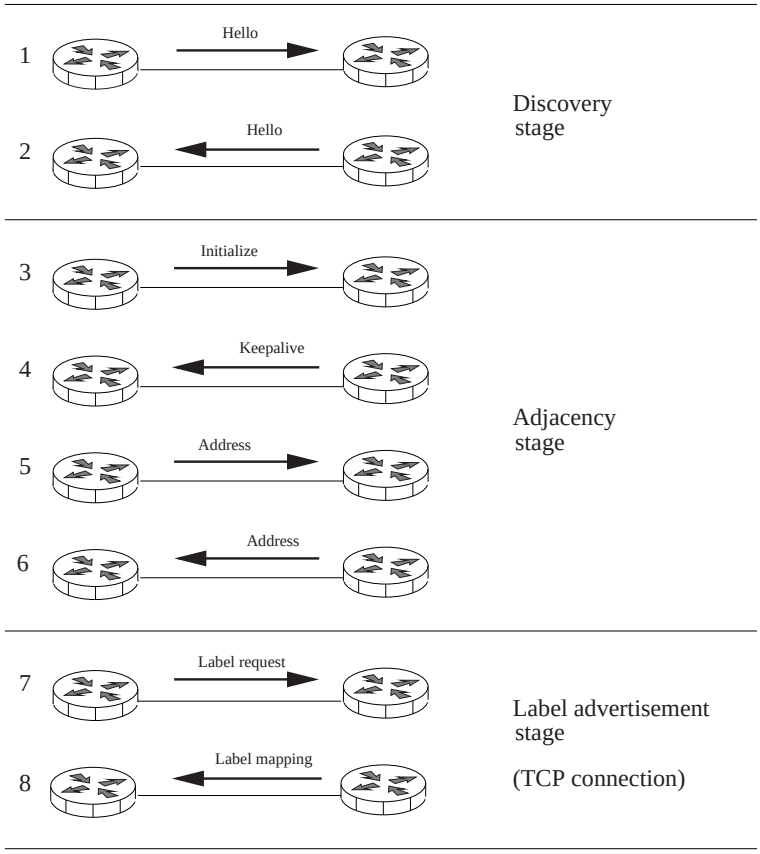

Figure 4.6: The sequence of the LDP messages exchanged by the two routers in the on-demand mode label distribution mode.

The way in which labels are being advertised to neighbors depends on the selected label distribution mode. However, downstream distribution is used in all cases. In general, the label bindings can be requested by an upstream router or sent unsolicited without any request. On the other hand, a router can either decide when to send the binding for a given FEC (requested or not) independent from the decision of other nodes or it waits until it receives the binding for a particular FEC from its downstream router. Combining these different possibilities, we receive four modes in which an LDP domain can operate[5]:

- Downstream unsolicited with independent control
- Downstream unsolicited with ordered control
- Downstream on-demand with independent control
- Downstream on-demand with ordered control

Note that the downstream unsolicited methods will initially generate more signaling traffic in a domain than the alternative modes. It is so, because a router probably receives the binding for a given FEC from all its neighbors, though it will select only one of them. Thus, the router receives

[5]The description of these modes can be found in Section 4.2.1.

more information than really needed. Depending on the retention mode, the router can ignore these extra label bindings by sending a **Label_Release** message (conservative label retention) or it can store them for future use. Whenever the routing table is updated in a way that a new node becomes the next hop of a particular FEC, the router can easily restore the label binding associated with the new next hop without any additionally LDP signaling (liberal retention mode). Clearly, the downstream on-demand mode does not allow this mode. Here, only the next hop for a given FEC is requested to advertise the binding. Hence, there are no extra binding information available.

In the independent control mode, the time needed for establishing an LSP is shorter than in ordered control where an upstream node has to wait for the binding from a downstream node before it is allowed to advertise its own. On the other hand, the ordered control mode offers more reliability since the label binding received for a given FEC indicates that the LSP is already established from this node towards the egress router for this FEC.

Once the labels were advertised and the network topology remains, the labels bindings do need to be refreshed. Only **Keepalive** messages were periodically sent to monitor the established LDP connections. Any time a router receives a **Keepalive** message, it resets a counter associated with the LDP session. Whenever the timer reaches its time-out value (hold-time), LDP assumes a fault condition and cares about the rearrangement of LSPs to omit the fault link or the node.

Implementation of LDP in Cisco routers

The tests of LDP were performed using the topology that is shown in Figure 4.7. All routers were configured to use the Open Shortest Path First (OSPF) routing protocol.

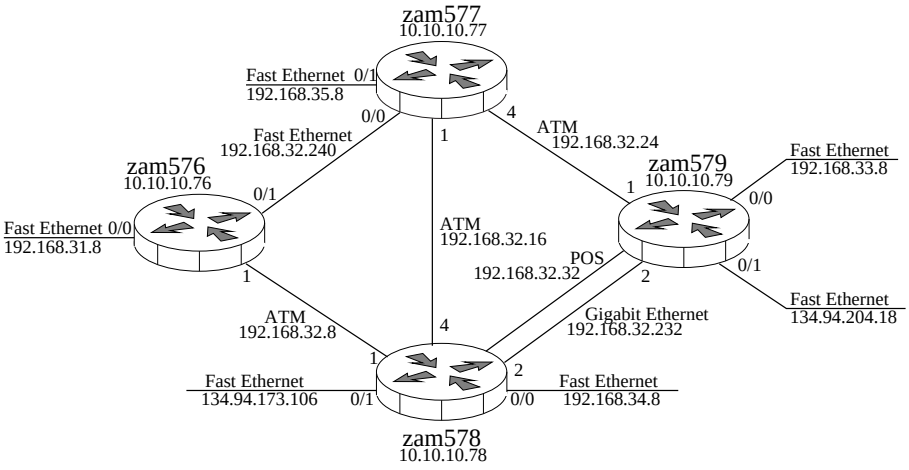

Figure 4.7: Configuration of the testbed used to evaluate the operation of LDP in an MPLS domain. Four Cisco 7200 routers were interconnected with Fast Ethernet, Gigabit Ethernet, Packet Over SONET and ATM links. The routers's names *zam576* ... *zam579* are given together with their loopback IP addresses, i.e. a unique IP address of a router. For each link its name, the IP address and the port numbers are shown.

All routers were using a common operating system version as indicated by the output of the `show version` command in the EXEC mode:

```
zam57X#show version
Cisco Internetwork Operating System Software
IOS (tm) 7200 Software (C7200-JS-M), Version 12.2(4)T,  RELEASE SOFTWARE (fc1)
(...)
```

Configuration of an MPLS domain

The first step to activate MPLS in the Cisco-based testbed is to enable Cisco Express Forwarding by the following command executed in the general configuration mode:

```
zam576(config)#ip cef
```

Next, in order to enable MPLS routing on all interfaces interconnecting the routers, the two-step configuration described below should be done.

- In the general configuration mode and in the interface configuration mode[6]:

  ```
  zam57X(config)#mpls ip
  zam57X(config-if)#mpls ip
  ```

- In the general configuration mode and in the interface configuration mode:

  ```
  zam57X(config)#mpls label protocol ldp
  zam57X(config-if)#mpls label protocol ldp
  ```

  In this way LDP is chosen to distribute MPLS labels. Alternatively one could select the older TDP (Cisco's Tag Distribution Protocol). Both protocols are functionally identical but they differ in the message formats and the related IOS procedures.

The flow of LDP messages.

By executing the configuration commands described above, LDP sessions were initiated between peered routers. After enabling the LDP message logging as below (in the privileged EXEC mode):

```
zam57X#debug mpls ldp messages sent all
zam57X#debug mpls ldp messages received all
```

one can use the following command to monitor the messages recorded by the router by executing the command:

```
zam57X#show logging
```

An example illustrating the LDP message flow between *zam576* and *zam577* (from the point of view of the first one) is given below:

```
13:16:15.768: ldp: Rcvd init msg from 10.10.10.77 (pp 0x0)
13:16:15.768: ldp: Sent init msg to 10.10.10.77:0 (pp 0x0)
13:16:15.768: ldp: Sent keepalive msg to 10.10.10.77:0 (pp 0x0)
13:16:15.792: ldp: Rcvd keepalive msg from 10.10.10.77:0 (pp 0x0)
13:16:15.792: ldp: Sent address msg to 10.10.10.77:0 (pp 0x6306D01C)
13:16:15.792: ldp: Sent label mapping msg to 10.10.10.77:0 (pp 0x6306D01C)
13:16:15.792: ldp: Sent label mapping msg to 10.10.10.77:0 (pp 0x6306D01C)
(...)
13:16:15.792: ldp: Sent label mapping msg to 10.10.10.77:0 (pp 0x6306D01C)
13:16:15.792: ldp: Rcvd address msg from 10.10.10.77:0 (pp 0x6306D01C)
13:16:15.792: ldp: Rcvd label mapping msg from 10.10.10.77:0 (pp 0x6306D01C)
13:16:15.792: ldp: Rcvd label mapping msg from 10.10.10.77:0 (pp 0x6306D01C)
(...)
```

[6] The notation `zam57X` means any of the routers in the testbed: `zam576`, `zam577`, `zam578` or `zam579`.

```
13:16:15.796: ldp: Rcvd label mapping msg from 10.10.10.77:0 (pp 0x6306D01C)
13:17:01.411: ldp: Sent keepalive msg to 10.10.10.77:0 (pp 0x6306D01C)
13:17:08.207: ldp: Rcvd keepalive msg from 10.10.10.77:0 (pp 0x6306D01C)
13:17:52.409: ldp: Sent keepalive msg to 10.10.10.77:0 (pp 0x6306D01C)
(...)
```

Here, the **Initialize**, **Keepalive**, **Address** and **Label_Mapping** messages described in Section 4.2.2 are denoted as "`init msg`", "`keep alive msg`", "`address msg`" and "`label mapping msg`", respectively.

We can see that the **Initialize** messages are exchanged first. They contain the LDP session parameters like the time-out for receiving **Keepalive** messages (default: 180 s) or the flag for the label distribution mode (default: downstream unsolicited). The time interval between sending the **Keepalive** messages is set by the system to 1/3 of the time-out and in the discussed example it was equal to 60 s. However, one can modify the default value for the time-out (for example to 360 s) by executing the following command in the general configuration mode:

```
zam57X(config)#mpls ldp holdtime 360
```

After this change the new value for the time interval between sending **Keepalive** messages is set by the router. Neither too high nor too low values of the holdtime are advisable. In the first case, the router would be too slow in detecting any link congestion or failure condition. In the second case, it might wrongly conclude that the link can not transmit data even if it is only affected by a short burst, which prevents the transmission of the **Keepalive** message on time. One can see the actual LDP parameters used by the router by executing the following command in the exec mode:

```
zam576#show mpls ldp parameters
Protocol version:  1
Downstream label pool:  min label:  16; max label:  100000
Session hold time:  360 sec; keep alive interval:  120 sec
Discovery hello:  holdtime:  15 sec; interval:  5 sec
Discovery targeted hello:  holdtime:  180 sec; interval:  5 sec
(...)
LDP loop detection:  off
```

Here, the allowed range for labels, the time-out, and the **Keepalive** message interval value is shown in conjunction with the hold times and hold intervals for **Hello** messages[7]. During the LDP Discovery, the "Hello" messages are sent every 5 s and should be received at least once per 15 s. These two time values can be also modified (for instance to 10 and 30 s respectively) by using the general configuration mode command:

```
zam57X(config)#mpls ldp discovery interval 10
zam57X(config)#mpls ldp discovery holdtime 30
```

The loop detection functionality allows for avoiding loops within LSPs. It is disabled by default. However, one can enable it by executing the following command in the general configuration mode:

```
zam57X(config)#mpls ldp loop-detection
```

A preceding "no" disables this option. The loop detection mechanism makes use of the following two objects included in **Label_Request** or **Label_Mapping** messages:

- Path_Vector object

[7] `Discovery targeted hello` statement refers to the situation where the LDP session is established between peers not directly connected by a link - this is however not the case in the discussed router configuration.

It contains the list of addresses of all nodes the message has already passed. If any node finds its address on this list, it means that the loop is detected.

- Hop_Count object
  It contains the number of hops the message has already passed. If this number reaches the preset value of the maximal number of hops then the node concludes that the loop is detected.

Once MPLS is fully enabled in the testbed one can look into the MPLS forwarding tables. The label forwarding table for the particular configuration of router *zam576* is shown below.

```
zam576#show mpls forwarding-table
Local  Outgoing    Prefix            Bytes tag  Outgoing    Next Hop
tag    tag or VC   or Tunnel Id      switched   interface
16     Pop tag     10.10.10.78/32    0          AT1/0.1     point2point
17     19          10.10.10.79/32    0          Fa0/1       192.168.32.242
       17          10.10.10.79/32    0          AT1/0.1     point2point
18     Pop tag     10.10.10.77/32    0          Fa0/1       192.168.32.242
19     Pop tag     192.168.34.8/30   0          AT1/0.1     point2point
20     Pop tag     192.168.35.8/29   0          Fa0/1       192.168.32.242
21     22          134.94.204.16/28  0          Fa0/1       192.168.32.242
       21          134.94.204.16/28  0          AT1/0.1     point2point
22     Pop tag     134.94.168.0/21   0          AT1/0.1     point2point
23     22          134.94.80.0/24    0          AT1/0.1     point2point
24     Pop tag     192.168.32.24/30  0          Fa0/1       192.168.32.242
25     Pop tag     192.168.32.16/30  0          Fa0/1       192.168.32.242
       Pop tag     192.168.32.16/30  0          AT1/0.1     point2point
26     Pop tag     192.168.32.232/30 0          AT1/0.1     point2point
27     27          192.168.33.8/29   0          Fa0/1       192.168.32.242
       25          192.168.33.8/29   0          AT1/0.1     point2point
```

The forwarding table information is presented in 6 columns.

- The "`Local tag`" is the incoming label assigned by the router.
- The "`Outgoing tag or VC`" is the label advertised by the downstream node (or the ATM VPI/VCI values to reach this node). The notion "`Untagged`" that might appear in this column indicates that all packets with the given incoming label are forwarded without any label (e.g. when the label binding for the given destination was not advertised by any of the neighboring nodes or the next hop for the given destination does not support MPLS). This information might also be an indication for a configuration problem. The notion "`Pop tag`" indicates that the incoming top label will be removed and no new outgoing label will take its place (it can happen when the egress node advertises so called "implicit NULL label" for a given FEC).
- The "`Prefix or Tunnel ID`" denotes the address of the destination network with its subnet mask, or the identity number of the tunnel through which the data are going to be sent.
- The "`Bytes tag switched`" contains the number of Bytes forwarded by means of MPLS.
- The "`Outgoing interface`" field contains the types of outgoing interfaces used in the MPLS forwarding.
- The "`Next hop`" field contains the IP addresses of the outgoing interfaces. Point-to-point links are marked as "`point2point`".

The above MPLS forwarding table matches the content of the conventional IP routing table which can be displayed by the "`zam576#show ip route`" command executed in the EXEC mode.

```
(...)
     192.168.31.0/30 is subnetted, 1 subnets
C       192.168.31.8 is directly connected,                FastEthernet0/0
     10.0.0.0/32 is subnetted, 4 subnets
O E2    10.10.10.78 [110/20] via 192.168.32.10, 00:03:15,          ATM1/0.1
O E2    10.10.10.79 [110/20] via 192.168.32.242, 00:03:15, FastEthernet0/1
                    [110/20] via 192.168.32.10, 00:03:15,          ATM1/0.1
C       10.10.10.76 is directly connected,                        Loopback0
O E2    10.10.10.77 [110/20] via 192.168.32.242, 00:03:15, FastEthernet0/1
     192.168.34.0/30 is subnetted, 1 subnets
O E2    192.168.34.8 [110/20] via 192.168.32.10, 00:03:15, ATM1/0.1
     192.168.35.0/29 is subnetted, 1 subnets
O E2    192.168.35.8 [110/20] via 192.168.32.242, 00:03:15,FastEthernet0/1
     134.94.0.0/16 is variably subnetted, 3 subnets, 3 masks
O E2    134.94.204.16/28
           [110/20] via 192.168.32.242, 00:03:17,          FastEthernet0/1
           [110/20] via 192.168.32.10, 00:03:17,                  ATM1/0.1
O E2    134.94.168.0/21 [110/20] via 192.168.32.10, 00:03:17,      ATM1/0.1
O E2    134.94.80.0/24 [110/20] via 192.168.32.10, 00:03:17,       ATM1/0.1
     192.168.32.0/30 is subnetted, 5 subnets
C       192.168.32.8 is directly connected,                        ATM1/0.1
O       192.168.32.24 [110/2] via 192.168.32.242, 00:03:17,FastEthernet0/1
O       192.168.32.16 [110/2] via 192.168.32.242, 00:03:17,FastEthernet0/1
                      [110/2] via 192.168.32.10, 00:03:17,         ATM1/0.1
O       192.168.32.232 [110/2] via 192.168.32.10, 00:03:17,        ATM1/0.1
C       192.168.32.240 is directly connected,              FastEthernet0/1
     192.168.33.0/29 is subnetted, 1 subnets
O E2    192.168.33.8 [110/20] via 192.168.32.242, 00:03:17,FastEthernet0/1
                     [110/20] via 192.168.32.10, 00:03:17,         ATM1/0.1
O*E2 0.0.0.0/0 [110/1] via 192.168.32.10, 00:03:17,                ATM1/0.1
```

Here, all destination addresses are listed together with the addresses and names of the associated outgoing interfaces through which the packets will be forwarded. The last line corresponds to the default route (all packets with destinations distinct from the ones listed above will be sent through the ATM interface (`192.168.32.10`). The notion "`O`" or "`O E2`" means that a given route was determined by the OSPF protocol while the notion "`C`" indicates a directly connected interface.

One can compare the next hops to which packets with a given destination address will be forwarded in case of MPLS and IP routing. Let us take two example destinations:

- `10.10.10.79/32` - the address of the router *zam577*
  The incoming label for this destination is `17` while the outgoing label advertised by the *zam577* router is `19` or `17`. MPLS forwards all packets with this destination to the Fast Ethernet interface (denoted as `Fa0/1`) having the address `192.168.32.242` or alternatively to the ATM interface (denoted as `AT1/0.1`).
  On the other hand, in the IP forwarding table, there are also two routes to this destination: the primary going through the Fast Ethernet interface (`192.168.32.242`) and the alternative going through the ATM interface (`192.168.32.10`). Thus, one can see that for this destination MPLS and IP forwarding are equivalent concerning both, primary and alternative routes.

- `192.168.34.8/30` - the address of the network connected to the router *zam578*
  The incoming label for this destination is `19` while the outgoing label is implicit NULL (i.e. `Pop tag`. This means that packets will be forwarded through the ATM interface (denoted as `AT1/0.1`) without any label. Also the IP forwarding uses this ATM interface for this destination.

Any time the network topology changes, the OSPF protocol modifies the IP forwarding table.

Since the LDP works in the liberal retention mode, the label forwarding tables follow this change without additional label distribution.

4.2.3 Constraint-Based Routing LDP (CR-LDP)

So far, all LSPs were established following the shortest paths determined by the interior gateway protocol (like OSPF). The Constraint-based Routing Label Distribution Protocol (CR-LDP), described in this section, extends the functionality of LDP by the ability to explicitly map LSPs to the topology (in a strict or a loose way). The basic document with the description of CR-LDP is RFC3212.

General description

The extension to LDP was done by defining a set of new TLV objects which may be included in the LDP "Label_Request" message. Although the preferred path or the QoS requirements are specified by the head-end router, they influence the behavior of all the routers along the path. Hence, all other routers are not allowed to make any label binding before they get those constraints from the head-end node. This implies that CR-LDP must work in the ordered downstream on-demand mode described in Section 4.

There are several new objects defined in the CR-LDP extension which can be included in the "Label_Request" message:

- `LSPID` (mandatory in CR-LDP)
  This object is used to identify the LSP within the MPLS domain. It contains the LSP identification number (locally unique within the head-end router) and the head-end router identification number (one of its IP addresses). The "Action Indicator Flag" carries the information whether the LSP of a given identification number should be created or the existing path should be modified.

- `EXPLICIT_ROUTE` (optional in CR-LDP)
  It is a list of abstract nodes which should be passed over by an LSP, where the term abstract node refers to a single node or a group of nodes described by a common IP prefix or an Autonomous System Number. An already existing LSP tunnel can also be the abstract node in this list.

  Each abstract node is marked by the "Strict" or "Loose" flag. The term "Strict" or "Loose" refers to a path from the previous abstract node to the abstract node carrying the flag.
  If the abstract node is marked as "Strict", then this path must not include other nodes than those belonging to it or to the prior abstract node.
  If the abstract node is marked as "Loose", then the path may include other nodes than those belonging to it or to the prior abstract node.

- `TRAFFIC_PARAMETER` (optional in CR-LDP)
  This object contains the characteristics of the traffic to be routed through the LSP. For this purpose the following token bucket parameters are used:

  - Committed Data Rate (CDR)
    The maximum average rate at which the data will flow through the LSP.
  - Committed Burst Size (CBS)
    The maximum number of extra Bytes which can be sent when the actual rate temporally exceeds the CDR value.
  - Excess Burst Size (EBS)
    The maximum number of uncommitted data bits that the network attempts to deliver over a measured time interval.

  - Peak Data Rate (PDR)
    The maximum rate which can be reached while sending the data through the LSP[8]. The Committed Data Rate alone does not limit the momentary rate of the data transmission.
  - Peak Burst Size (PBS)
    The maximum number of extra Bytes which can be sent during a short time when the actual rate exceeds the PDR value.

  In addition, the "Weight" parameter is used to determine the LSP's priority to access the bandwidth.

- `PINNING` (optional in CR-LDP)
  This object contains a bit which selects whether the route pinning is enabled or disabled for an LSP. It affects only those segments of the LSP which are loosely routed (when the next hop has a "Loose" flag or it is a group of nodes). When the route pinning bit is set to "1", all loosely routed LSP segments will not be redefined even though some better placement might appear at some later time.

- `RESOURCE_CLASS` (optional in CR-LDP)
  Each link can be attributed to up to 32 user defined resource classes (resource classes are specified in the RFC2702 document). This information is represented by a 32-bit string. Here, every bit indicates whether the link is assigned to a given resource class (the bit set to "1") or not (the bit is set to "0"). Similarly, the `RESOURCE_CLASS` object contains a 32-bit string indicating which resource classes may be used by the LSP. Any time an LSP is being established, the LSP string is compared to the links' bit strings and only those links are selected which belong to the required resource classes. An example for a possible application of this resource class affinity is shown in Figure 4.8.

- `PREEMPTION` (optional in CR-LDP)
  It affects the rearrangements of LSPs. This object includes the parameters: "Setup Priority" and "Hold Priority". Both of them are in the range from 0 to 7. Here, the first value corresponds to the highest priority and the second to the lowest. The priorities are compared when the two LSPs compete for network resources. The "Setup Priority" parameter is associated with the LSPs being established while the "Hold Priority" parameter is used for the LSPs already present.

After the CR-LDP sessions between peered routers in an MPLS domain are established, the head-end node of an LSP can initialize the creation of a constrained LSP. An example showing the operation of CR-LDP during establishment of such an LSP is presented in Figure 4.9.

The head-end router usually adds the `EXPLICIT_ROUTE` and the `TRAFFIC_PARAMETER` objects to the **Label_Request** message. The **Label_Request** message is then sent to the next node which is chosen by taking into account all the constraints and the local routing information. After the reception of the message (and preparing the appropriate network resource reservations if necessary) the next node will proceed with the same procedure and will choose the destination for its **Label_Request** message. Thus, this procedure is continued until the LSP's tail-end node is reached. Then the tail-end creates the label binding which is advertised in the **Label_Mapping** message. A sequence of **Label_Mapping** messages is sent from one node to another along the same path which was previously used by the **Label_Request** messages but in the opposite direction. Finally the head-end node receives the label binding and the constrained LSP is established.

CR-LDP is not implemented in the evaluated Cisco router software.

4.2.4 Resource Reservation Protocol with Traffic Engineering Extensions (RSVP-TE)

The Resource Reservation Protocol with Traffic Engineering extensions described in RFC3209 has a similar functionality as CR-LDP. It extends RSVP described in RFC2205 with a set of new

[8] In fact it is also some average value which can be momentary exceeded in a short time.

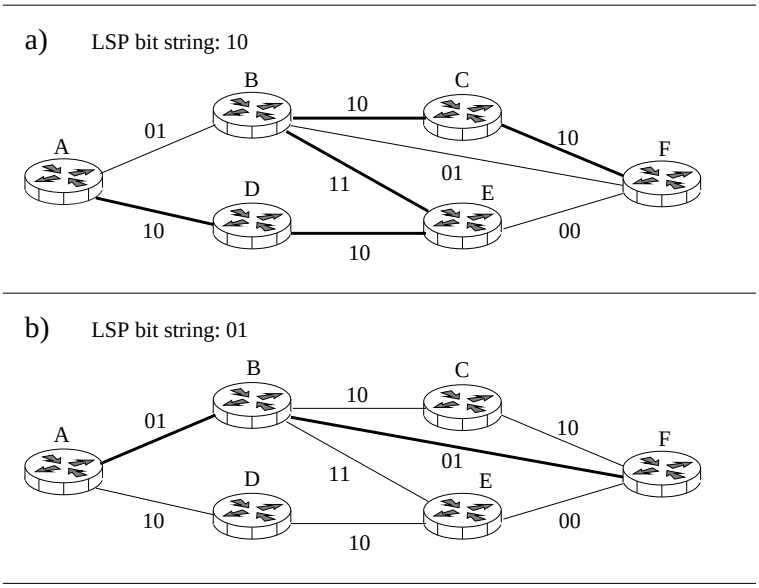

Figure 4.8: The possible application of the RESOURCE_CLASS object in selecting the placement of an LSP is shown. The LSP is going to be established between the routers A and F. For simplicity only two bits, representing two resource classes, are taken into account. Each link belongs to none, one or two resource classes what is indicated by the appropriate bit patterns. **a)** The LSP bit string "10" indicates that only links with the first bit set to "1" can be chosen for establishment of the path. Thus, only one path traversing nodes ADEBCF is possible in this policing (thick line). **b)** The LSP bit string "01"indicates that only links with the second bit set to "1" can be chosen for establishment of the path. Thus, only one path traversing nodes ABF is possible in this policing (thick line).

features supporting the distribution and maintenance of MPLS labels. The protocol uses the downstream on-demand model to create LSPs.

Routers running RSVP-TE communicate via exchanging messages in form of raw IP or UDP packets. The protocol defines 7 types of messages: Path, Resv, PathErr, ResvErr, PathTear, ResvTear, and ResvConf, where each message can contain one or more objects carrying data specific for a given type. The extension of RSVP to RSVP-TE supporting MPLS was done by including new objects classes into the existing message types.

General description

The most important messages of RSVP are the Path and the Resv message. To illustrate their usage for label distribution, let us consider the example of creating an LSP in an environment of four routers as shown in Figure 4.10. Let us also assume that the LSP should follow the path from the router A to the router D going through the routers B and C. Denoting the path in the explicit route object one can enforce the placement of an LSP independently from the path calculated by the shortest path algorithm.

When the LSP is to be established between routers A and D the router A must send a Path message containing the LABEL_REQUEST object to the router D. The packet containing this message is first received by the router B which adds its address to the RECORD_ROUTE objects and forwards the modified Path message further to the router C. The RECORD_ROUTE object keeps the actual information of the path traversed by the messages[9]. Router C forwards the message further to the destination router D. In response to the LABEL_REQUEST included in the Path message, the

[9]The presence of the RECORD_ROUTE object in a Path message is optional.

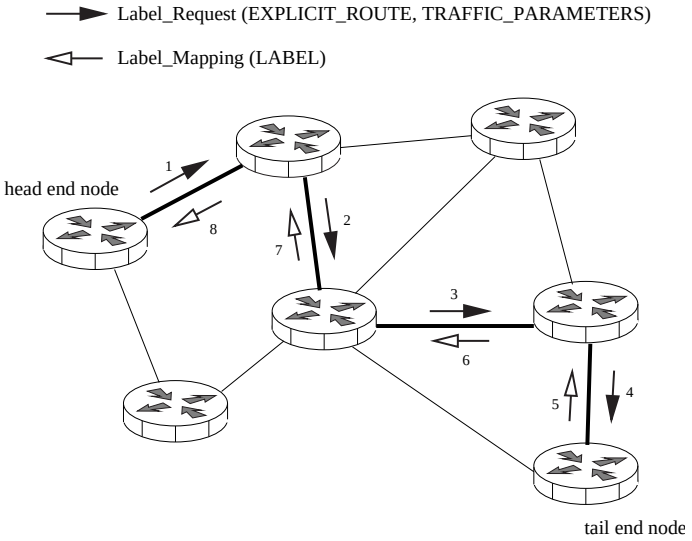

Figure 4.9: This figure shows the flow of the **Label_Request** and **Label_Mapping** messages during establishment of the CR-LSP tunnel. The time sequence of the messages corresponds to the order of the numbers placed at the arrows (1,2, ..., 8). The thick lines denote the Label Switched Path sectors.

router D binds an MPLS label as incoming label in its MPLS forwarding table and puts it in the LABEL object of the Resv message. Then this message is sent back to the router A following the path recorded in the `RECORD_ROUTE` object but in reverse direction. The Resv message is received first by the router C, which extracts the label from the `LABEL` object of the Resv message and stores it in its MPLS forwarding table as outgoing label for the related LSP. Then the router binds[10] the incoming label associated with the just received outgoing label and sends it to the router B in a `LABEL` object of the Resv message. When the router A receives the Resv message the LSP is established. All packets sent by the router A using the just received MPLS label will follow the newly established LSP. This procedure of distributing the MPLS labels is called downstream-on-demand.

CISCO implementation

The tests of RSVP-TE were performed in a network testbed consisting of four Cisco 7200 routers connected via Gigabit Ethernet, Fast Ethernet, and ATM links (see Figure 4.7). RSVP-TE is used by the Cisco software installed in the routers to support the creation and maintenance of MPLS tunnels. The setup of an MPLS tunnel is initiated by commands which trigger the RSVP-TE message flow along the tunnel path. A detailed view on this procedure can be gained by using the tracing facility of RSVP messages.

Figure 4.11 shows the topology of the testbed which was used to perform the tests. Here, an MPLS tunnel was created from the router *zam576* to the router *zam579*.

Configuration of the MPLS tunnel

In order to use an MPLS tunnel, the head-end router (in this example *zam576*) should be properly configured. The following two commands must be executed in the general configuration mode before defining any tunnel:

```
zam576(config)#ip cef
```

[10]This binding can already be done when the Path message was received.

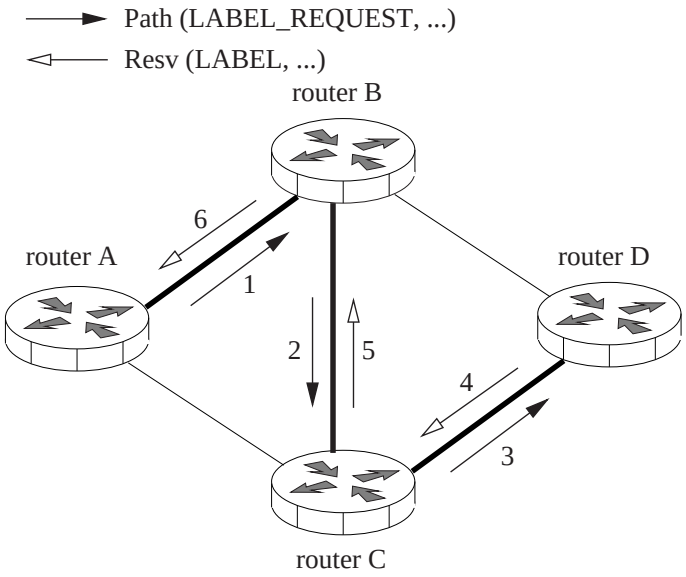

Figure 4.10: An example for the allocation of an MPLS tunnel using RSVP-TE. The `LABEL_REQUEST` objects were included in Path messages while the labels contained in the `LABEL` objects were distributed with the Resv messages. The time sequence of the messages corresponds to the order of the numbers placed at the arrows (1,2, ..., 6). The head-end router A sends the Path message to the router B which is the next hop on the path to the tail-end router D. Then the router B forwards the message to the router C which in turn sends it to the destination router D. Similarly, but in opposite direction, the Resv message generated by the tail-end router D travels the path to the head-end router A. When this procedure is completed, all the routers along the described path have a consistent mapping between incoming and outgoing label for this LSP. The LSP can now be used for sending data from the router A to D.

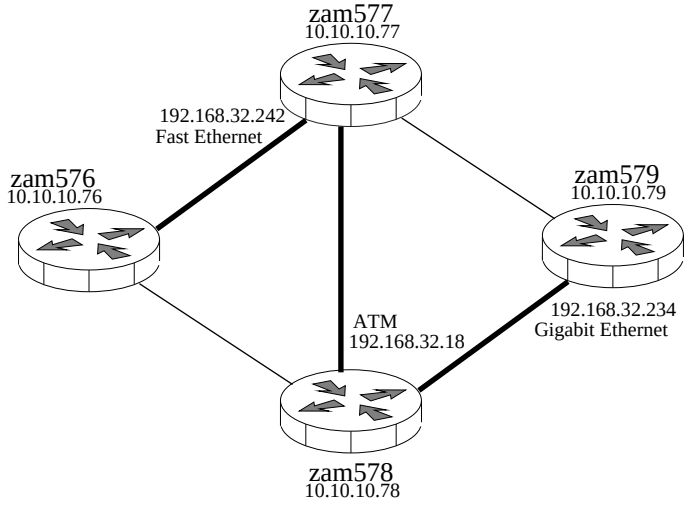

Figure 4.11: Testbed topology for an experiment in which an MPLS tunnel was placed between router *zam576* and router *zam579* (thick line). The interface addresses used to define the explicit path are shown together with the routers' loopback addresses.

```
zam576(config)#mpls traffic-eng tunnels
```

The first command activates the Cisco Express Forwarding (CEF), an advanced switching module which is mandatory for the operation of MPLS. The second command enables RSVP signaling. Additionally, it must be applied to all interfaces which will take part in transmitting MPLS traffic:

```
zam57X(config-if)#mpls traffic-eng tunnels
```

Then an MPLS tunnel interface can be created by entering the interface configuration mode (in this example the tunnel's name is Tunnel1).

```
zam576(config)#interface Tunnel 1
```

Next, Tunnel 1 can be configured by the specific commands in the interface configuration mode. The first one is used to assign an IP address to the tunnel. Here, the tunnel is assigned not to a separate address but to the unique IP address of the router, i.e. its Loopback interface:

```
zam576(config-if)#ip unnumbered Loopback 0
```

Next, one has to specify the destination address of the tunnel. Note that Cisco recommends to use the loopback address of the tail-end router (in this example it is 10.10.10.79):

```
zam576(config-if)#tunnel destination 10.10.10.79
```

Since tunnels, in general, can be implemented in various ways, one has to select the MPLS mode explicitly:

```
zam576(config-if)#tunnel mode mpls traffic-eng
```

The explicit path for this tunnel is identified by the `MYPATH` string (the explicit path, identified by this string, must be defined in the general configuration mode):

```
zam576(config-if)#tunnel mpls traffic-eng path-option 1 explicit name MYPATH
```

Here, the `path-option` value of 1 was chosen to enforce the prioritisation of this particular path. This prioritisation is introduced because of the possibility to define multiple paths for a given tunnel with different `path-option` values. Normally, the option with the lowest value is used. However, sometimes it is not possible that the tunnel is established with this `path-option` value, e.g. due to a link failure. Then the other paths are checked one by one, in ascending order of the `path-option` value, until the available one is found. When the primary path becomes available, the tunnel is automatically reconfigured to use the original path.

The path may be calculated dynamically, i.e. by using the Shortest Path First algorithm. In this case, the argument `explicit` from the command above should be replaced by the word `dynamic`. No further arguments are necessary. Note that it is always a good practice to define the last optional path, i.e. the one with the highest `path-option` value, as a dynamical one:

```
zam576(config-if)#tunnel mpls traffic-eng path-option 2 dynamic
```

The explicit path must be defined in the general configuration mode. The example of the MYPATH definition is cited below. The created tunnel is forced to pass the listed interfaces.

```
zam576(config)#ip explicit-path name MYPATH enable
zam576(cfg-ip-expl-path)#next-address 192.168.32.242
```

```
zam576(cfg-ip-expl-path)#next-address 192.168.32.18
zam576(cfg-ip-expl-path)#next-address 192.168.32.234
```

Path and Resv messages

Using the debugging capabilities offered by the Cisco software one can record the information about incoming and outgoing messages. In order to enable the logging of the RSVP events the following command must be executed:

```
zam576#debug ip rsvp detail
```

The recorded logging can be viewed with the command:

```
zam576#show logging
```

The system summarizes the information contained in a Path message as follows[11].

```
14:23:08.022: RSVP session 10.10.10.79_1: Sending PATH message for
              10.10.10.79 on interface FastEthernet0/1
14:23:08.022: RSVP:      version:1 flags:0000 type:PATH cksum:7088
              ttl:254 reserved:0 length:212
14:23:08.022:  SESSION              type 7 length 16:
14:23:08.022:    Destination 10.10.10.79, TunnelId 1, Source
                 10.10.10.76
14:23:08.022:  HOP                  type 1 length 12: C0A820F1
14:23:08.022:                                      : 00000000
14:23:08.022:  TIME_VALUES          type 1 length 8 : 00007530
14:23:08.022:  EXPLICIT_ROUTE       type 1 length 44:
14:23:08.022:    (#1) Strict IPv4 Prefix, 8 bytes, 192.168.32.242/32
14:23:08.022:    (#2) Strict IPv4 Prefix, 8 bytes, 192.168.32.18/32
14:23:08.022:    (#3) Strict IPv4 Prefix, 8 bytes, 192.168.32.233/32
14:23:08.022:    (#4) Strict IPv4 Prefix, 8 bytes, 192.168.32.234/32
14:23:08.022:    (#5) Strict IPv4 Prefix, 8 bytes, 10.10.10.79/32
14:23:08.022:  LABEL_REQUEST        type 1 length 8 : 00000800
14:23:08.022:  SENDER_TEMPLATE      type 7 length 12:
14:23:08.022:    Source 10.10.10.76, tunnel_id 6
```

The refresh time used during the RSVP session is carried by the `TIME_VALUES` object. The hexadecimal value of 0x00007530 is the refresh time parameter given in milliseconds what is equivalent to $R = 30$ seconds. This is the default value used by the Cisco software. However, it can be separately chosen for each host. The actual refresh time (the time between sending two subsequent Path or Resv messages) changes after sending every message and its new value is chosen from the range of $[0.5R, 1.5R]$. The refresh time is randomly set to avoid any synchronization effects from messages sourced by different nodes within the network. If the gap between the reception of two subsequent Path or Resv messages is more than the life time defined as $L = (K + 0.5) \cdot 1.5R$, it is assumed that the defined MPLS tunnel is not valid anymore. In this case, the LSP is cancelled (K is an integer value and it is currently suggested in RFC2205 to be equal to 3) by propagating PathTear and ResvTear messages downstream and upstream the LSP, respectively.

The `EXPLICIT_ROUTE` object contains a list of addresses of the interfaces which must be passed on the way to the tail-end router. This list is slightly different from the one specified by the `MYPATH` variable. The address of the Gigabit Ethernet port of the *zam578* router is added (192.168.32.233) together with the loopback address of the destination router (10.10.10.79). This is, however, a formal change which does not influence the actual path of the data flow. The `EXPLICIT_ROUTE`

[11] Since the reservation of the network resources is discussed in the next section the related objects are omitted in the description of the RSVP message content here.

object is being updated along the path, to eliminate interfaces which have already been passed. Note that the Cisco implementation uses the EXPLICIT_ROUTE object even if the path is calculated dynamically. This indicates that the Shortest Path First algorithm is applied by the head-end router prior to submitting the Path message.

The router *zam576* receives a Resv message with the following content:

```
14:32:01.914: RSVP:      version:1 flags:0000 type:RESV cksum:0000
              ttl:255 reserved:0 length:108
14:32:01.914:  SESSION              type 7 length 16:
14:32:01.914:    Destination 10.10.10.79, TunnelId 1, Source  10.10.10.76
14:32:01.914:  HOP                  type 1 length 12: C0A820F2
14:32:01.914:                                      : 00000000
14:32:01.914:  TIME_VALUES          type 1 length 8 : 00007530
14:32:01.914:  LABEL                type 1 length 8 : 00000010
14:32:01.914:
14:32:01.914: RSVP 10.10.10.76_6-10.10.10.79_1: RESV message
              for 10.10.10.79 (FastEthernet0/1) from 192.168.32.242
```

Some parameters like SESSION, HOP and TIME_VALUES have the same meaning as we described in case of Path messages.

For MPLS the most important object is the LABEL object which contains the value of the MPLS label. In the example for the head-end *zam576* router it equals to 16 (10 in the hexadecimal system). The routers *zam577* and *zam578* received from their downstream nodes the labels 3 and 0 respectively.

All the nodes along the LSP periodically send the Path and Resv messages to their neighbors with the average frequency of $R = 30$ s.

PathTear and ResvTear messages

RSVP-TE maintains a "soft state". The state information needs to be periodically refreshed by the Path and Resv messages. Otherwise, the LSP will be automatically deleted. Thus, the use of the special PathTear and ResvTear commands for an efficient cancellation of an LSP is recommended, but not obligatory in the operation of RSVP.

PathTear and ResvTear messages travel along the LSP in analogy to the Path and Resv messages: PathTear in the direction of the tail-end and ResvTear in the direction of the head-end. The messages contain the same objects as included in Path and Resv messages but the triggered action is opposite: instead of establishing an LSP the LSP is deleted.

The sending of tear down messages can be initialized anywhere along the LSP. It thus has not to be sourced by either the head-end or the tail-end router. The reason for that is clear: a timeout in receiving refresh Path or Resv messages might occur in any node along an LSP.

Below is the example of the PathTear message sent by the *zam576* head-end router to the *zam577* router which is the next hop along the defined LSP.

```
15:22:52.666: RSVP session 10.10.10.79_1: send path teardown  multicast
              about 10.10.10.79 on FastEthernet0/1
15:22:52.666: RSVP:      version:1 flags:0000 type:PTEAR cksum:9CFE
              ttl:254 reserved:0 length:132
15:22:52.666:  SESSION              type 7 length 16:
15:22:52.666:    Destination 10.10.10.79, TunnelId 1, Source  10.10.10.76
15:22:52.666:  HOP                  type 1 length 12: C0A820F1
15:22:52.666:                                      : 00000000
15:22:52.666:  SENDER_TEMPLATE      type 7 length 12:
15:22:52.666:    Source 10.10.10.76, tunnel_id 60
```

In response to the PathTear message the router *zam577* sends the ResvTear message back to *zam576*:

```
15:22:52.677: RSVP session 10.10.10.79_1: send reservation teardown
              to 192.168.32.241 about 10.10.10.79
15:22:52.677: RSVP:  version:1 flags:0000 type:RTEAR cksum:1ACE  ttl:255
              reserved:0 length:100
15:22:52.677:  SESSION              type 7 length 16:
15:22:52.677:        Destination 10.10.10.79, TunnelId 1,
                     Source  10.10.10.76
15:22:52.677:  HOP                  type 1 length 12: C0A820F2
15:22:52.677:                                       : 00000000
15:22:52.677:  CONFIRM              type 1 length 8 : C0A820F2
```

PathErr and ResvErr messages

When an error is detected by any router during the processing of an LSP setup request, it sends RSVP-TE error messages. A PathErr message is sent upstream to the head-end when the problem was caused by the Path message; a ResvErr message is sent downstream to the tail-end if the problem was related to the Resv message.

4.3 MPLS protection capabilities

A major benefit of MPLS is caused by its path-orientation. MPLS can potentially provide faster and more predictable protection and restoration capabilities in the face of topology changes than conventional hop-by-hop routed IP systems. Whenever a link fails, routing protocols are involved to recover the situation and to reestablish the routing convergence again. Unfortunately, the link-state protocols such as Open Shortest Path First (OSPF) do use timer mechanisms with exponential backoffs to avoid the frequent flooding of link-state changes. Even in an excellently tuned system, the recovery time is of the order of seconds. Note that this time moght even be increased by the time it takes to expose the interface error to the IP-layer. Multiple “Hello”-messages are involved to identify a broken link, which again takes of the order of seconds. Finally, the shortest path calculation and the flooding of the new link state takes a while. MPLS offers three particular recovery mechanisms, all involving the use of a pre-configured, but unused switched path: a backup tunnel.

- Link protection uses a backup tunnel which connects to the peered router of the protected link. In case of failures, the upstream router simply pushes an additional label to the MPLS label stack and transmits the packets to the backup tunnel. On the receiving router, so-called global labels are used to accept the arriving packets of different switched paths. The underlying idea is to forward a given label independently from the incoming interface. Note that a single link protection is in principle capable of protecting multiple switched paths.
- Node protection follows a similar concept, but connects to the downstream router of the peered node. It therefore ensures that any traffic from an LSP traversing the protected router can continue to reach its destination even if the router fails.
- With path protection the backup tunnel covers major parts of the switched path using a diversely-routed path. Note that path protection does not necessarily connect the ingress router with the egress router.

The most efficient recovery approach is link protection. When applied to Packet Over SONET (POS) interfaces, which provide convenient feedback mechanisms for link failure detection, recovery can be kept within strong time constraints.

Chapter 5

Service Provisioning in MPLS-Domains

5.1 Bandwidth reservation mechanisms

RSVP is a signaling protocol which is used by the Integrated Services architecture. It is therefore capable of transporting information about the traffic profile and the expected level of service. This all is, in principle, also true for the RSVP-TE extensions, since it is possible to associate some bandwidth reservation with the allocation of an LSP.

The Cisco software in the version 12.2(4)T allows to specify classes of MPLS packets which can be offered a special QoS treatment (e.g. dedicated bandwidth, priority queue). However, those classes can be only defined by means of EXP values in an MPLS header. This scheme of ensuring the QoS is called MPLS Differentiated Services (MPLS DS). A tunnel-based classification of MPLS traffic is opaque to this QoS mechanism and one can not assign the QoS parameters to a given tunnel without referring to the EXP field. RSVP-TE can therefore not be used to automatically allocate appropriate resources in the traversed routers.

On the other hand, Cisco implements a bandwidth reservation mechanism which can keep track of available and reserved bandwidth for every interface. A given tunnel must not contain a particular link if the requested bandwidth is more than the available bandwidth on that link[1]. Again we note that this is independent from any scheduling mechanism ensuring an appropriate bandwidth allocation. Moreover, it does not even police tunnel traffic to ensure that the actual usage of the bandwidth conforms the reservation made. Thus, it is clear that in order to create a network environment supporting Quality of Service, the bandwidth reservation mechanism must be combined with QoS mechanisms such as MPLS DS in a consistent way.

There are two bandwidth pools offered for reservations: a main pool and a sub-pool (which is a part of the main pool). The maximal reservable bandwidth should be configured individually for every interface. In order to allocate `<X>` kbits/s in the main pool and `<Z>` kbits/s in the sub-pool, the following command must be executed in the interface configuration mode:

```
zam57X(config-if)#ip rsvp bandwidth <X> sub-pool <Z>
```

It is not required that the `<X>` value matches the link physical bandwidth. In fact, it can range from 1 to 10^8 kbits/s. Since the sub-pool is a part of the pool, the `<Z>` value range is from 0 to `<X>`. If the configured reservable pool or sub-pool bandwidth exceeds the link physical bandwidth then it is said that the link is over-booked.

An MPLS tunnel can request bandwidth from either the main pool or the sub-pool of every interface along the path of the tunnel. The following commands performed in the tunnel interface modes show the two variants of the reservation:

[1] For simplicity, it is assumed here that all tunnels were defined with the same priority.

```
zam57X(config-if)#tunnel mpls traffic-eng bandwidth <x>
zam57X(config-if)#tunnel mpls traffic-eng bandwidth sub-pool <z>
```

where <x> and <z> parameters are given in kbits/s and relate to the main pool and sub-pool reservations. The allowed range for both parameters is limited by 0 and $2^{32} - 1 = 4\,294\,967\,295$ kbits/s. The tunnel can not be established if there is not enough bandwidth from the given pool on all interfaces along the path. A given tunnel can, however, preempt other tunnels (and obtain the bandwidth assigned to them) if they have lower hold priorities than the set-up priority of the particular tunnel. The values of the <hold> and <setup> priorities are used for this purpose. They can be individually configured for every MPLS tunnel by the following command in the tunnel interface configuration mode:

```
zam57X(config-if)#tunnel mpls traffic-eng priority <setup> <hold>
```

Both values can be chosen from the range between 0 and 7, where the first value specifies the highest priority and the second one the lowest priority.

The amount of bandwidth requested by a tunnel is signalled in the SENDER_TSPEC object in a RSVP Path message.

```
14:23:08.022:  SENDER_TSPEC         type 2 length 36:
14:23:08.022:    version=0, length in words=7
14:23:08.022:    Token bucket fragment (service_id=1, length=6 words
14:23:08.022:        parameter id=127, flags=0, parameter length=5
14:23:08.022:        average rate=125000 bytes/sec, burst depth=1000
                     bytes
14:23:08.022:        peak rate   =125000 bytes/sec
14:23:08.022:        min unit=124 bytes, max pkt size=1662480416 bytes
14:23:08.022:  SESSION_ATTRIBUTE    type 7 length 20:
14:23:08.022:    setup_pri: 7, reservation_pri: 7 MAY REROUTE
14:23:08.022:    SESSION_NAME:zam576_t1
14:23:08.022:  ADSPEC               type 2 length 48:
14:23:08.022:  version=0  length in words=10
14:23:08.022:  General Parameters  break bit=0  service length=8
14:23:08.022:                                          IS Hops:1
14:23:08.022:               Minimum Path Bandwidth (bytes/sec):12500000
14:23:08.022:                      Path Latency (microseconds):0
14:23:08.022:                                        Path MTU:1500
14:23:08.022:  Controlled Load Service  break bit=0  service  length=0
```

The SENDER_TSPEC object is passed unchanged from the head-end router *zam576* to the tail-end router *zam579*. ¿From all the parameters included in this object only the (‘‘average rate’’) parameter carries meaningful information - the requested bandwidth. Other parameters like (‘‘burst depth’’) and the peak data rate (‘‘peak rate’’) are ignored by the MPLS bandwidth reservation system[2]. In the example above, a bandwidth of 1000 kbits/s is requested. The value of the ‘‘average rate’’ is therefore set to 125000 Bytes/s. On the other hand, the SESSION_ATTRIBUTE object informs about the priorities of the tunnel concerning taking (setup_pri) and holding (reservation_pri) resources. Here, both parameters are set by default to 7, i.e. to the lowest available priority. In the Cisco implementation, the presence of the Controlled Load Service part in the ADSPEC object indicates that the bandwidth reservation is requested from the main pool. On the other hand, if the Guaranteed Service would be present in this object, the requested bandwidth would be taken from the sub-pool.

[2]Those additional parameters are present here in order to ensure the compatibility with the original RSVP object specification.

The bandwidth bookkeeping on a particular interface is made when the router receives the FLOWSPEC object in the Resv message.

```
14:32:01.914:  FLOWSPEC             type 2 length 36:
14:32:01.914:    version = 0 length in words = 7
14:32:01.914:    service id = 5, service length = 6
14:32:01.914:    tspec parameter id = 127, tspec flags = 0,
                 tspec length = 5
14:32:01.914:    average rate = 125000 bytes/sec,
                 burst depth =  1000 bytes
14:32:01.914:    peak rate    = 125000 bytes/sec
14:32:01.914:    min unit = 124 bytes, max pkt size = 1500 bytes
```

The FLOWSPEC object contains the same token bucket parameters as in the SENDER_TSPEC object described above. The average rate parameter is the only meaningful parameter here: it informs about the requested bandwidth. The service id value of 5 (originally encoding Controlled-Load Service) indicates that the bandwidth should be reserved from the main pool. A value of 2 (originally encoding Guaranteed Service) would indicate a bandwidth request from the sub-pool.

5.2 Guaranteed bandwidth - example

This section evaluates the bandwidth bookkeeping function described in the previous section. An MPLS tunnel (Tunnel 1) was established at the router *zam576* with the destination to the router *zam579* (see Figure 5.1).

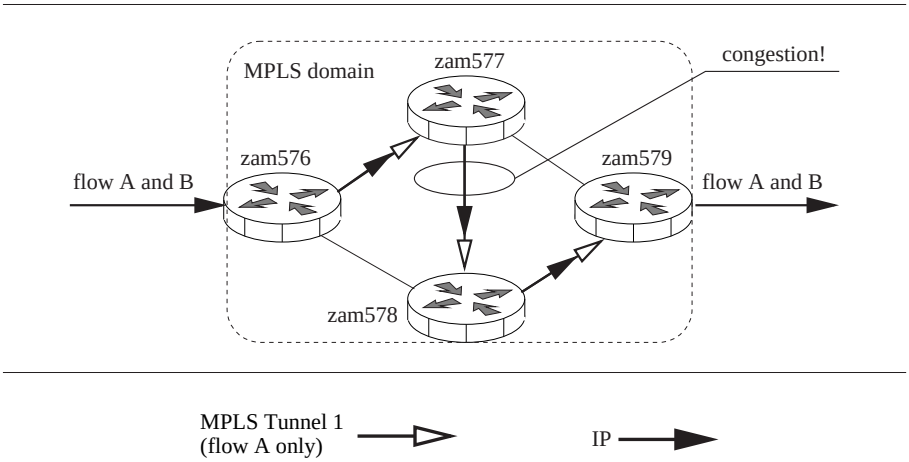

Figure 5.1: Two flows A and B traverse the same path from the ingress router *zam576* to the egress router *zam579*. However, the flow A is forwarded using an MPLS tunnel (Tunnel 1), while flow B is forwarded in a conventional way as IP traffic. The congestion occurs on the ATM link between the router *zam577* and *zam578*.

This tunnel was configured to reserve a bandwidth of 1 Mbit/s from the main pool (the details of the configuration procedure were described in the previous section). On the other hand, this amount of bandwidth was actually guaranteed for all MPLS packets with an EXP field equal to 5 (MPLS Differentiated Service). We therefore actually provided bandwidth guarantees based on a static scheduling configuration. This was achieved by the following configuration of all the interfaces along the tunnel:

```
zam57X(config)#class-map match-all myclass
```

```
zam57X(config-cmap)# match mpls experimental 5
zam57X(config-cmap)#exit

zam57X(config)#policy-map mypolicy
zam57X(config-pmap)#class myclass
zam57X(config-pmap)#priority 1000
zam57X(config-pmap)#exit

zam57X(config)#interface <int>
zam57X(config-if)#service-policy output mypolicy
```

In the above example, we first create a class-map `myclass` which filters all (MPLS) packets which have the EXP field set to 5. Then, we create a policy map `mypolicy` which assigns to all packets associated to the class map `myclass`, i.e. an EXP field of 5, a priority queue with the bandwidth of 1000 kbits/s. At the end, we attach this policy map as a service policy to a given interface `<int>` along the path of the Tunnel 1. As long as packets passing over the tunnel are marked with EXP field set to the value of 5 the tunnel can use the priority queue with the guaranteed bandwidth of 1 Mbit/s.

For the purpose of the test, two flows A (0.8 Mbit/s) and B (3.2Mbit/s) were generated, see Figure 5.1. The flow A was forwarded to the Tunnel 1 by the static routing mechanism[3]. Additionally, all packets belonging to this flow were marked with the EXP value set to 5. Thus, the flow A travelling over the tunnel was prioritized (priority queue is served before any other queue) up to a bandwidth of 1 Mbit/s. At the same time, the flow B was offered a Best-Effort Service.

Note that both flows used the same path from the router *zam576* to router *zam579*. We created a single bottleneck link between router *zam577* and *zam578*. This ATM link was configured to provide a gross bandwidth of 2 Mbits/s (i.e. a net bandwidth of around 1.6 Mbit/s) while the bandwidth demand of both flows was 4 Mbits/s in total.

Figure 5.2 shows the latency and the amount of lost packets for flow A:

- as described above (denoted in Figure as “Priority Queuing”),
- as described above with the difference that the packets from the flow A were not marked with an EXP field of 5 - thus no QoS was offered for the flow (denoted in the Figure as “Priority Queuing”).

When Priority Queuing is applied, the latency remained constant and no packet loss was observed. On the other hand, when the packets did not belong to the appropriate packet class, packets get queued until a tail-drop mechanism results in a constant and significant delay.

5.3 DSCP policy based routing

There are three ways of forwarding packets to MPLS tunnels at an ingress router:

- Static routing
  All traffic to a given destination is mapped to a given tunnel.
- Making the tunnel available to a routing protocol
  A given tunnel is treated by the routing protocol as one of the router interfaces.

[3]It was checked that the Cisco software, in the version which we used, does not support the functionality of the policy-based routing simultaneously with the ability to modify of the EXP field in the MPLS header. Cisco is investigating the problem but does not have a recommended workaround available at the time this report was written

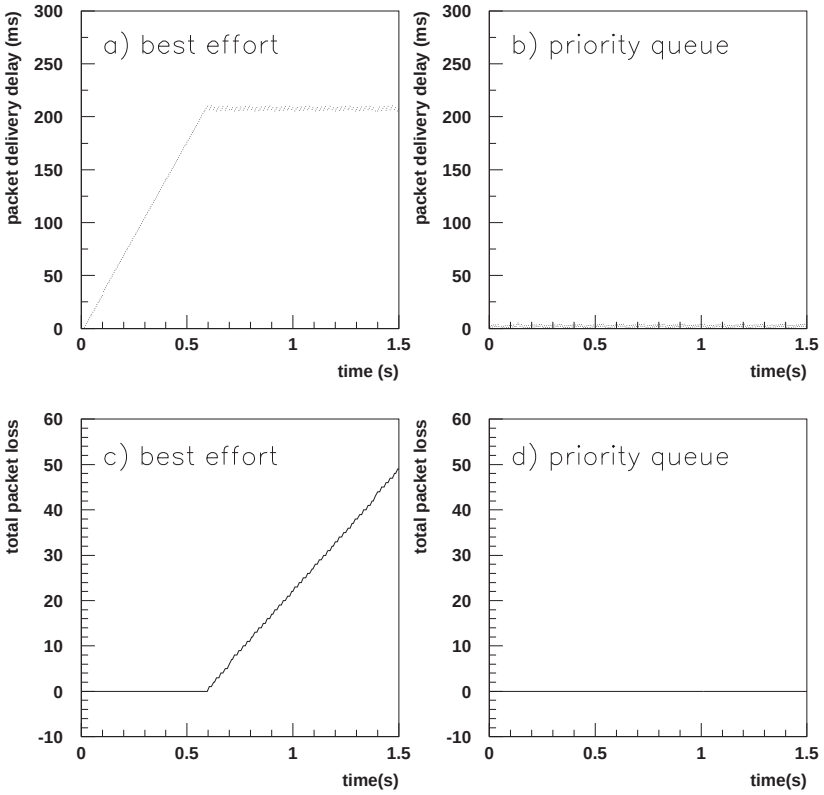

Figure 5.2: Packet delivery delay and total loss of packets as a function of time from the beginning of the data transmission. The packets were sent through an MPLS tunnel defined with (panels **b** and **d**) and without (panels **a** and **c**) QoS mechanisms.

- Policy based routing
  Only selected packets are being forwarded to a given tunnel. The traffic can be filtered according to packets' source address, destination address, source or destination port numbers, IP DSCP field, etc.

This subsection describes the case where the tunnel to which the packets are forwarded is solely chosen based on packets' DSCP field in the IP header (see Figure 5.3). We created two MPLS tunnels with the common head-end of *zam576*: Tunnel 1 and Tunnel 2. The steps needed to configure a tunnel were described in Section 4.2.4.

IP packets entering the domain at the fast Ethernet interface (FastEthernet 0/0) of the router *zam576* were marked by the sending application with the DSCP values of 2 or 3. The policy-based routing was configured in the following way.

Two access lists were defined in the general configuration mode to filter the incoming traffic according to the DSCP field:

```
zam576(config)#access-list 112 permit ip any any dscp 2
zam576(config)#access-list 113 permit ip any any dscp 3
```

The parameters `any any` denote that the IP packets are not filtered according to their source

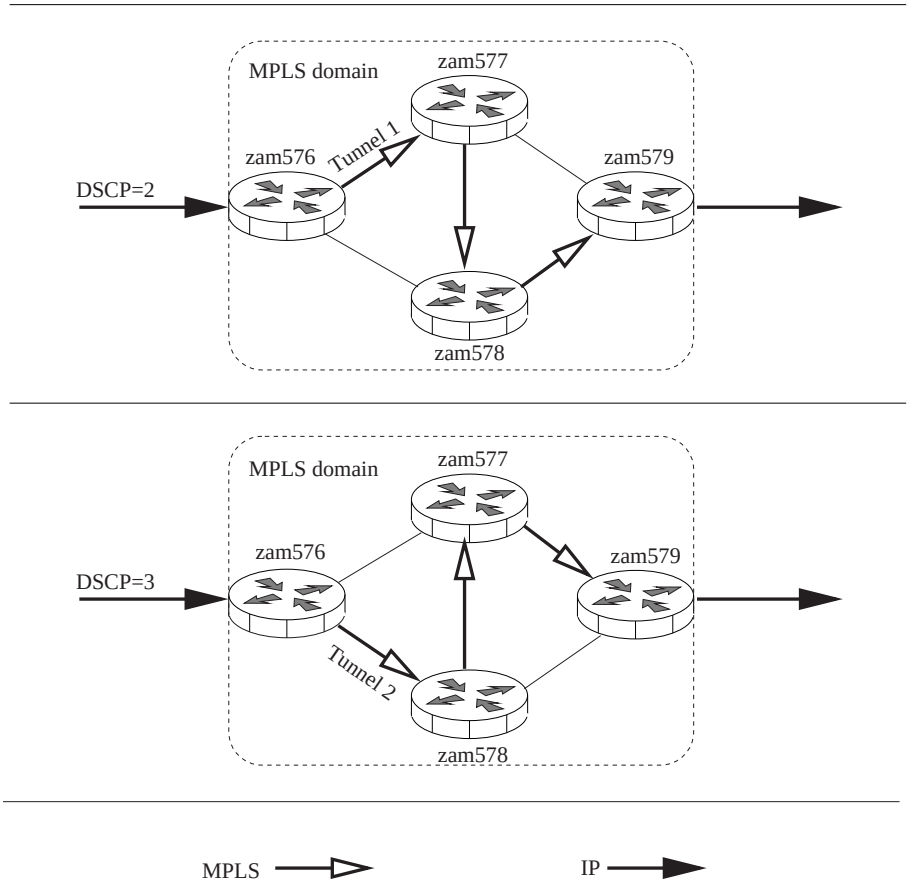

Figure 5.3: DSCP policy based routing. Two tunnels: Tunnel 1 and Tunnel 2 were defined at the ingress router *zam576*. The incoming IP packets were forwarded according to their DSCP values. Packets marked with DSCP equal to 2 were sent to Tunnel 1 while packets marked with DSCP equal to 3 were sent to Tunnel 2.

and destination address. In the next step the route map `myroute` was configured as follows:

```
zam576(config)#route-map myroute permit 1
zam576(config-route-map)# match ip address 112
zam576(config-route-map)# set interface Tunnel1
zam576(config-route-map)# exit
zam576(config)#route-map myroute permit 2
zam576(config-route-map)# match ip address 113
zam576(config-route-map)# set interface Tunnel2
zam576(config-route-map)# exit
```

The route map `myroute` consists of two items which were identified by the last parameter in the `route-map myroute permit <n>` command. In the discussed example `<n>` equals to 1 and 2. Each item is configured separately by specifying the access list which should be matched (`match ip address <nl>`) and the interface (`set interface Tunnel <nt>`). All packets which match the access list `<nl>` were then sent over the tunnel `<nt>`.

The routing policy `myroute` defined as above is not yet assigned to any interface. To install it at the FastEthernet 0/0 interface of the *zam576* router, the following command in the interface

configuration mode must be executed:

```
zam576(config-if)#ip policy route-map myroute
```

After this step the configuration is completed. It is worth noting that the forwarding defined by the `myroute` at the ingress router is not based on the destination address but solely on the DSCP field. Thus, the application which sends DSCP marked packets can influence the way how the packets are forwarded at the ingress router. DSCP encoding of 2 or 3 will result in sending the traffic through the Tunnel 1 and 2, respectively, while other encodings will cause that the traffic will be forwarded traditionally. Moreover, if the tunnels were defined with the QoS parameters (see previous section), the sender application can decide, in this way, whether to use those QoS mechanisms or not. This can be done without changing a configuration of any router in the domain.

5.4 Fast re-routing

A major benefit of the use of MPLS is given by its failure protection capabilities. The fast-reroute feature allows to preserve the connectivity of LSPs for failures of a particular link, a particular node, or a more complex error condition. Especially link protection is designed to perform a quick restoration of connectivity. It also has the scalable property that the configuration of a single backup tunnel is able to protect all LSPs which are traversing the particular link.

The experiments presented in this subsection explore the actual benefit of link protection compared to a failure situation occurred in a typical IP-domain which uses the popular Open Shortest Path First (OSPF) protocol as routing protocol. The layout of the two experiments is as follows, see Figure 5.4.

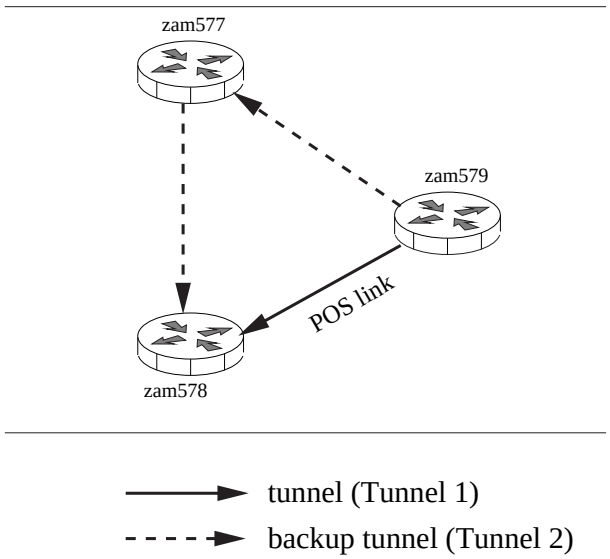

Figure 5.4: Fast re-routing mechanism. Two tunnels are defined at the router *zam579*. The primary Tunnel 1 goes over the Packet Over Sonet link. Under normal conditions, the traffic from the router *zam579* to the router *zam578* uses Tunnel 1 while the backup Tunnel 2 is not used. If the POS link fails the MPLS forwarding at the router *zam579* is automatically modified to use Tunnel 2 instead of Tunnel 1 thus omitting the failure.

Under normal conditions the flow from the router *zam579* to the router *zam578* goes over the

Packet Over Sonet (POS) link using the logical interface Tunnel 1 which is configured as below:

```
zam579(config)#interface Tunnel 1
zam579(config-if)#ip unnumbered Loopback0
zam579(config-if)#no ip directed-broadcast
zam579(config-if)#tunnel destination 10.10.10.78
zam579(config-if)#tunnel mode mpls traffic-eng
zam579(config-if)#tunnel mpls traffic-eng autoroute announce
zam579(config-if)#tunnel mpls traffic-eng path-option 1 explicit name PRIMARY
zam579(config-if)#tunnel mpls traffic-eng fast-reroute
zam579(config-if)#exit
```

The major part of the MPLS tunnel configuration is described in detail in Section 4.2.4. Here, two additional commands were added, marked in bold. The first one causes that Tunnel 1 is announced to the Interior Gateway Protocol (OSPF in this example) as one of the *zam579* interfaces. The second one enables the fast re-routing functionality. The explicit path `PRIMARY` is defined as a direct path between nodes *zam579* and *zam578*.

The backup Tunnel 2 is defined according to the procedure described in Section 4.2.4 with the explicit path `BACKUP` going from the router *zam579* to the router *zam578* not directly, but through the router *zam577*. In this case the tunnel is not advertised to the routing protocol.

```
zam579(config)#interface Tunnel 2
zam579(config-if)#ip unnumbered Loopback0
zam579(config-if)#no ip directed-broadcast
zam579(config-if)#tunnel destination 10.10.10.78
zam579(config-if)#tunnel mode mpls traffic-eng
zam579(config-if)#tunnel mpls traffic-eng path-option 1 explicit name BACKUP
zam579(config-if)#exit
```

During the first experiment, the POS interface was not configured to use any backup path after the failure condition. In the second experiment, the POS interface was configured to use the backup Tunnel 2 whenever a failure occurs. To enable link protection, one has to add the following command in the interface configuration mode, specifying the tunnel which should be used as a backup path.

```
zam579(config)#interface POS 3/0
zam579(config-if)#mpls traffic-eng backup-path Tunnel 2
zam579(config-if)#exit
```

In both experiments the POS link was manually disconnected a few seconds after beginning the data transmission. The flow was sent at the rate of 1 packet per millisecond. Figure 5.5 shows the service offered to the traffic in both cases.

- Experiment 1
  Figures 5.5-a and c illustrate the achieved service when a standard OSPF configuration was deployed. Roughly at the 9-th second, the link failure occurred. IP-convergence took around 5 seconds, after which connectivity was reestablished. As the constant bit rate stream was sending packets with a frequency of 1000 Hz, the overall amount of lost packets was around 5000. The reason for the experienced disruption of connectivity was not caused by the layer 2 failure detection. The POS interface allows rapid failure detection by the explicit configuration of feedback mechanisms, which was done in this experiment. When a link state protocol is used to maintain the routing tables of each router, timers come into place

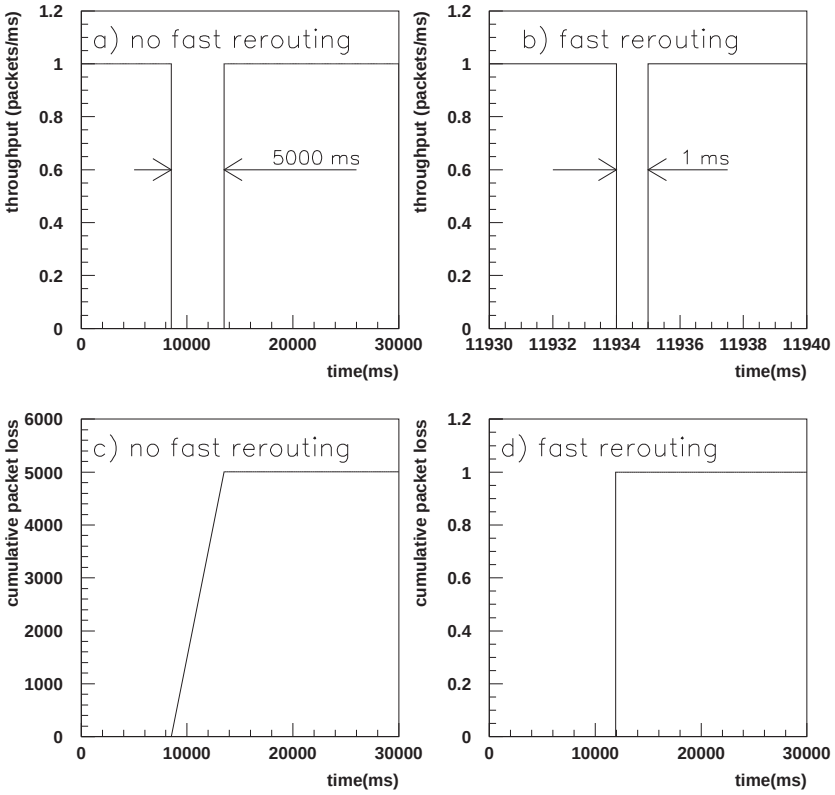

Figure 5.5: Loss of packets due to link failure with (panels **b** and **d**) and without (panels **a** and **c**) fast re-routing functionality. In both cases the flow was sent at the rate of 1 packet per millisecond. The break in transmission was roughly 1 ms (**b**) when the fast re-routing was active and it was roughly 5000 ms (**a**) when this functionality was disabled. That led to the cumulative packet loss of 1 packet(**d**) and ca. 5000 packets (**c**), respectively.

which delay the recalculation of the shortest path. The intention of these timers is to avoid an extreme flooding of routing updates, with the goal to prevent thrashing, especially when the link is flapping. It were exactly these timers which caused the significant disruption of service. Whenever the values of these timers were reduced, it is done at the risk of an extreme overhead of CPU cycles of many routers. A significant decrease of the measured disruption is therefore not a realistic assumption.

- Experiment 2
  Figures 5.5-b and d illustrate the achieved service when link protection was deployed. Roughly at the 12-th second, the link failure occurred. MPLS link protection was able to recover the failure situation quite quickly. The POS interface allowed rapid failure detection by the explicit configuration of feedback mechanisms, which was done in this experiment. In a time frame of 1 ms all traffic was redirected to the backup Tunnel 2. Only one single packet was lost.

The use of the protection capabilities of MPLS is an appropriate mechanism for a robust provisioning of network guarantees.

5.5 Any Transport over MPLS

IP networks transport data from one point to another, usually along the shortest path in some metric. The MPLS extension to IP provides better control of the traffic and thus allows for a more efficient usage of network resources. One of the very promising applications for the MPLS technique is the creation of virtual transport networks from network resources spread among different locations. IP/MPLS networks can transport any data, in particular they can transport packets between any connected networks, even on the data link level. The Any Transport over MPLS (AToM) technique developed by Cisco uses this idea by offering connectivity over an IP/MPLS backbone between transport networks which are not directly connected one to another. Thus, for example geographically separated networks, i.e. Ethernet, can act as a single one although no direct Ethernet connection between them exists.

The important aspect of the AToM solution is that a single IP/MPLS network infrastructure can be used to transport both Layer 2 and Layer 3 traffic. Initially, the Layer 2 Virtual Private Networks (VPN) are built with leased lines or virtual circuits offered by ATM or Frame Relay technologies. Unfortunately, interconnecting networks with this kind of virtual circuits does not scale. The core switches along virtual lines have to store the information about every single virtual circuit. Thus, any time a new connection is added the core switches have to be reconfigured to properly forward new traffic. Moreover, separate networks are usually exploited for Layer 2 and Layer 3 traffic what is not an optimal solution from the network management and economical point of view.

The AToM concept allows for more efficient usage of the network resources offering different network services within a single IP/MPLS infrastructure. Moreover, the AToM solution for building layer 2 VPNs is much more scalable than the conventional one. This is because core routers in the backbone network store information only about the configured MPLS tunnels while every single tunnel can transport many virtual circuits. Thus, there is no need to reconfigure the core routers any time a new virtual connection is added. Only edge routers (head-end and tail-end of the tunnels) should be updated in this case. This significantly simplifies the network management task.

AToM is an open specification which does not restrict a list of potentially supported transport protocols. At the current stage the following protocols are supported by Cisco's AToM solution:

- Ethernet
- ATM AAL5
- Frame Relay
- ATM Cell Relay
- Point-to-Point Protocol (PPP)
- High Level Data Link Protocol (HDLC)

The basic idea of AToM is shown in Figure 5.6. Here, the packet from the Network A1 is forwarded to the Network B1, and similarly for the networks A2-B2 and A3-B3. The letter "h" denotes the header of the packet while the rest is data. In the IP/MPLS backbone this packet is treated as a whole (although some fields in the header are modified) and it is transported through a Label Switched Path towards the destination. Since one tunnel can transport many virtual circuits, the circuits are identified by the second label in an MPLS header stack "L2". This label is added by the "PE1" edge router at the head-end of the tunnel and it is read and interpreted by the "PE2" edge router at the tail-end of the tunnel. From the information carried by the label "L2" the "PE2" router knows to which destination network, e.g. B1, B2 or B3 in our example, a given packet should be sent. All core routers like "PC" look only at "L1" labels which identify MPLS tunnels. This makes that the AToM technique scales quite well with the increasing number of virtual connections.

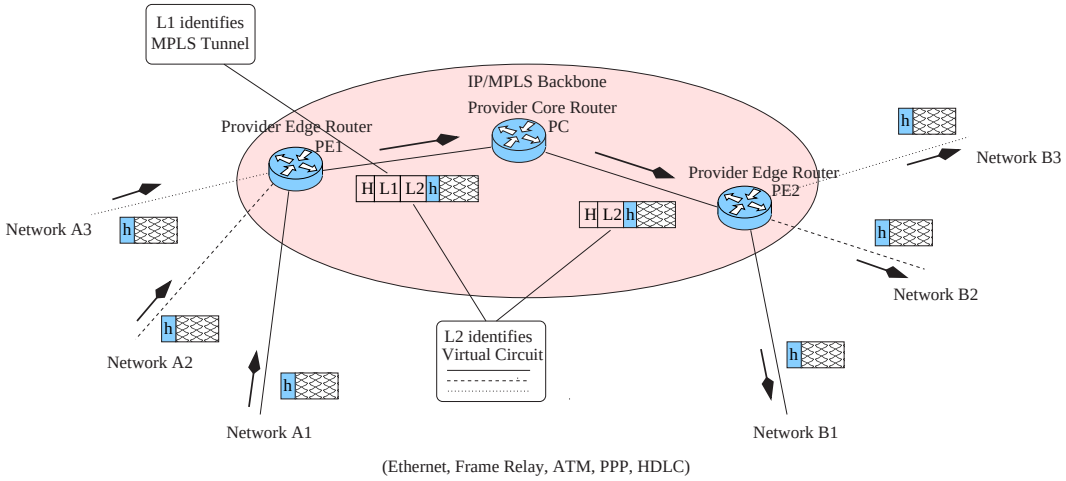

Figure 5.6: The idea of the AToM solution. The IP/MPLS backbone network virtually connects layer 2 networks: A1 with B1, A2 with B2 and A3 with B3. For this purpose only two MPLS connections (two tunnels, one for each direction) are needed, independent of the number of virtual circuits. The MPLS packets are forwarded according to their "L1" labels, while "L2" labels carry information about the virtual circuit. The penultimate router usually removes the "L1" label, thus the tail-end router of the LSP obtains the packet only with the "L2" label which includes the information to which of the B networks a given packet should be forwarded. By "H" the headers of the data link protocols are shown, which enable connectivity between routers in the backbone network.

In the current CISCO implementation Label Switched Paths (LSP) are signaled by the Label Distribution Protocol (LDP). LDP was extended by a new Forwarding Equivalence Class (FEC) element, specific for the virtual network connectivity. It contains information like a kind of transported protocol (Ethernet, PPP, etc...) or an identity number of a virtual circuit.

AToM can be combined with QoS provisioning. Any IP/MPLS network can be configured to offer different QoS treatments to packets with different EXP fields in their MPLS headers. One solution for the QoS policy would be a mapping of a QoS related field from a layer 2 header, if such exists - for instance 802.1P field in the Ethernet header, to the mentioned EXP bits. On the other hand a backbone network provider can apply its own QoS policy to the traffic going through its network.

Currently, the AToM solution is implemented in various CISCO platforms including Cisco 12000 Series and Cisco 7600, 7500 and 7200 routers.

5.5.1 Tests with Ethernet over MPLS

AToM can be used to build virtual networks of several types, as listed in the previous section. This section presents an evaluation of one particular option: Ethernet over MPLS. The tests were made using the MPLS testbed installed at the Research Centre Jülich. It consisted of four Cisco 7200 routers forming an MPLS domain and two Linux machines belonging to the virtual Ethernet network. The topology of the testbed is presented in Figure 5.7. Hosts A and B belong to the same virtual Ethernet network. All frames sent by the host A to the host B go to the input of the `FastEthernet0/0.2` interface of the ingress router PE1. The router is configured to forward all frames to the `FastEthernet0/0.2` interface of the egress router PE2 (within the MPLS domain the frames are transported in dynamically created MPLS tunnels). From there they are sent to the destination host B. Similarly, the frames from the host B are transported to the host A.

Following subsections deliver the details about the router and host configuration.

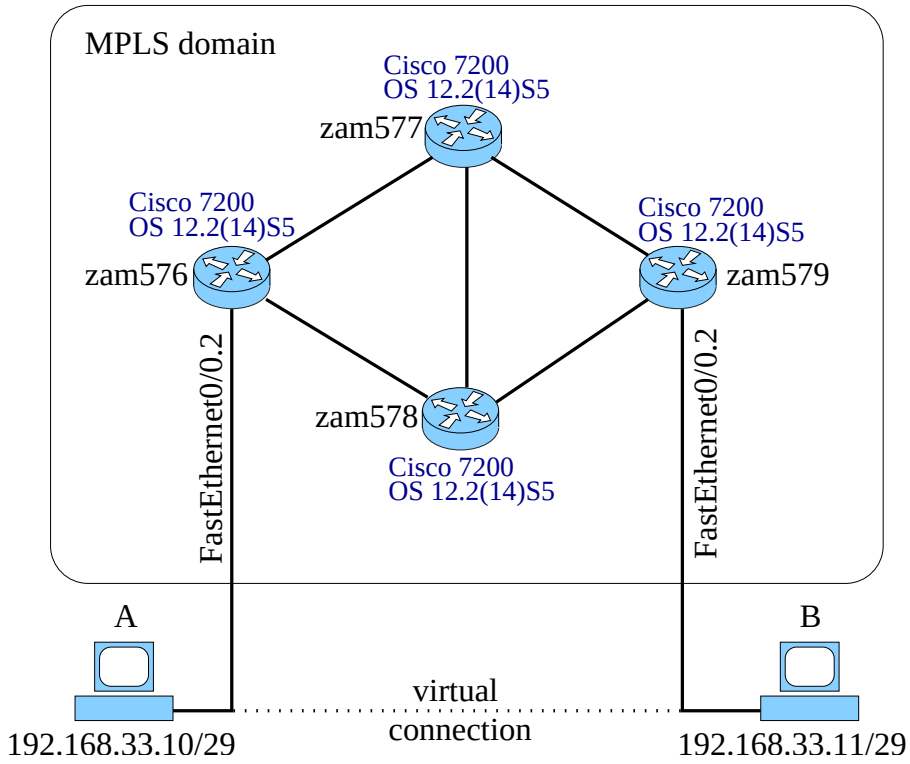

Figure 5.7: Topology of the testbed. Two hosts A and B belong to the same virtual Ethernet network. Although they are not directly connected by any Ethernet link, the AToM technique in the MPLS domain simulates the missing link by transferring whole Ethernet frames through MPLS tunnels.

5.5.2 Ethernet types

Since only *VLAN 802.1q* frames can be transported over an MPLS domain, the hosts with standard setting of their Ethernet cards could not be used in the tests, however, the latest releases of the Linux kernel give support for the VLAN Ethernet. Thus, the operating systems of two Linux hosts from the testbed were updated to make the Ethernet over MPLS evaluation feasible.

The classical *Ethernet II* frame structure and the *VLAN 802.1q* extension is presented in Figure 5.8. Both frames contain preambles, source and destination MAC addresses, type field specifying a layer-3 protocol, data and the checksum. VLAN frames contain 4 additional Bytes following the source address field: 2 Bytes of the VLAN type (`0x8100`) and 2 Bytes of the Tag Control Information containing VLAN ID number.

5.5.3 Configuration

All interfaces in the MPLS domain

AToM uses LDP as a signaling protocol to distribute labels. The first step to activate MPLS is to enable Cisco Express Forwarding by the following command executed in the general *CONFIGURATION* mode on every router in the domain:

```
router(config)#ip cef
```

Next, in order to enable MPLS routing on all interfaces, the two-step configuration described below should be done.

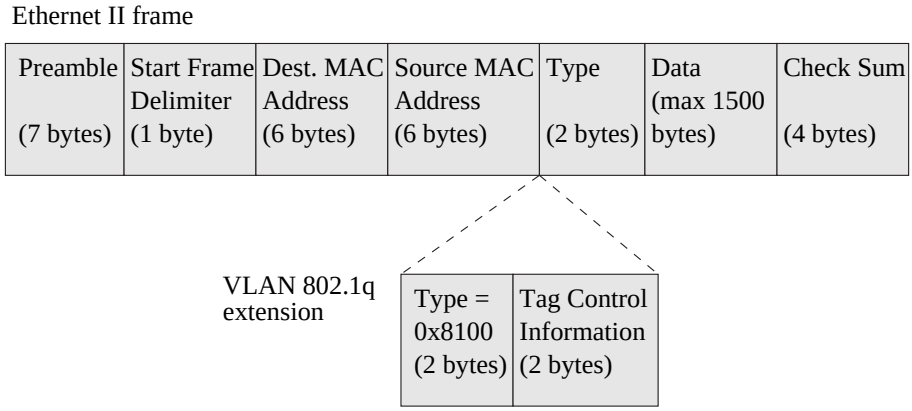

Figure 5.8: Structure of the *Ethernet II* and *VLAN 802.1q* frames.

- In the general configuration mode and in the interface configuration mode:

```
router(config)#mpls ip
router(config-if)#mpls ip
```

- In the general configuration mode and in the interface configuration mode:

```
router(config)#mpls label protocol ldp
router(config-if)#mpls label protocol ldp
```

  In this way LDP is chosen to distribute MPLS labels.

Now, by executing the following command in the *EXEC* mode, one can check whether the LDP peers discovered each other:

```
router#show mpls ldp discovery
```

For example, if the above command is executed on the router *zam577* one gets the following output:

```
 Local LDP Identifier:
    10.10.10.77:0
    Discovery Sources:
    Interfaces:
        FastEthernet0/0 (ldp): xmit/recv
            LDP Id: 10.10.10.76:0
        ATM1/0.1 (ldp): xmit/recv
            LDP Id: 10.10.10.78:0
        ATM4/0.1 (ldp): xmit/recv
            LDP Id: 10.10.10.79:0
```

One can see in this example that the LDP discovery process was successfully completed for the *zam577* router (`10.10.10.77`), since all neighbors were identified: *zam576* (`10.10.10.76`), *zam578* (`10.10.10.78`) and *zam579* (`10.10.10.79`), see Figure 5.7. In this way all links between routers in the domain can be checked.

To make sure whether the LDP sessions were correctly established one uses the command:

```
router#show mpls ldp neighbor
```

Below, again the output for the *zam577* router:

```
Peer LDP Ident: 10.10.10.79:0; Local LDP Ident 10.10.10.77:0
    TCP connection: 10.10.10.79.11005 - 10.10.10.77.646
    State: Oper; Msgs sent/rcvd: 9843/9835; Downstream
    Up time: 5d22h
    LDP discovery sources:
      ATM4/0.1, Src IP addr: 192.168.32.26
    Addresses bound to peer LDP Ident:
      192.168.32.34   10.10.10.79     192.168.32.26   134.94.204.17
Peer LDP Ident: 10.10.10.78:0; Local LDP Ident 10.10.10.77:0
    TCP connection: 10.10.10.78.11003 - 10.10.10.77.646
    State: Oper; Msgs sent/rcvd: 9818/9851; Downstream
    Up time: 5d22h
    LDP discovery sources:
      ATM1/0.1, Src IP addr: 192.168.32.18
    Addresses bound to peer LDP Ident:
      134.94.173.106  192.168.32.10   192.168.32.33   192.168.32.18
      10.10.10.78     192.168.34.9
Peer LDP Ident: 10.10.10.76:0; Local LDP Ident 10.10.10.77:0
    TCP connection: 10.10.10.76.646 - 10.10.10.77.11003
    State: Oper; Msgs sent/rcvd: 1517/1514; Downstream
    Up time: 21:48:56
    LDP discovery sources:
      FastEthernet0/0, Src IP addr: 192.168.32.241
    Addresses bound to peer LDP Ident:
      10.10.10.76     192.168.32.9    192.168.32.241
```

Ingress/egress routers

In the presented tests *zam576* and *zam579* were the edge routers for the MPLS tunnels. The virtual circuit was terminated at the `FastEthernet 0/0.2` subinterfaces of both routers, see Figure 5.7.

Below are the detailed configurations of the *zam576* and *zam579* routers:

- *zam576*

```
interface FastEthernet0/0
 no ip address
 no ip redirects
 no ip proxy-arp
 no ip mroute-cache
 no keepalive
 duplex auto
 speed auto
 mpls label protocol ldp
 tag-switching ip
 no cdp enable
!

interface FastEthernet0/0.2
 encapsulation dot1Q 123
 mpls l2transport route 10.10.10.79 200
 no cdp enable
!
```

Both interface and subinterface have no IP address assigned. The subinterface *FastEthernet0/0.2* uses the 802.1q standard (VLAN) for Ethernet frames with the VLAN ID number set to `123`, see command `encapsulation dot1Q 123` above. A virtual circuit identified by a number of `200` (VC ID) should connect this (*zam576*) router to the *zam579* router (`10.10.10.79`), see the `mpls l2transport route 10.10.10.79 200` command. Cisco Discovery Protocol (CDP) should be disabled on both interface and subinterface.

- *zam579*

```
interface FastEthernet0/0
 no ip address
 no ip redirects
 no ip proxy-arp
 no ip mroute-cache
 no keepalive
 duplex auto
 speed auto
 mpls label protocol ldp
 tag-switching ip
 no cdp enable
!

interface FastEthernet0/0.2
 encapsulation dot1Q 123
 mpls l2transport route 10.10.10.76 200
 no cdp enable
!
```

The configuration is similar to the one of *zam576* router except the `mpls l2transport route 10.10.10.76 200` line which builds a connection to the *zam576* router. Note, that VLAN ID (`123`) respective VC ID (`200`) should be identical for both routers.

To check whether the configuration of VC was successful one can execute the following command on the *zam576* and *zam579* edge routers in the *EXEC* mode:

```
router#show mpls l2transport vc detail
```

The example output of this command on the *zam576* router gives an indication that the virtual circuit to the *zam579* router is operational:

```
Local interface: Fa0/0.2 up, line protocol up, Eth VLAN 123 up
  Destination address: 10.10.10.79, VC ID: 200, VC status: up
    Tunnel label: 17, next hop 192.168.32.242
    Output interface: Fa0/1, imposed label stack {17 27}
  Create time: 1d04h, last status change time: 22:41:51
  Signaling protocol: LDP, peer 10.10.10.79:0 up
    MPLS VC labels: local 25, remote 27
    Group ID: local 1, remote 1
    MTU: local 1500, remote 1500
    Remote interface description:
  Sequencing: receive disabled, send disabled
  VC statistics:
    packet totals: receive 92628, send 97611
    byte totals:   receive 10047130, send 10825718
    packet drops:  receive 0, send 0
```

The same command executed on the *zam579* router gives a similar output:

```
Local interface: Fa0/0.2 up, line protocol up, Eth VLAN 123 up
  Destination address: 10.10.10.76, VC ID: 200, VC status: up
    Tunnel label: 18, next hop point2point
    Output interface: AT1/0.1, imposed label stack {18 25}
  Create time: 1d04h, last status change time: 22:48:52
  Signaling protocol: LDP, peer 10.10.10.76:0 up
    MPLS VC labels: local 27, remote 25
    Group ID: local 1, remote 1
    MTU: local 1500, remote 1500
    Remote interface description:
  Sequencing: receive disabled, send disabled
  VC statistics:
    packet totals: receive 98048, send 93082
    byte totals:   receive 10482630, send 10468378
    packet drops:  receive 0, send 0
```

End systems

Two linux machines were used as the end systems in the performed tests, see Figure 5.7. The AToM technique does not support an Ethernet II but only Ethernet VLAN 802.1q frame transport. Thus, Ethernet VLAN subinterfaces had to be configured in both systems[4].

The configuration steps were as follows:

- **Creating the VLAN subinterface**. Initially, the systems had only standard Ethernet `eth0` network devices, as can be seen from the output of the `ifconfig` command:

  ```
  linux# ifconfig
  eth0      Link encap:Ethernet  HWaddr 00:D0:59:D9:5E:C5
            inet6 addr: fe80::2d0:59ff:fed9:5ec5/64 Scope:Link
            UP BROADCAST NOTRAILERS RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:1049751 errors:0 dropped:0 overruns:0 frame:0
            TX packets:572912 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:100
            RX bytes:136272414 (129.9 Mb)  TX bytes:51463589 (49.0 Mb)
            Interrupt:11 Base address:0x7000 Memory:c0200000-c0200038

  lo        Link encap:Local Loopback
            inet addr:127.0.0.1  Mask:255.0.0.0
            inet6 addr: ::1/128 Scope:Host
            UP LOOPBACK RUNNING  MTU:16436  Metric:1
            RX packets:80 errors:0 dropped:0 overruns:0 frame:0
            TX packets:80 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:6172 (6.0 Kb)  TX bytes:6172 (6.0 Kb)

  linux#
  ```

  Then, a new subinterface supporting Ethernet VLAN can be added to the system by the following command:

  ```
  linux# vconfig add eth0 123
  Added VLAN with VID == 123 to IF -:eth0:-
  linux#
  ```

[4]Note that older Linux kernels do not support Ethernet VLAN

When the subinterface is no longer needed it can be deleted by the command below:

```
linux# vconfig rem eth0.123
Removed VLAN -:eth0.123:-
linux#
```

After creation of a subinterface it is shut down by default. To make it operational one should use the `ifconfig` command as stated below:

```
linux# ifconfig eth0.123 up
linux#
```

- **Assigning an IP address to the Ethernet VLAN subinterface**. Till now, the `eth0.123` subinterface has no assigned IP address. In the tests the host A got the address `192.168.33.10`. This was done by the command:

```
linux# ip addr add 192.168.33.10/29 dev eth0.123
linux#
```

Then one can check the status of the new created `eth0.123` subinterface as well as `eth` and `lo` interfaces:

```
linux# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:D0:59:D9:5E:C5
          inet6 addr: fe80::2d0:59ff:fed9:5ec5/64 Scope:Link
          UP BROADCAST NOTRAILERS RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1268098 errors:0 dropped:0 overruns:0 frame:0
          TX packets:700763 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:164634226 (157.0 Mb)  TX bytes:62876810 (59.9 Mb)
          Interrupt:11 Base address:0x7000 Memory:c0200000-c0200038

eth0.123  Link encap:Ethernet  HWaddr 00:D0:59:D9:5E:C5
          inet addr:192.168.33.10  Bcast:0.0.0.0  Mask:255.255.255.248
          inet6 addr: fe80::2d0:59ff:fed9:5ec5/64 Scope:Link
          UP BROADCAST NOTRAILERS RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:320 (320.0 b)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:80 errors:0 dropped:0 overruns:0 frame:0
          TX packets:80 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:6172 (6.0 Kb)  TX bytes:6172 (6.0 Kb)
linux #
```

- **Controlling the routing table**. Then, after all configuration tasks described above, the routing table can be checked by the `route` command:

```
linux # route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.33.8    *               255.255.255.248 U     0      0        0 eth0.123
linux #
```

The above configuration steps were applied to both linux machines A and B. The assigned IP addresses were `192.168.33.10` and `192.68.33.11`, respectively.

5.5.4 Virtual Ethernet network

The connectivity between host A and B belonging to the same virtual Ethernet network was checked with the network traffic monitoring application *ethereal*. It allows to list all packets coming to and going out from a particular network interface, e.g `eth0.123` in our case. A simple test with ping packets sent from host A to B was performed. Figure 5.9 shows the screen dump of the *ethereal* program running on the host B which was registering the packets.

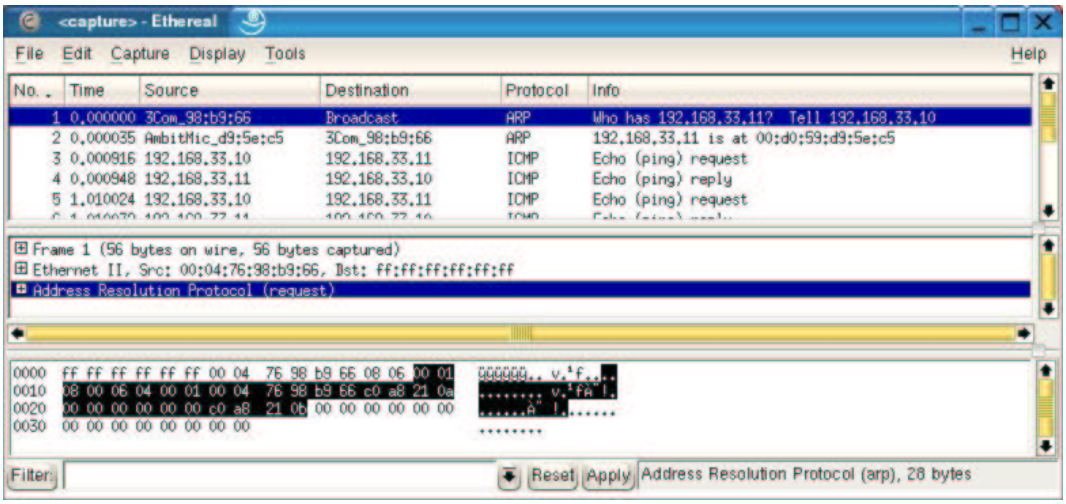

Figure 5.9: Trace of a VLAN connection

Before the first ping packet is sent by the host A, the hardware address of the host B must be learned. Thus, the host A first sends an Address Resolution Protocol (ARP) broadcast frame which is transferred through the MPLS domain to the host B (packet nr 1). Then, host B send its IP address in reply to the ARP query (packet nr 2). From now, the host A can send ping packets (packets nr 3, 5, ...) and the host B replies to them (packets nr 4, 6, ...).

5.5.5 AToM and Quality of Service

Transporting of VLAN frames across the MPLS core network may require a QoS treatment better than the standard Best-Effort approach. For this purpose the AToM frames must be distinguished from the rest of the traffic what can be realized by modifying an EXP field in the MPLS header. Since the EXP field contains 3 bits, it gives a possibility to define up to 8 classes of MPLS traffic which can be bound to different QoS policies in the core network.

This concept was tested in the testbed configuration presented in Section 5.5.3. In order to mark frames sent by hosts A or B an appropriate policy should be installed on the *FastEthernet0/0* subinterfaces of the *zam576* and *zam577* routers, see Figure 5.7. In the presented example, these policies mark the packets by setting the MPLS EXP field to 5 (default is 0). Below is the fragment of the *zam576* and *zam577* configurations:

```
class-map match-all exp5
  match any
!
policy-map exp5
  class exp5
   set mpls experimental 5
!
interface FastEthernet0/0
 no ip address
 no ip redirects
 no ip proxy-arp
 no ip mroute-cache
 no keepalive
 duplex auto
 speed auto
 mpls label protocol ldp
 tag-switching ip
 service-policy input exp5
 no cdp enable
!
```

The input policy `exp5` installed on the *FastEthernet0/0* interface will cause that all packets coming to this interface, so all traffic in the virtual network between hosts A and B, will be pushed to the MPLS domain with an MPLS EXP field of 5.

To ensure an appropriate QoS treatment for AToM packets output policies should be defined and installed on all interfaces in the MPLS domain. These policies filter the MPLS traffic according to their EXP field. In the discussed example, packets marked with the EXP field of 5 are scheduled for the priority queue. For details see the following configuration:

```
class-map match-all qos
  match mpls experimental  5
!
policy-map qos
  class qos
    priority 10000
!
interface FastEthernet0/1
 (...)
 service-policy output qos
 (...)
!
```

The example above shows how the `qos` output policy is configured on the *FastEthernet0/1*. However, it should be installed on all internal links in the MPLS domain in a similar way. All packets from the `qos` class are placed in the priority queue that is limited by a serving rate of `10000` *kbps*.

Chapter 6

Traffic Engineering Framework for Backbone Networks

In Chapter 2 relevant QoS parameters as well as corresponding traffic specifications have been introduced. The following chapters provided detailed insights into the enabling components that allow realizing a certain level of QoS. Yet, a management component is still missing, for which we provide algorithms in terms of the PAB project's traffic engineering framework.

The main QoS parameters that are addressed here are packet loss, which corresponds to certain buffer and bandwidth guarantees, reliability, delay, and delay jitter. Bandwidth constraints and availability respective survivability issues are covered by the routing in Section 6.1, whereas the provisioning of delay bounds builds on Network Calculus in Section 6.2. A simulative evaluation is provided in Section 6.3.

6.1 Routing

In on-line systems where requests arrive one-by-one bandwidth guarantees can be provided efficiently applying capacity constrained routing. The survivability of the network can be further increased by pre-computation and pre-establishment of backup paths, usually considering single element of failure scenarios.

6.1.1 Capacity Constrained Routing

Certain capacity constraints can be efficiently integrated into known routing algorithms. This is for example done in case of Constrained Shortest Path First (CSPF) [55] which is an extension of the widely applied Shortest Path First (SPF) algorithm. The addition of capacity constraints is straight forward. The amount of capacity that is available for a certain service on each of the links is configured statically. Further on, the requested bandwidth of each of the accepted requests that are active concurrently is known. Hence, the residual capacity of the individual links can be computed. If a new request is received, all links which have a residual capacity that is smaller than the bandwidth of this request are removed from the network graph. Thus, all links of the remaining graph offer sufficient resources to accommodate the new request. Based on this reduced graph, any routing algorithm can be run and, if a suitable path is found, it fulfills all of the capacity constraints. However, doing so requires that routing algorithms are executed for each request on an individual network graph, instead of once at initialization time, as it applies for the static SPF scheme. In the following, a few known on-line routing algorithms [49, 50, 73, 69, 42, 43] that are a subset of the routing algorithms that are implemented as part of our traffic engineering framework are introduced.

Shortest Path First

The SPF algorithm is the standard routing scheme used in today's Internet. It minimizes a single additive metric that is assigned as a cost term to each link. This metric can simply be the hop-count or the propagation delay. One important drawback of the SPF scheme is that it does not define rules for the case in which several paths for the same source and destination pair have the same cumulated metric. Instead of optimizing a second metric in this case, SPF chooses the path randomly. This unfortunate property is addressed by the subsequent routing schemes. SPF extensions towards Constrained SPF (CSPF) have already been addressed above. The complexity of an SPF implementation based on the Floyd algorithm that finds shortest paths between all nodes is in $\mathcal{O}(v^3)$ for a network graph with v vertices. The paths from one node to all other nodes can be found by Dijkstra's algorithm in $\mathcal{O}(v^2)$, which applies for CSPF on a per-request basis.

Widest Shortest Path First

The Widest Shortest Path First (WSPF) algorithm is very similar to the SPF scheme. The only difference is that a second metric is considered, if the first metric leads to several alternative paths. In this case the path with the maximal minimum capacity is chosen. Note that capacity is not an additive metric. The capacity of a path is determined by the bottleneck link that is the minimum of the capacities of all links on the path. The second metric is tested with just one additional check, which does not change the complexity of the original SPF algorithm. Capacity constraints are accounted for similar to the CSPF algorithm with one extension that is the residual capacity is applied as the second metric.

Shortest Widest Path First

The Shortest Widest Path First (SWPF) algorithm [73] is another variant of the SPF scheme. The first metric is the maximal minimum capacity along the paths. As second metric the propagation delay or hop-count is applied. The target of the SWPF algorithm is to spread the offered traffic load evenly across the network. Since the first metric is not additive, the implementation of SWPF is more complicated compared to SPF or WSPF. In [50], a double application of Dijkstra's algorithm for each source and destination pair is proposed. In the first run widest paths are found. Then, the next steps are made for each source and destination pair independently. All links that offer a lower capacity than the widest path between a certain source and destination are eliminated from the network graph. Among the remaining links Dijkstra's algorithm is applied again to find the shortest path. The complexity of Dijkstra's algorithm to find the widest paths from one source to all destinations is in $\mathcal{O}(v^2)$. However, in the second step, the Dijkstra algorithm has to be applied on network graphs that are individual for each source and destination pair. Thus, it has to be run at most $v \cdot (v-1)$ times, resulting in a complexity of this SWPF implementation of $\mathcal{O}(v^4)$. Capacity constraints are integrated by deleting links from the network graph as done in case of CSPF and by applying the residual capacity as first metric. The complexity to derive a single path from one source to one destination is given by a double application of the Dijkstra algorithm. It is in $\mathcal{O}(2 \cdot v^2) = \mathcal{O}(v^2)$.

Maximally Disjoint Paths

The Maximally Disjoint Paths (MDJ) algorithm [69] differs from the routing schemes described so far in that it finds more than one path for each source and destination pair. It generates a tree from a source node to all destination nodes based on a simplified SWPF algorithm. Then, a pair of maximally disjoint paths is computed by adding common links as a further routing metric. This step is repeated for each possible source. Further details of the algorithm are given in [69] and are omitted here. The MDJ scheme can be efficiently applied in MPLS networks that implement path protection to pre-configure disjoint backup LSPs. Here, the second path that is the disjoint path is used, if the first path does not fulfill the capacity constraint of the current request. The MDJ algorithm is applied statically. It is run only once applying the configured link capacities and

not for each request based on the residual link capacities. In the latter case, the second disjoint path would never be required since the first one would fulfill the capacity constraints anyway. The complexity of MDJ is in $\mathcal{O}(v^2 \cdot e)$, assuming the number of edges e fulfills $e \geq v$.

Minimum Interference Routing

Minimum Interference Routing (MIR) [42, 43] extends the described algorithms by considering possible future requests. Doing so is of particular interest for immediate reservation systems, where already active traffic trunks cannot be re-routed to accommodate a current request. In contrast, advance reservation systems can implement a planning component, which potentially resolves future conflicts. However, also for immediate reservation systems some vague knowledge about future requests can be used. Here, edge routers are applied as potential sources or destinations. The MIR algorithm is briefly described below: For each source and destination pair the maximum flow problem is solved applying the residual capacity of the links. Then, the interference of a new request on a certain path is defined to be the reduction of the maximum flow value derived for any source destination pair. The objective is to find a path for the current request that maximizes a weighted sum of all of the maximum flow values. Since this problem is NP-hard [42], a heuristic is applied. The minimum cut problem, which is the dual problem to the maximum flow problem is solved and all links that belong to any minimum cut are marked to be critical. Then, modified weights are assigned to the critical links, depending on the reciprocal value of the maximum flow values assigned to these links and depending on the importance that is assigned to the relevant source and destination pairs. Non-critical links are assigned the weight zero. Resulting the CSPF algorithm is executed applying the modified weights as a metric. The complexity of the MIR algorithm is dominated by the maximum flow problem that can be solved for a source and destination pair by the Goldberg Tarjan algorithm in $\mathcal{O}(v^2 \cdot \sqrt{e})$. In addition determining the critical links is in $\mathcal{O}(e^2)$. The maximum flow respective minimum cut problem has to be solved at most $v \cdot (v-1)$ times, resulting in a complexity of $\mathcal{O}(v^4 \cdot \sqrt{e} + v^2 \cdot e^2)$ for the derivation of a single path.

6.1.2 Protection Algorithms

The protection capabilities that are offered by MPLS have already been discussed in Section 4.3, where a classification into link, node, and path protection was introduced. Section 5.4 provided results from experiments on fast re-routing, which applies, if errors can be detected and repaired locally. This local repair comprises of link and node protection and can be distinguished from so-called global repair, where signalling of failure events and path protection are used.

The choice between global and local repair mainly concerns restoration times and bandwidth efficiency. The issue of restoration speed is for example analyzed in [33]. Clearly local repair is much faster, since efficient techniques for error detection can be combined with an immediate local repair using pre-established backup tunnels. Global repair in addition requires a signalling procedure that allows propagating failure events upstream to the point where the primary path and the pre-configured backup path split before the backup path can be used.

However, the pre-configuration of backup paths comes at a prize, that is resources have to be reserved in advance, both for the working paths and for the backup paths. While global repair requires only one backup path, local repair builds on a large number of usually shorter backup paths. Thus, efficient schemes that minimize the required backup bandwidth are required. For this purpose, usually single point of failure scenarios are addressed. The preferred options are $1+1$, $1:1$, and $1:n$ protection. In case of $1+1$ protection the traffic is duplicated and both the working and the backup path transport identical copies of the traffic simultaneously. The downstream node where both paths merge then chooses which copy to use. Thus, each working path requires one exclusive backup path. The same applies for $1:1$ protection, however, traffic is not duplicated, but switched to the backup path only in case of failure of the primary path. Thus, resources that are allocated for backup paths can still be used, for example by Best-Effort traffic. A more resource efficient scheme is $1:n$ protection, where n working paths share one backup path.

Bandwidth sharing can be used inter-demand or in case of local repair also intra-demand [44, 48]. It usually applies among backup paths, whereby more sophisticated schemes also allow to exploit bandwidth sharing between primary and backup paths [52].

¿From a routing point of view $1+1$ and $1:1$ can be implemented straight forward using the remove and find approach that also applies for capacity constrained routing. All links that do not provide a sufficient amount of bandwidth are removed from the network graph before a feasible working path is found. Then, the network elements that shall be protected are also removed from the graph and in a second step backup routes are computed. This approach finds the best disjoint backup path according to a certain metric, if one exists. More flexibility is, however, provided by Maximally Disjoint Paths [69], which allows finding backup paths, even if no completely disjoint paths exist. Further on, algorithms have been developed that allow for a joint optimization of primary and backup paths [35]. Bandwidth sharing schemes like $1:n$ protection require more sophisticated algorithms, where the additional bandwidth that has to be allocated for a primary and backup path pair can be efficiently used as a joint routing metric [44].

6.2 Network Calculus

The reservation-based approach of the Integrated Services architecture relies on a analytical admission control model that is called Network Calculus. Network Calculus is a theory of deterministic queuing systems that is based on the early work on the calculus for network delay in [18, 19], and on the work on Generalized Processor Sharing (GPS) presented in [57, 58]. Further extensions, and a comprehensive overview on current Network Calculus are given in [46, 47], and from the perspective of filtering theory in [13].

The PAB-project selected Network Calculus as the framework for deriving delay-bounds for a Premium Service. In extending the Network Calculus principles used by the Integrated Services architecture towards aggregate based scheduling, we show how these procedures can provide reasonable tight delay-bounds [27].

The basis of Network Calculus is a min-plus calculus where addition is replaced by computation of the minimum and multiplication becomes addition. A comprehensive overview on min-plus algebra can be found in [47]. In the sequel, min-plus convolution and min-plus de-convolution are of particular interest:

Theorem 1 (Min-plus Convolution) *In (6.1) the min-plus convolution denoted by $\otimes$ is given.*

$$(f \otimes g)(t) = \inf_{t \geq s \geq 0} [f(s) + g(t-s)] \tag{6.1}$$

Theorem 2 (Min-plus De-convolution) *The min-plus de-convolution is denoted by $\oslash$. It is given in 6.2.*

$$(f \oslash g)(t) = \sup_{s \geq 0} [f(t+s) - g(s)] \tag{6.2}$$

6.2.1 Arrival Curves

Network Calculus applies a description of data flows by means of cumulative functions $F(t)$ that are called arrival functions. By definition $F(0)$ is zero and $F(t)$ is increasing that is $F(t) \geq F(s)$ for all $t \geq s$. An upper bound on the arrival function can be defined, that is called arrival curve $\alpha(t)$, according to (6.3).

Definition 5 (Arrival Curve) *Consider an increasing function $\alpha(t)$ for $t \geq 0$. A flow $F(t)$ is constrained by $\alpha(t-s)$ if (6.3) holds for all $t \geq s \geq 0$.*

$$F(t) - F(s) \leq \alpha(t-s) \tag{6.3}$$

Based on min-plus convolution (6.4) can be found to be equivalent to (6.3).

$$F(t) \leq (F \otimes \alpha)(t) = \inf_{t \geq s \geq 0} [F(s) + \alpha(t-s)] \tag{6.4}$$

A typical arrival curve is given in the following definition:

Definition 6 (Leaky Bucket Constraint) *The leaky bucket constrained curve $\alpha_{r,b}$ is given in (6.5) for $t > 0$. It is zero otherwise.*

$$\alpha_{r,b}(t) = b + r \cdot t \tag{6.5}$$

An example of an arrival function and an arrival curve is shown in figure 6.1. A simple leaky bucket constraint is applied to give an arrival curve for the particular arrival function that is displayed. The arrival curve gives an upper bound for any interval $t - s \geq 0$ as indicated by the dashed lines in figure 6.1.

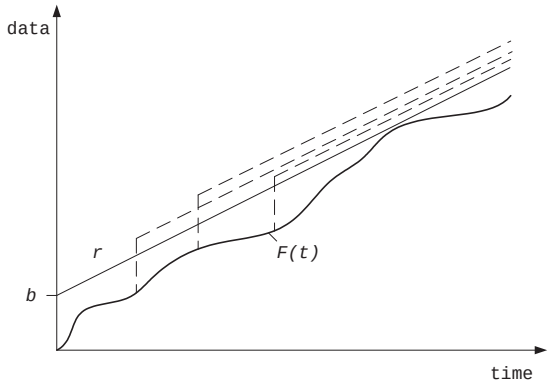

Figure 6.1: Leaky Bucket Arrival Curve

Usually the ingress router of a DS domain meters incoming flows against a leaky bucket algorithm and either shapes, re-marks, or drops non-conforming traffic.

6.2.2 Service Curves

In [57, 58, 20] an important concept for the derivation of service guarantees called minimum service curve is introduced. An extension of this work is given by the following definition [46] that is applied throughout this work.

Definition 7 (Minimum Service Curve) *Consider a flow $F(t)$ that is input to a system with the respective output flow $F^*(t)$. The system offers to the flow a minimum service curve $\beta(t)$, if a time instance s exists for all t with $t \geq s \geq 0$ such that (6.6) holds.*

$$F^*(t) - F(s) \geq \beta(t-s) \tag{6.6}$$

From (6.6) the form in (6.7) follows.

$$F^*(t) \geq (F \otimes \beta)(t) = \inf_{t \geq s \geq 0} [F(s) + \beta(t-s)] \tag{6.7}$$

A special characteristic of service curve is given in the following definition:

Definition 8 (Rate-Latency Function) *The rate-latency function is given by (6.8), where* $[t-T]^+ = 0$ *if* $t < T$*. It describes a service element that offers a minimum service of* R*, but that reacts with a worst case latency of* T*.*

$$\beta_{R,T}(t) = R \cdot [t-T]^+ \tag{6.8}$$

An example of the lower bound for the output of a rate-latency service curve is given in Figure 6.2 for a given input arrival function.

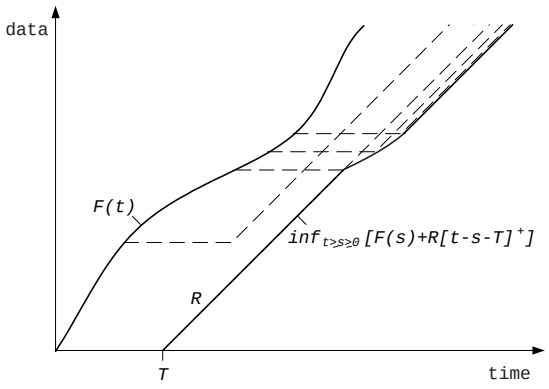

Figure 6.2: Rate-Latency Service Curve

The rate-latency property applies for a variety of actual scheduler implementations, including Priority Queuing (PQ) and Weighted Fair Queuing (WFQ).

6.2.3 Basic Network Calculus

This subsection covers some important concepts on the derivation of backlog, delay, and output bounds at first for the single node case and then for the concatenation of nodes. Further on, aggregate scheduling and rules for multiplexing are addressed. The respective proofs can for example be found in [13, 47].

Bounds on the Backlog, Delay, and Output Flow

Bounds on the backlog, the delay, and on the output flow $F^*(t)$ for an $\alpha(t)$-upper constrained flow $F(t)$ that is input to a system that offers a minimum service curve $\beta(t)$ are provided by the following theorems.

Theorem 3 (Backlog Bound) *Assume a flow* $F(t)$ *that is upper constrained by* $\alpha(t)$ *is input to a system that offers a minimum service curve* $\beta(t)$*. The output flow is denoted by* $F^*(t)$*. The backlog* $F(t) - F^*(t)$ *that is the amount of data that arrived up to a time instance* t *but that has not left the system by* t *is bound according to (6.9). The backlog bound is the supremum of the vertical deviation of the arrival curve and the service curve.*

$$F(t) - F^*(t) \leq \sup_{s \geq 0}[\alpha(s) - \beta(s)] \tag{6.9}$$

For the special case of service curves of the rate-latency type $\beta_{R,T}(t)$ and leaky bucket constrained arrival curves $\alpha_{r,b}(t)$, (6.9) simplifies to (6.10).

$$F(t) - F^*(t) \leq b + r \cdot T \tag{6.10}$$

Theorem 4 (Delay Bound) *The maximal virtual delay d for a FIFO system that offers a service curve of $\beta(t)$ with an input flow that is constrained by $\alpha(t)$, is given as the supremum of the horizontal deviation of the arrival curve and the service curve according to (6.11).*

$$d \leq \sup_{s \geq 0} \big[\inf[\tau \geq 0 : \alpha(s) \leq \beta(s+\tau)]\big] \tag{6.11}$$

For rate-latency service curves and leaky bucket constrained arrival curves (6.12) is given.

$$d \leq T + b/R \tag{6.12}$$

Theorem 5 (Output Flow Constraint) *Assume an input flow $F(t)$ that is upper constrained by $\alpha(t)$ traverses a system that offers a service curve $\beta(t)$. The output flow $F^*(t)$ is upper constrained by an arrival curve $\alpha^*(t)$ that is given according to (6.13).*

$$\alpha^*(t) = (\alpha \oslash \beta)(t) = \sup_{s \geq 0}[\alpha(t+s) - \beta(s)] \tag{6.13}$$

In case of rate-latency service curves and leaky bucket constrained arrival curves (6.14) can be found.

$$\alpha^*(t) = b + r \cdot T + r \cdot t \tag{6.14}$$

An example of the graphical representation of the bounds is shown for rate-latency service curves and leaky bucket arrival curves in figure 6.3.

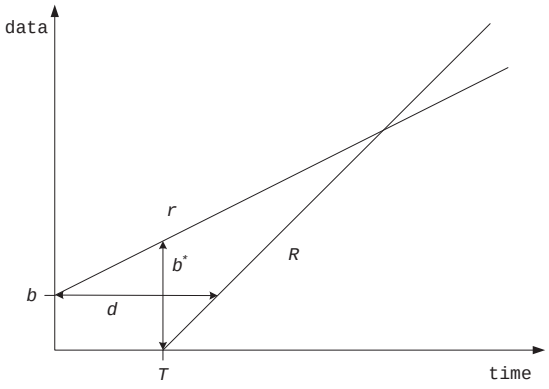

Figure 6.3: Backlog and Delay Bound

The Multiple Node Case

The bounds on the backlog, delay, and on the output shown in theorems 3, 4, and 5 can be applied directly to networks that consist of several nodes. In the following, networks are assumed to consist of n such queuing and scheduling systems that are indicted by upper indices j. In the simplest scenario a flow traverses two systems j and $j+1$ in sequence, whereby the results immediately extend to larger scenarios.

Assume that the input flow $F^j(t)$ to system j is $\alpha^j(t)$-upper constrained. The output flow $F^{j+1}(t)$ of system j is $\alpha^{j+1}(t)$-upper constrained according to theorem 5. The flow $F^{j+1}(t)$ is then input to system $j+1$. Based on theorem 3 backlog bounds for the individual systems can be derived. Similarly, theorem 4 allows to derive delay bounds at each of the systems that can be summed up to worst case end-to-end delays. An example of the described incremental derivation

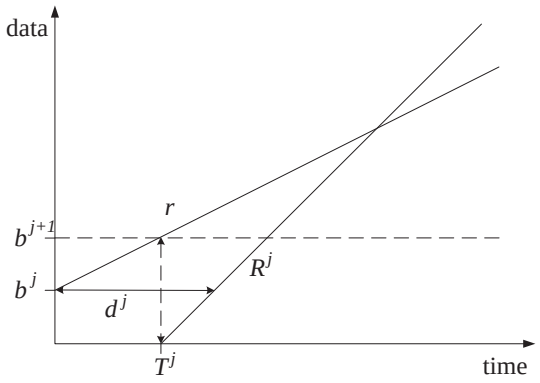

Figure 6.4: Delay Bound Node j

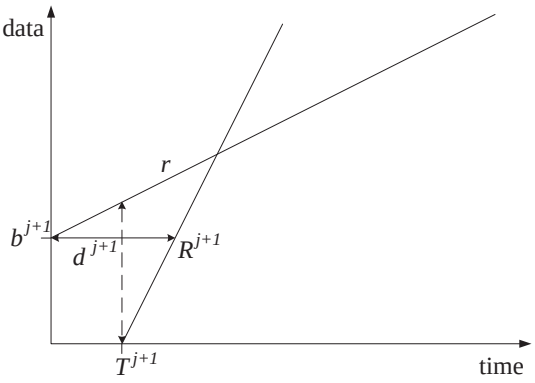

Figure 6.5: Delay Bound Node $j+1$

of end-to-end delays is shown in Figures 6.4 and 6.5 for leaky bucket constrained arrival curves and rate-latency service curves. The end-to-end delay bound d is given in (6.15).

$$d = d^j + d^{j+1} = T^j + \frac{b^j}{R^j} + T^{j+1} + \frac{b^{j+1}}{R^{j+1}} = T^j + T^{j+1} + \frac{b^j}{R^j} + \frac{b^j + r \cdot T^j}{R^{j+1}} \tag{6.15}$$

However, a property known as Pay Bursts Only Once [47] allows to give a tighter end-to-end delay bound, if the delay computation is based on the end-to-end service curve, instead of summing the delays at individual systems up.

Theorem 6 (Concatenation) *The end-to-end service curve of two systems j and $j+1$ that each offer a service curve β^j respective β^{j+1} is given in (6.16). The direct application of (6.16) also holds for n links.*

$$\beta^{j,j+1}(t) = \inf_{t \geq s \geq 0} [\beta^j(s) + \beta^{j+1}(t-s)] = (\beta^j \otimes \beta^{j+1})(t) \tag{6.16}$$

The concatenation of two rate-latency service curves according to (6.16) can be simplified to (6.17).

$$\beta^{j,j+1}(t) = \min[R^j, R^{j+1}] \cdot [t - T^j - T^{j+1}]^+ \tag{6.17}$$

The same example shown in Figures 6.4 and 6.5, but this time based on the concatenation theorem 6, is given in figures 6.6 and 6.7. Assume that the input flow $F^j(t)$ to link j is leaky bucket constrained. The service curves of system j and $j+1$ are again of the rate-latency type. At first, figure 6.6 shows the derivation of the end-to-end service curve. Based on this service curve the end-to-end delay is shown in Figure 6.7 and given by (6.18). As can be seen from (6.15) and (6.18) the delay bound is smaller, if the end-to-end service curve is derived prior to delay computations. Obviously the burst of the flow $F^j(t)$ is paid only once with a latency that is determined by the bottleneck link capacity in this case.

$$d = T^j + T^{j+1} + \frac{b^j}{\min[R^j, R^{j+1}]} \tag{6.18}$$

Aggregate Scheduling

In order to address aggregate based networks, rules for multiplexing and aggregate scheduling have to be added. Assume that m flows are scheduled by a network in an aggregate manner. Let lower indices i denote flows in the sequel.

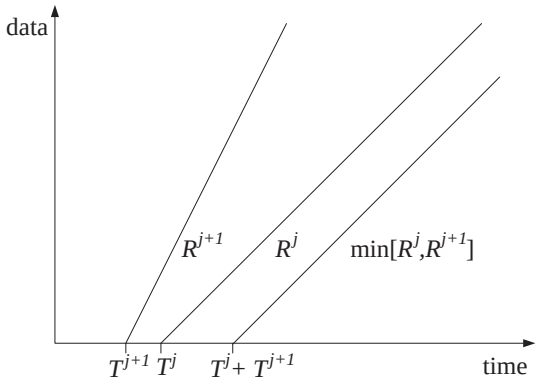

Figure 6.6: Concatenation of Nodes

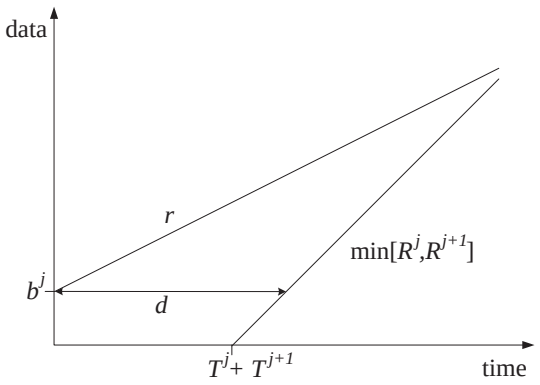

Figure 6.7: End-to-end Delay Bound

Theorem 7 (Multiplexing) *An aggregation, or multiplexing of flows can be expressed by addition of the arrival functions according to (6.19) respective arrival curves according to (6.20).*

$$F_{1,2}(t) = F_1(t) + F_2(t) \tag{6.19}$$
$$\alpha_{1,2}(t) = \alpha_1(t) + \alpha_2(t) \tag{6.20}$$

For leaky bucket constrained arrival curves $\alpha_1 = b_1 + r_1 \cdot t$ and $\alpha_2 = b_2 + r_2 \cdot t$ the form in (6.20) simplifies to $\alpha_{1,2}(t) = (b_1 + b_2) + (r_1 + r_2) \cdot t$.

Theorem 8 (FIFO Aggregate Scheduling) *In aggregate scheduling networks with FIFO service elements, families of per-flow service curves $\beta_{i,\theta}(t)$ according to (6.21), with an arbitrary parameter $\theta \geq 0$ are derived in [20, 47]. $\beta_{i,\theta}(t)$ gives a family of service curves for a flow i that is scheduled in an aggregate manner with a flow, or a traffic trunk $i+1$ by a system that offers a service curve $\beta(t)$ to the aggregate of trunks i and $i+1$. The term $1_{t>\theta}$ is zero for $t \leq \theta$.*

$$\beta_{i,\theta}(t) = [\beta(t) - \alpha_{i+1}(t-\theta)]^+ 1_{t>\theta} \tag{6.21}$$

The parameter θ has to be set to zero in case of blind multiplexing, which applies if no assumption about the order of processing, for example FIFO, can be made.

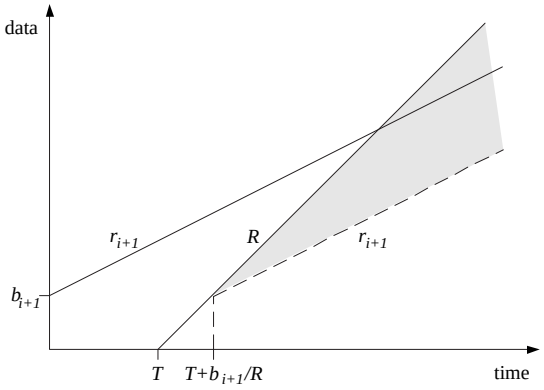

Figure 6.8: Flow i Service Curve

For the special case of rate-latency service curves and leaky bucket constrained flows, the service curve for flow i in (6.22) can be derived from theorem 8 for $r_i + r_{i+1} < R$. The optimal setting of θ with respect to backlog bounds is $\theta = T + b_{i+1}/R$ [47]. The example is shown in figure 6.8, where the gray shaded area denotes the service that remains for flow i in the worst-case.

$$\beta_i(t) = (R - r_{i+1}) \cdot \left[t - T - \frac{b_{i+1}}{R}\right]^+ \tag{6.22}$$

6.2.4 Feed-Forward Networks

Unfortunately, the direct application of Network Calculus has one important prerequisite, namely the network has to be of a feed-forward nature.

Definition 9 (Feed-Forward Property) *A feed-forward queuing network is a network, in which all queues can be ordered in such a way that whenever a traffic flow traverses from queue i to queue j, this implies that $i < j$ [37], or in a more verbatim way: the links of a feed-forward network cannot form any cycles, i.e. it is impossible for traffic flows to create cyclic dependencies on each other [13].*

An example of a non feed-forward network is shown in the left of figure 6.9. The four traffic trunks that are mapped onto the small five node network form a cycle. The service that is offered by each link to each of the trunks depends on the interference introduced by trunks that traverse the same link. However, the traffic specification of interfering trunks is subject to the service offered to these trunks at the predecessor links, resulting in a cyclic dependency. Analytical methods which address such scenarios like time-stopping have been discussed [13, 47]. However, these methods are considered to be too complicated for an algorithmic implementation, here. An alternative is to apply algorithms that transform the topology of an arbitrary network into a feed-forward network. Mainly two approaches are known in current literature [66, 68], which either prohibit the use of certain links or the use of certain turns. A link can be specified by the tuple consisting of source and destination node for example (a, b) or (b, c), whereas a turn always consists of two successive links. The turn around node b can for example be given by (a, b, c). Prohibiting this turn means that the triple (a, b, c) is not allowed being part of any path, whereas the links (a, b) or (b, c) may still be used. Intuitively, the prohibition of turns is less restrictive than prohibiting the use of links.

Algorithms that ensure the feed-forward property have to be executed before routing algorithms are run to ensure that the derived mapping of trunks on the network results in cycle-free aggregates. We devolved the Turnnet concept, which allows integrating the different feed-forward schemes efficiently into known routing algorithms [26].

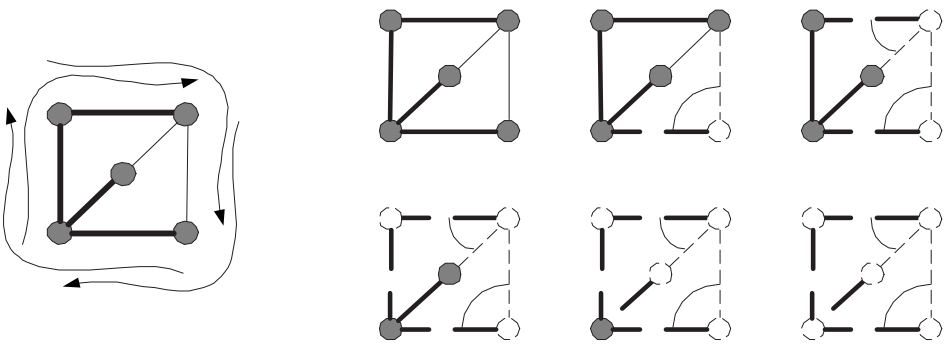

Figure 6.9: Non Feed-Forward Network and Turn Prohibition Example

Spanning Tree

A means to ensure the feed-forward property of an aggregate is to forbid all links that could create a cycle within the topology. The Spanning Tree (ST) of a given network topology fulfills this characteristic. All links that do not belong to the spanning tree are forbidden and routing can only be performed along the tree. The connectivity is not affected for networks that consist of bi-directional links. Several ways to construct Spanning Trees exist. Among these, the Kruskal algorithm is implemented in our admission control. It starts with an empty link list and adds links that do not create a cycle to this list until the spanning tree is complete. The links can be ordered in advance according to some metric like the capacity to ensure that for example wider links are tried first. The complexity is dominated by the sorting of links. It is in $\mathcal{O}(e \cdot \log e)$ for e edges.

A major drawback of applying Spanning Trees to ensure the feed-forward property is that forbidden links are completely unused whereas links that are close to the root of the tree are likely to become congested. Further on, the resulting paths can be significantly longer than the ones that are for example found by SPF routing.

Root Spanning Tree

A method which partly overcomes the problems of Spanning Trees is to apply a Root Spanning Tree (RST). A root is chosen explicitly according to a criterion like capacity or degree that is the number of connected links. Then, links are added in breadth-first order, if they do not introduce a cycle. The resulting paths are likely to be shorter than the ones that are found by the Spanning Tree algorithm. However, there is still a significant risk that links that are close to the root become congested, whereas other links are only lightly loaded or even forbidden.

Up-Down Routing

The Up-Down Routing (UDR) algorithm [66] prevents from cyclic dependencies not by forbidding the use of certain links, but only by prohibiting turns. A turn consists of two unidirectional links, for example the two links (a, b) and (b, c) constitute the turn (a, b, c). Bidirectional links are split up into two unidirectional links in advance.

Up-Down Routing is based on a classification of links to be either Up or Down. This classification is performed based on a Root Spanning Tree. Links are considered as Up, if they are directed towards the root of the Root Spanning Tree that is if the distance of the connected nodes towards the root decreases in the direction of the links. Otherwise links are considered as Down. Note that this classification is done for all links not only for links that belong to the Root Spanning Tree. The Root Spanning Tree is applied only to order the nodes according to their distance to the root from which the direction of the links follows. Based on the notation of Up and Down links, turns can be classified to be either Up-Up, Down-Up, Up-Down, or Down-Down turns. A Down-Up turn for example consists of a first link that is a Down link and a second link that is an Up link. In [66] it is shown that a cycle must contain at least one Up-Down turn and one Down-Up turn. Hence, it is sufficient to prohibit for example all Down-Up turns to prevent from cycles to ensure the feed-forward property.

As pointed out in [68], Up-Down Routing still suffers from its dependency on the Root Spanning Tree. The choice of the root can have a large impact on the performance.

Turn Prohibition

Turn Prohibition (TP) [68] like Up-Down Routing ensures the feed-forward property of the network by prohibiting the use of certain turns, whereas complete links are not removed from the network graph. The algorithm processes all nodes of the network in the order of their degree that is the number of a node's incoming and outgoing links. If several nodes have the same degree the cumulated capacity of the links is applied as a second metric. During each iteration it basically does the following:

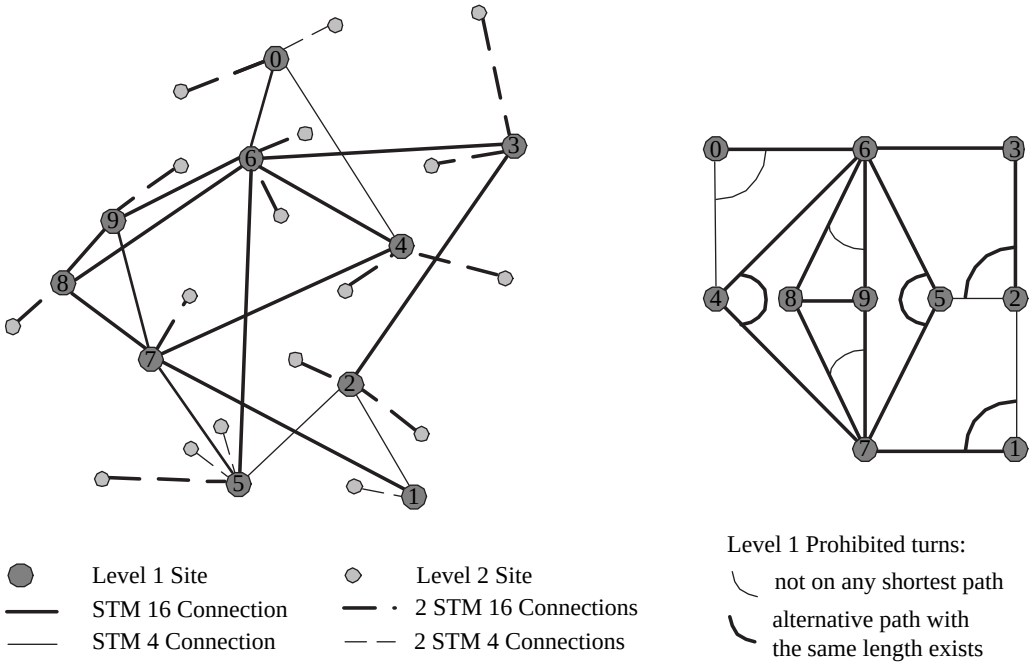

Figure 6.10: DFN G-WiN Topology as of 2000 and Level 1 Prohibited Turns

1. The node with the minimal degree is found.

2. All turns around the node are prohibited.

3. The node and all connected links are deleted from the network graph.

The different steps of the algorithm are shown in figure 6.9. The small example network consists of five nodes that are either connected by low bandwidth links indicated by thin lines or high bandwidth links that are thick lines. It is used by four traffic trunks that are shown in the left of the figure. These trunks form a cycle, which does not allow the direct application of Network Calculus. Now, if Turn Prohibition is applied, at first a node with minimal degree has to be found. The degree is determined by the number of outgoing, respective incoming interfaces and in a second step by the sum of the bandwidth of all these links. The node in the right lower corner is chosen since it has the minimum number of interfaces and among these the smallest cumulative capacity of the links. Another possible choice would have been the node in the center. All turns around the node are prohibited and the node and all connected links are deleted. This process repeats until all nodes have been inspected. At the end, two turns are prohibited to ensure the required feed-forward property.

The actual algorithm that is implemented is more complex, to guarantee that the network remains fully connected that is, it does not become divided into several unconnected subnetworks. In [68], it is proven that the Turn Prohibition algorithm breaks all cycles, preserves global connectivity, and prohibits at most one third of all turns.

Example results of an application of the Turn Prohibition algorithm to the DFN G-WiN topologies as of 2000 [39] and 2002 [1] are shown in Figures 6.10 and 6.11. The nodes have been processed by Turn Prohibition in the order of their numbering. We find that SPF routing with Turn Prohibition does not result in any longer paths compared to simple SPF routing for the G-WiN topology of 2000, whereas in the G-WiN topology of 2002 the turns $(2, 1, 9)$ and $(9, 1, 2)$

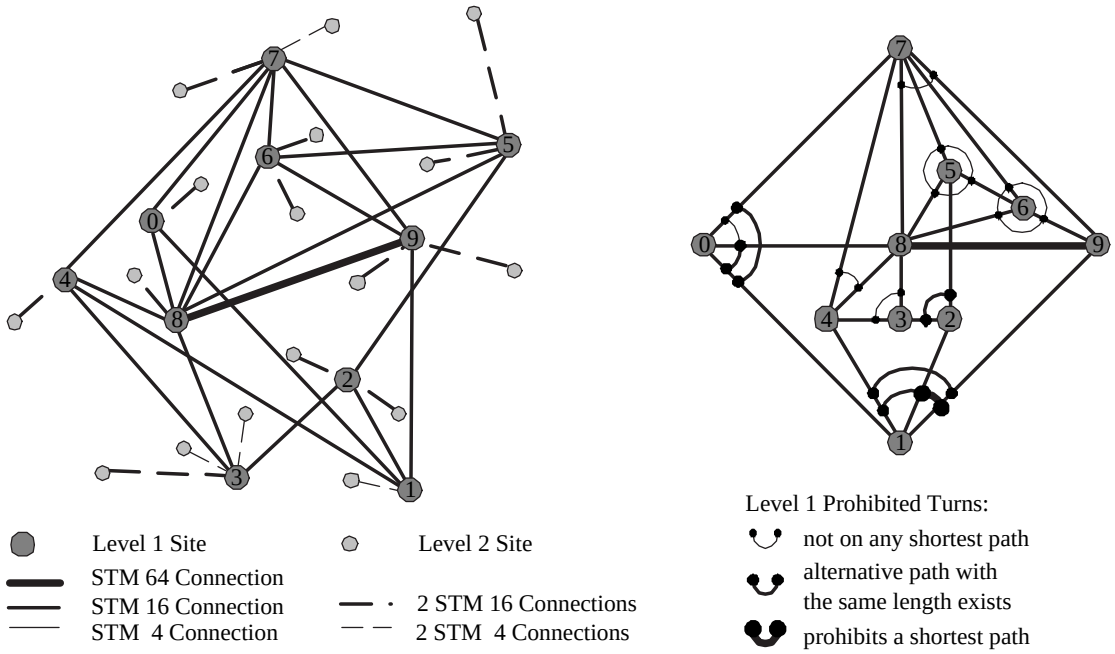

Figure 6.11: DFN G-WiN Topology as of 2002 and Level 1 Prohibited Turns

and thus the shortest path between nodes 2 and 9 is prohibited. However, this is the only path, which has to be replaced by a longer one, due to Turn Prohibition.

6.2.5 Advanced Network Calculus for Aggregate Scheduling Networks

The Pay Bursts Only Once phenomenon is accounted for when the end-to-end service curve of all links on the path of a flow is computed. While the reservation-based approach of the Integrated Services architecture relies on this phenomenon, its applicability for aggregate scheduling, particularly in the context of admission control procedures of a Bandwidth Broker, is still subject to research. We now show how the Pay Bursts Only Once phenomenon can be extended to aggregate scheduling networks [27]. We motivate this step by investigation of the scenario in figure 6.12, where two flows 1 and 2 are scheduled in FIFO order and in an aggregate manner at two consecutive links I and II.

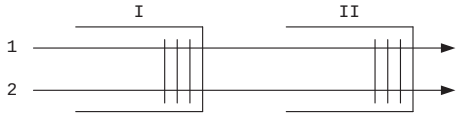

Figure 6.12: Two Flows Share Two Consecutive Links

We derive the end-to-end service curve offered to flow 1 in this scenario. At first, the arrival curve of flow 2 as it is input to link I respective link II can be computed. It can then be subtracted independently from the service curves of links I and II according to theorem 8 and the resulting end-to-end service curve for flow 1 can be obtained from concatenation of the two modified service

curves as shown in (6.23), with the arbitrary parameters $\theta \geq 0$ and $\vartheta \geq 0$.

$$\begin{aligned}\beta^{\mathrm{I,II}}_{1,\theta,\vartheta}(t) &= (\beta^{\mathrm{I}}_{1,\theta} \otimes \beta^{\mathrm{II}}_{1,\vartheta})(t) \\ &= [(\beta^{\mathrm{I}}(t) - \alpha^{\mathrm{I}}_2(t-\theta))]^+ 1_{t>\theta} \otimes [(\beta^{\mathrm{II}}(t) - \alpha^{\mathrm{II}}_2(t-\vartheta))]^+ 1_{t>\vartheta}\end{aligned} \tag{6.23}$$

For rate-latency service curves $\beta^{\mathrm{I}} = R^{\mathrm{I}} \cdot [t - T^{\mathrm{I}}]^+$ and $\beta^{\mathrm{II}} = R^{\mathrm{II}} \cdot [t - T^{\mathrm{II}}]^+$ and leaky bucket constrained arrival curves $\alpha^{\mathrm{I}}_2(t) = b^{\mathrm{I}}_2 + r_2 \cdot t$ and $\alpha^{\mathrm{II}}_2(t) = b^{\mathrm{II}}_2 + r_2 \cdot t$ equation (6.24) can be derived. The actual parameters θ and ϑ applied are $\theta = T^{\mathrm{I}} + b^{\mathrm{I}}_2/R^{\mathrm{I}}$ and $\vartheta = T^{\mathrm{II}} + b^{\mathrm{II}}_2/R^{\mathrm{II}}$ in accordance with the derivation of (6.22).

$$\begin{aligned}\beta^{\mathrm{I,II}}_1(t) &= \left((R^{\mathrm{I}} - r_2) \cdot \left[t - T^{\mathrm{I}} - \frac{b^{\mathrm{I}}_2}{R^{\mathrm{I}}}\right]^+\right) \otimes \left((R^{\mathrm{II}} - r_2) \cdot \left[t - T^{\mathrm{II}} - \frac{b^{\mathrm{II}}_2}{R^{\mathrm{II}}}\right]^+\right) \\ &= \min[R^{\mathrm{I}} - r_2, R^{\mathrm{II}} - r_2] \cdot \left[t - T^{\mathrm{I}} - T^{\mathrm{II}} - \frac{b^{\mathrm{I}}_2}{R^{\mathrm{I}}} - \frac{b^{\mathrm{II}}_2}{R^{\mathrm{II}}}\right]^+\end{aligned} \tag{6.24}$$

Doing so does unfortunately lead to unnecessarily loose bounds, although the Pay Bursts Only Once phenomenon is taken into account. Bursts of flow 2 can occur and interfere at link I as well as at link II. However, both links are traversed by both flows in FIFO order and flow 2 is aggregated with flow 1 only once namely at link I. Thus, multiplexing effects can occur only once. A closer bound can be obtained, if at first the end-to-end service curve for the aggregate of flows 1 and 2 is derived by min-plus convolution of the service curves of links I and II, before the arrival curve of flow 2 is subtracted as shown in (6.25), where $\kappa \geq 0$.

$$\beta^{\mathrm{I,II}}_{1,\kappa}(t) = [(\beta^{\mathrm{I}} \otimes \beta^{\mathrm{II}})(t) - \alpha^{\mathrm{I}}_2(t-\kappa)]^+ 1_{t>\kappa} \tag{6.25}$$

For rate-latency service curves and leaky bucket constrained arrival curves (6.26) follows with $\kappa = T^{\mathrm{I}} + T^{\mathrm{II}} + b^{\mathrm{I}}_2 / \min[R^{\mathrm{I}}, R^{\mathrm{II}}]$.

$$\begin{aligned}\beta^{\mathrm{I,II}}_1(t) &= (R^{\mathrm{I}} \cdot [t - T^{\mathrm{I}}]^+) \otimes (R^{\mathrm{II}} \cdot [t - T^{\mathrm{II}}]^+) - b^{\mathrm{I}}_2 - r_2 \cdot \left(t - T^{\mathrm{I}} - T^{\mathrm{II}} - \frac{b^{\mathrm{I}}_2}{\min[R^{\mathrm{I}}, R^{\mathrm{II}}]}\right) \\ &= (\min[R^{\mathrm{I}}, R^{\mathrm{II}}] - r_2) \cdot \left[t - T^{\mathrm{I}} - T^{\mathrm{II}} - \frac{b^{\mathrm{I}}_2}{\min[R^{\mathrm{I}}, R^{\mathrm{II}}]}\right]^+\end{aligned} \tag{6.26}$$

It can be immediately seen that the burst size of flow 2 applies only once in (6.26) with b^{I}_2 instead of twice in (6.24) with b^{I}_2, and b^{II}_2, where $b^{\mathrm{I}}_2 \leq b^{\mathrm{II}}_2 = b^{\mathrm{I}}_2 + r_2 \cdot (T + b^{\mathrm{I}}_1/R^{\mathrm{I}})$ according to (6.14) and (6.22).

Here, the concept is extended inductively to cover larger scenarios. Consider a flow of interest i that shares a certain sub-path $\mathbb{J}_{i,k}$ with an interfering flow k, as for example shown in Figure 6.12. Then, add another interfering flow $k+1$ to the setting that shares the sub-path $\mathbb{J}_{i,k+1}$ with flow i. The following four different scenarios can be constructed:

Case 1 ($\mathbb{J}_{i,k} \cap \mathbb{J}_{i,k+1} = \emptyset$) If the two sub-paths $\mathbb{J}_{i,k}$ and $\mathbb{J}_{i,k+1}$ do not share any common links, the service curves for flow i can be derived independently for each of the two sub-paths as shown in (6.25) and (6.26) and concatenated according to (6.16) afterwards.

Case 2 ($\mathbb{J}_{i,k} = \mathbb{J}_{i,k+1}$) If the two sub-paths $\mathbb{J}_{i,k}$ and $\mathbb{J}_{i,k+1}$ are identical, the arrival curves of flows k and $k+1$ can be multiplexed to a single interfering traffic trunk according to (6.20) for which (6.25) respective (6.26) can be applied immediately.

Case 3 ($\mathbb{J}_{i,k} \subset \mathbb{J}_{i,k+1} \vee \mathbb{J}_{i,k+1} \subset \mathbb{J}_{i,k}$) If one of the two sub-paths $\mathbb{J}_{i,k}$ or $\mathbb{J}_{i,k+1}$ is completely included within the other one, the service curve for flow i can be derived starting at the innermost sub-path. An example scenario is shown in figure 6.13. Following the argumentation above, (6.27) can be set up.

$$\beta^{\mathrm{I,II,III}}_{1,\theta,\vartheta}(t) = \beta^{\mathrm{I}}(t) \otimes [(\beta^{\mathrm{II}}(t) - \alpha^{\mathrm{II}}_3(t-\theta))]^+ 1_{t>\theta} \otimes [\beta^{\mathrm{III}}(t) - \alpha^{\mathrm{I}}_2(t-\vartheta)]^+ 1_{t>\vartheta} \tag{6.27}$$

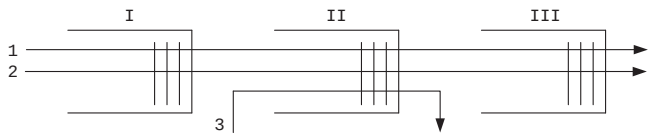

Figure 6.13: Example Network with Nested Interfering Flows

For rate-latency service curves and leaky bucket constrained arrival curves and with $\theta = T^{\mathrm{II}} + b_3^{\mathrm{II}}/R^{\mathrm{II}}$ and $\vartheta = T^{\mathrm{I}}+T^{\mathrm{II}}+T^{\mathrm{III}}+b_2^{\mathrm{I}}/\min[R^{\mathrm{I}}, R^{\mathrm{II}}-r_3, R^{\mathrm{III}}]$, equation (6.28) follows similar to (6.26) before.

$$\begin{aligned}\beta_1^{\mathrm{I,II,III}}(t) &= (\min[R^{\mathrm{I}}, R^{\mathrm{II}} - r_3, R^{\mathrm{III}}] - r_2) \\ &\quad \cdot \left[t - T^{\mathrm{I}} - T^{\mathrm{II}} - T^{\mathrm{III}} - \frac{b_3^{\mathrm{II}}}{R^{\mathrm{II}}} - \frac{b_2^{\mathrm{I}}}{\min[R^{\mathrm{I}}, R^{\mathrm{II}} - r_3, R^{\mathrm{III}}]}\right]^+ \end{aligned} \tag{6.28}$$

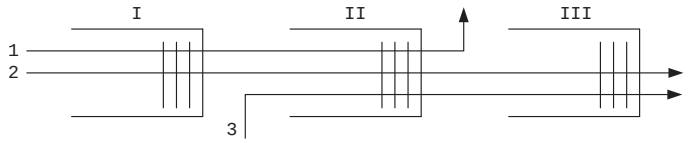

Figure 6.14: Example Network with Overlapping Interfering Flows

Case 4 ($\mathbb{J}_{i,k} \cap \mathbb{J}_{i,k+1} \neq \emptyset \wedge \mathbb{J}_{i,k} \nsubseteq \mathbb{J}_{i,k+1} \wedge \mathbb{J}_{i,k+1} \nsubseteq \mathbb{J}_{i,k}$) If the two sub-paths $\mathbb{J}_{i,k}$ and $\mathbb{J}_{i,k+1}$ overlap each other but none is completely included within the other one, the problem of deriving a flow i service curve, where bursts of flows k and $k+1$ are paid only once, is more complicated. An example setting is shown in figure 6.14, where flow 2 is the flow of interest for which the end-to-end service curve is given in (6.29), where b_3^{III} is the burst size of flow 3 as it is input to link III.

$$\begin{aligned}\beta_2^{\mathrm{I,II,III}}(t) &= \min[R^{\mathrm{I}} - r_1, R^{\mathrm{II}} - r_3 - r_1, R^{\mathrm{III}} - r_3] \\ &\quad \cdot \left[t - T^{\mathrm{I}} - T^{\mathrm{II}} - T^{\mathrm{III}} - \frac{b_1^{\mathrm{I}}}{\min[R^{\mathrm{I}}, R^{\mathrm{II}} - r_3]} - \frac{b_3^{\mathrm{III}}}{\min[R^{\mathrm{II}} - r_1, R^{\mathrm{III}}]}\right]^+ \end{aligned} \tag{6.29}$$

The alert reader will have noticed that the above listed scenarios can be generalized to cover all possible changes in the composition of an aggregate. First, the existence of more or less links than in the scenarios presented does not change the above derivations. Whenever there are no multiplexing or de-multiplexing points in between consecutive links, these can be concatenated in advance as shown in (6.26). Further on, the assumption of a leaky-bucket constrained arrival curve also applies for an aggregate of leaky-bucket constrained arrival curves according to (6.20), i.e for traffic trunks. The derived formulas therefore also apply for more complex scenarios where multiple-flows leave or join the aggregate. Finally, any combinations of the four cases shown is possible to address scenarios with more than three flows. Each further flow increases the interference and thus obviously has impacts on the offered service curve. However, any additional flow can overlap each of the other interfering flows only according to one of the four cases addressed before, so that the same rules as above can be applied.

6.3 Simulation Framework and Evaluation

This section gives evaluation results that are obtained for a Premium Service, based on the DFN G-WiN topology as of 2002 shown in Figure 6.11. The theory discussed in Section 6.2 is applied

for a parameter based admission control. In detail we use a leaky bucket Traffic Specification (TSpec) that consists of a maximum burst size b and a sustainable rate r. The respective Service Specification (RSpec) defines a target of zero loss and a requested delay bound d.

The application of Network Calculus requires that the respective aggregate is cycle-free, that is the feed-forward property has to be fulfilled. Here, we apply Turn Prohibition for this purpose. However, the prize at which the feed-forward property comes is still uncertain to some extend. In Figure 6.11, we have shown that Turn Prohibition increases only one path by one hop in case of the DFN G-WiN topology and SPF routing. Nevertheless, we want to quantify the impacts of Turn Prohibition, when actually providing a certain service. Therefore, we at first relax the RSpec and investigate a Guaranteed Rate Service that does not provide any delay guarantees.

6.3.1 Guaranteed Rate Service

The Guaranteed Rate Service that is investigated here is a service that simply provides bandwidth on demand. Requests for this service are accepted as long as the capacity constraints can be fulfilled for all of the links, otherwise requests are rejected. Evaluation results are derived for SPF, MDJ [69], and CSPF routing, based on the topology of the DFN G-WiN as of 2002 [1]. The level one nodes are applied as core nodes. End-systems are connected to the level two nodes that are edge nodes. In the emulated scenario, a share of 25 % of the available link capacities is reserved statically to provide the Guaranteed Rate Service. The propagation delay is applied as link metric. It is assumed to be 2 ms for each link. The performance measure that we apply is the ratio of accepted requests based on the overall number of request. All simulations are run until the confidence interval of the acceptance ratio respective blocking ratio is smaller than 0.01 for a confidence level of 0.95. Initial-data deletion [45] is applied to capture only the steady-state behavior and the replication and deletion approach for means, that is for example shown in [45] as a variance reduction technique, is used.

Figures 6.15 and 6.16 show the acceptance ratio of simulated Guaranteed Rate requests for a varying load ρ. The load is defined to be the mean number of concurrent requests. Start and end times of requests are modelled as negative exponentially distributed with a mean inter-arrival time $1/\lambda$ and a mean service time $1/\mu$. Resulting a mean of $\rho = \lambda/\mu$ requests are active concurrently. The requested guaranteed rate is a random variable that is chosen uniformly from the interval $[10, \ldots, 80]$ Mb/s, which is assumed to be appropriate for aggregated traffic.

Figure 6.15 addresses the performance of static routing algorithms that do not consider the residual capacity of the links, but only the capacity that is configured. Thus, these algorithms have to be executed only once at startup time and not for each request individually. The MDJ algorithm computes two maximally disjoint paths for each source and destination pair. Thus, it allows to apply an alternative path. Though the MDJ algorithm is executed only once at startup time, capacity constraints can be applied to map requests on the two available paths. The performance gain of the MDJ scheme compared to single path SPF routing is significant.

Results obtained from the respective capacity constrained routing algorithm CSPF are shown in Figure 6.16. CSPF applies a reduced network graph from which all links that do not offer a sufficient residual capacity to accommodate the current request are deleted prior to route computation. Resulting, if a path is found, it consequently fulfills the respective capacity constraint. A significant performance increase compared to the static SPF based scheme can be noticed. However, it comes at the prize of additional computational complexity that is created by an execution of the respective routing algorithm on a per-request basis. Note that though MDJ is a static scheme that is not executed for each request, it performs only slightly worse than CSPF.

Besides, the impacts of Turn Prohibition (TP) on SPF, MDJ, respective CSPF routing are addressed in figures 6.15 and 6.16. There is obviously no benefit from cycle-free Guaranteed Rate aggregates, however, the performance degradation that is due to the feed-forward property can be measured for a Guaranteed Rate Service to evaluate feed-forward routing that is required for the Premium Service in the sequel. We find a certain performance degradation due to Turn Prohibition, which results in a reduction of the acceptance ratio by ~ 0.025 for a load of $\rho \geq 100$ concurrently active requests. However, it has to be noted that the level one mesh of the G-WiN

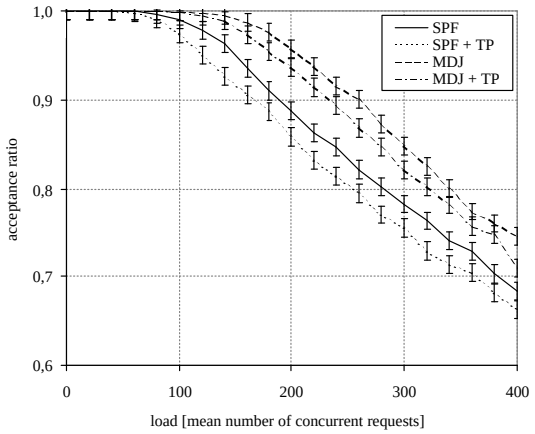

Figure 6.15: Static Routing Schemes and Turn Prohibition, Guaranteed Rate Service

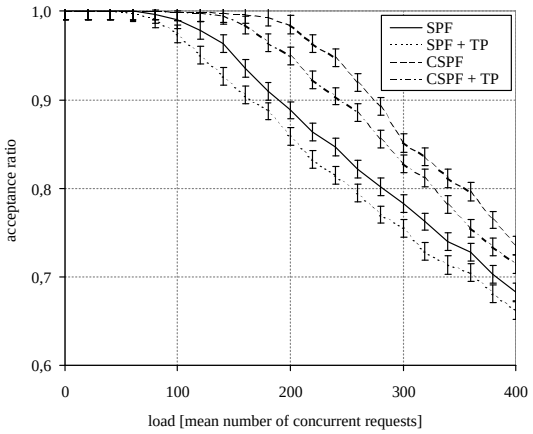

Figure 6.16: Constrained Routing and Turn Prohibition, Guaranteed Rate Service

topology is comparably challenging. In contrast, if redundancy is achieved by means of parallel links in star-shaped topologies, for example in case of the level two nodes of the G-WiN, the feed-forward property is fulfilled anyway.

6.3.2 Premium Service

The Premium Service that is investigated here extends the Guaranteed Rate Service by providing hard delay bounds and zero loss. For this purpose the simple capacity check that is performed by the admission control so far is extended by Network Calculus based backlog and delay computations. Premium Service requests are only accepted, if the capacity and backlog constraints can be fulfilled at each link of the network and, if the requested edge-to-edge delay bound across the respective domain can be ensured for all concurrently active requests. A link propagation delay of 2 ms per link is included into the edge-to-edge delay computation.

The evaluation scenario is similar to the setup for the Guaranteed Rate Service in Section 6.3.1. However, service requests are extended by a burst size parameter that is chosen randomly from the interval $[1, \ldots, 8]$ Mb and a target delay bound within the interval $[40, \ldots, 80]$ ms. Further on, a Priority Queuing (PQ) environment is emulated, where 100% of the link capacities can be used to transmit Premium bursts. PQ is the canonical implementation of the EF PHB to support a Premium Service. The latency of the PQ scheduler is assumed to be equal to the time it takes to transmit 4 Maximum Transmission Units (MTU) of 9.6 kB, to account for non-preemptive scheduling, packetization, and a router internal buffer for up to 2 MTU [63].

Simulation results that compare the option of an incremental delay computation, the application of the well-known Pay Burst Only Once (PBOO) phenomenon, and the Extended Pay Bursts Only Once (EPBOO) principle are shown in Figures 6.17 and 6.18. As before, the G-WiN topology shown in figure 6.11 is used. In addition to the acceptance ratio of Premium Service requests, we apply the Cumulative Density Function (CDF) of the derived delay bounds as a performance measure. To illustrate how tight the derived delay bounds actually are, worst-case per packet delays that are derived by state-event simulation of the respective domain are shown for comparison.

Starting with Figure 6.17, it can be noticed that applying the Extended Pay Bursts Only Once principle for aggregate scheduling allows to derive significantly tighter delay bounds than the common Pay Burst Only Once principle, which in turn outperforms the incremental delay derivation. The gain is small for traffic flows that have a relatively small delay bound, since these flows mainly use short paths with a small number of interfering flows. In contrast, the gain is large

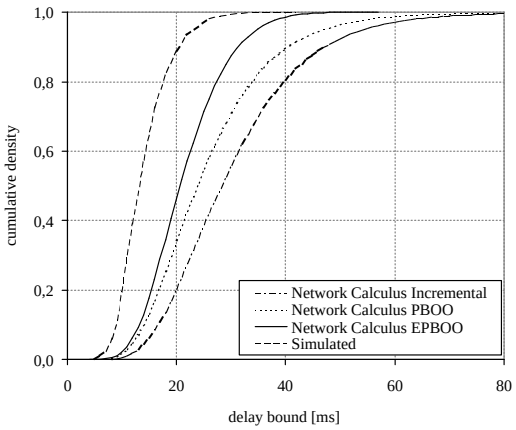

Figure 6.17: Cumulative Density Function of the Worst-Case Delay for a load of $\rho = 50$

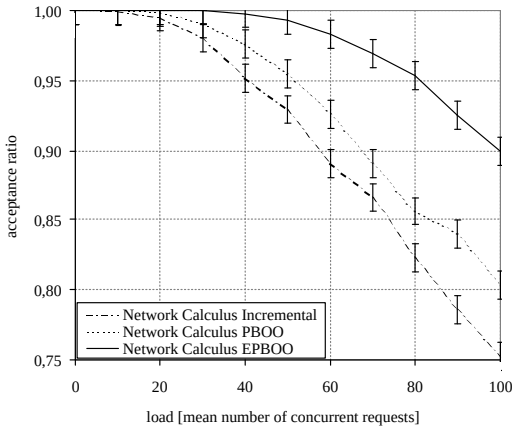

Figure 6.18: Acceptance Ratio for a Premium Service that Provides Bounded Delay

for flows with a large delay bound, that are mainly flows that use a comparably long path and that are disturbed by high interference. Resulting, the Extended Pay Burst Only Once principle is advantageous in two ways: It allows to derive tighter delay bounds and it in particular addresses problematic flows for which the derived bounds are large and that are thus likely to violate delay constraints. Compared to simulated worst-case delays, we find that the EPBOO option allows to derive reasonably tight delay bounds.

Consequently, the call acceptance ratio shown in figure 6.18 is increased so that for example an acceptance ratio of 0.95 can be achieved up to a mean load of nearly 70 instead of 50, respective 40 concurrent requests.

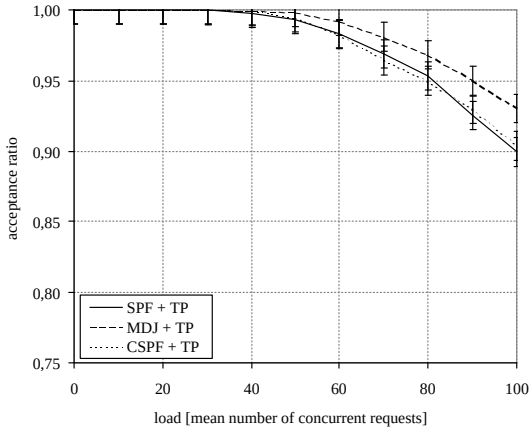

Figure 6.19: Impacts of Routing Schemes for a Premium Service

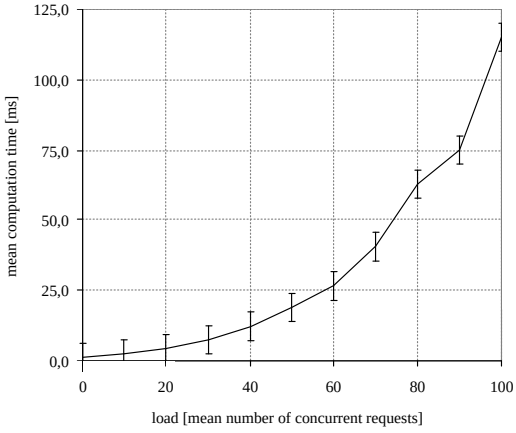

Figure 6.20: Mean Computation Time for Premium Requests

So far, simple SPF routing has been applied. Now, figure 6.19 extends the results obtained for the EPBOO option to CSPF and MDJ routing. Surprisingly, CSPF, which outperformed SPF routing significantly in case of the Guaranteed Rate Service, does not increase the performance for the Premium Service. This is due to the fact that the link capacity is not the limiting factor for the Premium Service, but the edge-to-edge delay. Thus, SPF and CSPF compute the same paths.

In contrast, the MDJ scheme outperforms both SPF and CSPF. The advantage of MDJ is again created by the second maximally disjoint path. As long as the first path is capable of providing the desired service, it is applied. However, if not, the second path provides a good alterative, since there is a reasonable chance that certain congested links are avoided.

Finally, Figure 6.20 addresses the computational complexity of the implemented algorithms. The theoretical complexity of the Network Calculus based admission control is in $\mathcal{O}(m^2 \cdot n)$ with m denoting the number of traffic trunks and n the number of links of the respective domain. However, there has been concern about the scalability of the approach that is followed here [72]. Therefore, we add results on the actually measured complexity of our implementation in Figure 6.20. The control procedures of our admission control have been implemented in C++. The external signalling interface is based on TCP sockets. The admission control is executed on a 2.4 Ghz Pentium 4 running Linux kernel version 2.4.19. Static SPF routing is applied. For this configuration, we find that the response time of the system is below 150-ms almost up to a load of 400 concurrently active requests, which clearly allows to apply our admission control in immediate reservation systems. Further on, we find that the actual implementation scales with less than a quadratic dependence on the number of admitted traffic trunks.

Chapter 7

Application Initiated Allocation of Label Switched Paths

7.1 Background

7.1.1 The General-purpose Architecture for Reservation and Allocation

The General-purpose Architecture for Reservation and Allocation (GARA) provides advance reservations and end-to-end management for Quality of Service on different types of Grid resources. GARA is a research prototype of the Globus Project, a project that is developing fundamental technologies needed to build Grids, and an advance reservation architecture for Grid infrastructures.

A GARA system is composed of a number of resource managers that each implement reservation, control, and monitoring operations for a specific resource. Resource managers have been implemented for a variety of resource types. A Bandwidth Broker is the resource manager of particular interest here. Uniform interfaces allow applications to express QoS needs for different types of resources in similar ways, thereby simplifying the development of end-to-end QoS management strategies. Mechanisms provided by the Globus Toolkit are used for secure authentication and authorization of all requests to resource managers. A directory service allows applications to discover resource properties such as current and future availability.

GARA has a four-layer architecture, as illustrated in Figure 7.1.

The layer that most programmers would use is the uniform remote access layer, via the GARA Application Programmers Interface (API). This layer provides three essential services. First, it allows reservation requests to be described in a simple, uniform way. Second, when a reservation has been granted, it is described by a unique, data structure called a handle that can be stored, transmitted, and used to refer to the reservation. This reservation handle is intended to be opaque, that is, the format of the data structure should not be interpreted by programmers or users outside of GARA. Third, it communicates with reservation services that may be located remotely. This is an essential feature, because it allows for coordinated reservations on different resources to be easily made and for reservations to be made in advance of when they are needed.

Applications also communicate with an information service that can inform them about likely reservations that can be made, and whom to contact to make them. By combining resource reservation and allocation with the ability to search the information service, GARA offers a flexible framework for the construction of higher-level scheduling services.

The GARA API communicates with a resource interface. The resource interface is responsible for authenticating and authorizing that the user is allowed to make a reservation. This layer is unaware of the specifics of the reservation, so it can only provide coarse authorization such as "Alice can make reservations", but not "Alice can only make reservations for bandwidths less than ten percent of the available bandwidth". That fine-grained authorization happens at a lower-level

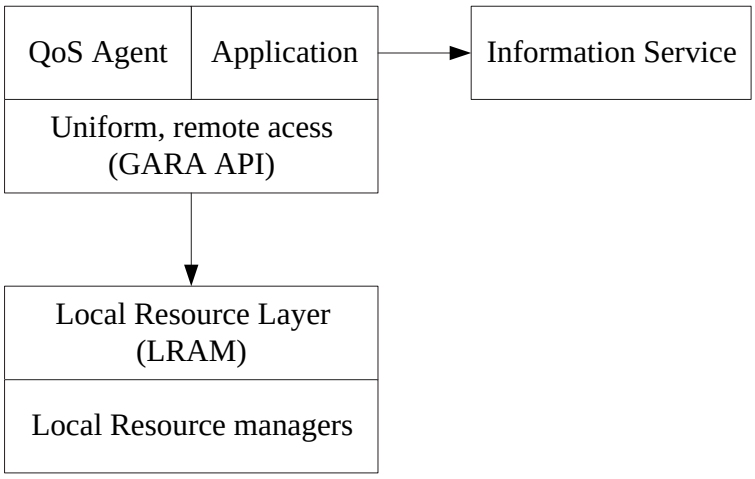

Figure 7.1: GARA's four-layer architecture. Applications and higher-level services use the GARA API, which communicate securely to the local resource managers level, which in turn communicates with resource managers. Applications also communicate with an information service to find out information about resources for which they can make reservations.

because it often depends on specifics of the available resource.

The resource interface passes the reservation requests to a lower-level interface. This is a "mostly uniform" interface. It is not completely uniform because it is unnecessary: this layer provides a thin shim between the resource interface layer and the resource manager level beneath. It is the responsibility of this layer to translate all incoming requests so that they can be presented to the resource managers that actually provide the QoS reservations.

The resource managers are responsible for tracking reservations and enforcing them by communicating with the lower-level resources, such as the network.

Instead of applications, there may be higher-level services at the top level. These handle QoS requests for applications, often interacting with the information service and making multiple reservations at the same time.

This four layer architecture allows for a uniform interface at the top, secure access to remote resources, and any number of QoS implementations.

7.1.2 Resource Managers: Service Provisioning for Grid Resources

In a sense, resource managers are the most important part of GARA, because without them there would be no QoS.

GARA was written to make it easy to integrate new resource managers written by other people, but several resource managers were also created just for use in GARA. GARA incorporates a network QoS resource manager that uses Differentiated Services, a prototype disk space QoS resource manager, and a CPU QoS resource manager that uses process priorities. Additionally, GARA offers a hybrid resource manager that interacts with the Dynamic Soft Real-Time (DSRT) CPU scheduler, but adds advance reservations on top of DSRT. The most advanced resource manager is a Bandwidth Broker for Differentiated Services networks.

To be used in GARA, resource managers need to have a few common features:

- **Advance Reservations:** Each resource manager must support advance reservations. If a resource manager does not support advance reservations, support can be added by using a hybrid resource manager on top of the resource manager, similar to the DSRT example mentioned above. To implement the related bookkeeping, GARA uses a simple but effective slot table manager. Here, reservations are considered as slots in time. Each slot represents

a single capacity delegation as a "slot" of time. These slot tables can be used by any resource manager to keep track of reservations. In addition to the provision of basic slot table operations such as creating, modifying, and deleting an entry, the manager can also deliver asynchronous events when a reservation begins or ends. Therefore, it offers an implementation framework for implementing advanced notification services as described above and can be reused in different resource managers.

- **Interaction with Resources:** Each resource manager needs to interact with the underlying resource in order to enforce the QoS. If the resource manager does not have complete control over the QoS, then reservations cannot be guaranteed.
- **External Interface:** Services provided by resource managers need to be accessed. Because GARA incorporates the interface of resource managers into a Grid resource management framework, it depends on the ability to interface directly with the resource manager. Note that GARA was implemented within the Globus framework which provides user delegation. It therefore knows which user is accessing a resource manager, and all interaction happens as that user.

Note that resource managers do not necessarily need to provide authentication or remote access, since that is provided through the higher levels in GARA. However, because GARA understands users and uses delegation when users authenticate, resource managers can do additional authentication.

7.1.3 Network Reservations

In a Grid environment, networks are essential to the smooth operation of the Grid, and they are also often the most shared resources. Therefore, a considerable effort was spent in ensuring that GARA could effectively manage advance reservations and QoS for applications that made demanding use of networks.

Grid applications have a wide-variety of network QoS needs which are much more challenging than other network QoS applications such as Voice over IP. This is because Grid applications use a wide variety of network flows, including low-bandwidth but low-latency control flows, high-bandwidth and low-latency data transfers for applications such as visualization, and high-bandwidth but not necessarily low-latency data transfers for moving large amounts of data for analysis and other purposes. This combination of types of network flow places strong requirements on the network QoS layers.

The Differentiated Services (DS) architecture is a reaction to the scalability problems of the flow-based approach of the Integrated Services architecture. While DS follows the fundamental principle of using a common infrastructure for both non-real-time and real-time traffic, it leaves the reservation based approach of the Integrated Services architecture. In contrast to serving individual flows it focuses on defining the behavior of aggregates. GARA builds a reservation scheme on top of DS.

Packets are identified by simple markings in the Type of Service (ToS) field of the IP-header that indicate how routers should treat the packets. In the core of the network, routers need not to determine which flow a packet is part of, only which aggregate behavior they should apply.

In this scenario, of course, the question arises of which packets will get marked. This is especially the case when the environment is dynamic, that is, when varying flows should be able to use the available services. Here, a particular resource manager called Bandwidth Broker is used. A Bandwidth Broker is a middleware service which controls and facilitates the dynamic access to network services of a particular administrative domain. GARA designs and implements a resource manager which fulfills the functions of a Bandwidth Broker.

DS allows packets to be marked either by applications or by the first router that receives the packets—the edge router. If edge routers mark packets, which is the more general solution, they could do so on any sort of basis—perhaps all IP packets for a particular protocol—but GARA marks packets that belong to a reservation. In order to enforce the reservation, packets are only

marked when they are "within profile"—that is, when the sender is sending within rate given to the reservation.

Core routers (those that are not on the edge) have an easier job because they do not need to identify packets that need marking, nor police packets to ensure they are within profile. Instead, core routers apply a particular packet treatment—called Per-Hop Behavior (PHB)—based only on these markings. Currently, the Internet Engineering Task Force's Differentiated Services Working Group has specified a small set of PHBs, namely the Assured Forwarding and the Expedited Forwarding PHB.

GARA uses the Expedited Forwarding (EF) PHB, which is intended to be used for a high-priority service with little jitter and queuing delay. The exact definition of EF is rather technical, so to simplify, each router interface can be configured so that traffic marked for the EF aggregate is prioritized over all other packets. To prevent starvation, we somehow have to limit the amount of data which is marked to belong to EF. This admission control is task of our GARA resource manager.

In order to implement a Premium Service based on EF, GARA assumes that each output link is configured to identify EF packets and to prioritize them appropriately by applying Priority Queuing. Note that this potentially requires a configuration update of all routers in a domain. Fortunately, this only has to be done once. While the admission control procedure uses the slot table manager to respond to reservation requests, reservations also have to be instantiated and policed in the edge routers. GARA dynamically configures the packet classifier, the policer, and the packet marker to appropriately map packets to EF. Figure 7.2 illustrates this scenario. Packet classification is done based on the reservation attributes specified when a user made a reservation. When applied to the scenario we describe here, it is done based on the end-points (address and port number) and the protocol (TCP or UDP).

Once the packets have been classified, a so-called "token bucket" is used to ensure that during the time interval $[t_0, t_1]$ of length $T = t_1 - t_0$ the amount of data sent does not exceed $r \cdot T + b$ Bytes. Here, r denotes the average rate at which the token bucket operates and b represents the depth of the token bucket which allows some limited bursts. Packets which fulfill this constraint will be marked to belong to EF. To do this correctly, GARA identifies and configures the relevant edge router every time a reservation is activated.

Note that applications may have a difficult time staying within the limits of their reservations. While monitoring the policing function and providing feedback to the application is appropriate for UDP-based flows, this mechanism does not work well for TCP-based communication. In order to assist applications, a third mechanism, called traffic shaping, is used for the traffic entering the edge router. The idea is to shape the injected TCP-traffic that it injects a smooth, reservation conforming rate to the core network. By incorporating this with the relaxed configuration of the policing function, TCP-applications can effectively be paced to use their reserved bandwidth.

7.1.4 Resource Reservations: A uniform API for Grid Applications

This subsection describes the exposed reservation interface offered by GARA. Let us take an example of how a programmer may make a reservation for network bandwidth needed tomorrow afternoon. First, the program has to make a list of the attributes needed for the reservation. GARA uses a text-based attribute-value representation of a reservation. The representation language currently used (the Globus Resource Specification Language, or RSL) is schema-free, but GARA has some standard attributes. A typical reservation request may look like:

```
&(reservation-type=network)
 (start-time=953158862)
 (duration=3600)
 (endpoint-a=140.221.48.146)
 (endpoint-b=140.221.48.106)
 (bandwidth=150);
```

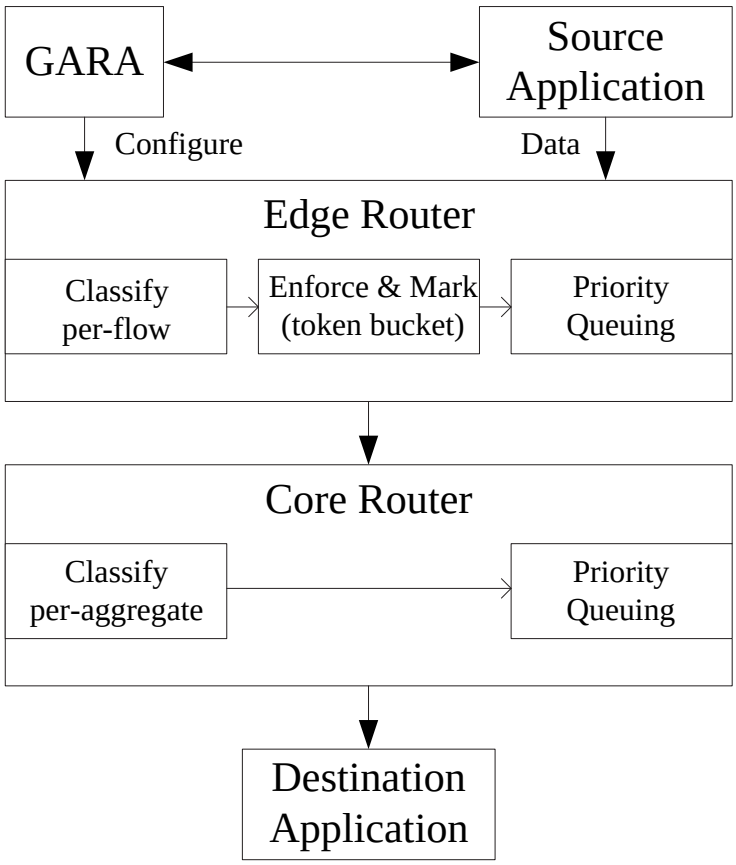

Figure 7.2: A simple network that shows how GARA uses DS. GARA configures edge routers to use classification, marking, and enforcement per-flow, and Priority Queuing is used in the core.

The first three fields (reservation-type, start-time, and duration) are used for all reservations. The last three fields are unique to network reservations.

To request the reservation, the programmer makes a simple request:

```
(error, handle) = reservation-create(resource-name, resv-desc);
```

Assuming there is no error, the reservation has been made. It can be queried at any time to find out the status of the reservation, such as if it has started:

```
(error, status) = reservation-status(handle);
```

There is also an asynchronous event delivery service that can inform a program about reservation related events, both simple state and more complex state. These events are sent by the resource manager. An example of a simple state event would be the information stating that the reservation time has begun. An example for a complex state would be the notification that the application is sending data faster than the reservation allows.

When a program is ready to use a reservation, it sometimes needs to inform GARA of the last-minute information that was not previously available. For example, a network reservation needs to provide the port numbers used by the TCP or UDP connection so that the network routers can provide QoS guarantees, but these port numbers are often not known in advance. Providing this information is known as binding the reservation. The binding information is provided as a textual

description, similar to the reservation request, then given to GARA:

```
bind_params = "(endpoint-a-port=1234)" + "(endpoint-b-port=5678)";
error = reservation-bind(handle, bind\_params);
```

When the program is done with a reservation, it can be cancelled:

```
error = reservation-cancel(handle);
```

Note that the information passed within a bind request is always related to the requested type of service. GARA uses RSL to provide a generic interface. As much as possible, these parameters are kept consistent in GARA, but they must change to reflect the underlying properties of the QoS. Beyond this difference though, GARA present a uniform interface to the underlying QoS providers.

7.1.5 An Implementation for the Globus Toolkit

The current implementation of GARA uses the Globus Toolkit as a foundation. It was initially implemented as part of Globus 1.1.3, although a port to Globus 2.x is available. Note that the results of the EU-funded Grid Interoperability Grid Project (GRIP) led by FZ-Jülich can be used as a basis for the incorporation of Globus services into a UNICORE-based Grid. This conveniently gives a roadmap for application initiated network services in both frameworks: UNICORE and Globus.

The four layers of the GARA architecture shown in Figure 7.1 map closely to the layers of the Globus Toolkit, as shown in Figure 7.3. The GARA API, which resides in the remote access layer, corresponds closely with the Globus Resource Allocation Manager (GRAM) API. The resource interface layer is the Globus gatekeeper interface. The gatekeeper is responsible for authenticating and authorizing all GRAM and GARA interactions with a system. The local resource access layer and the local resource managers do not have exact analogies in the Globus Toolkit, but were implemented completely within the GARA framework.

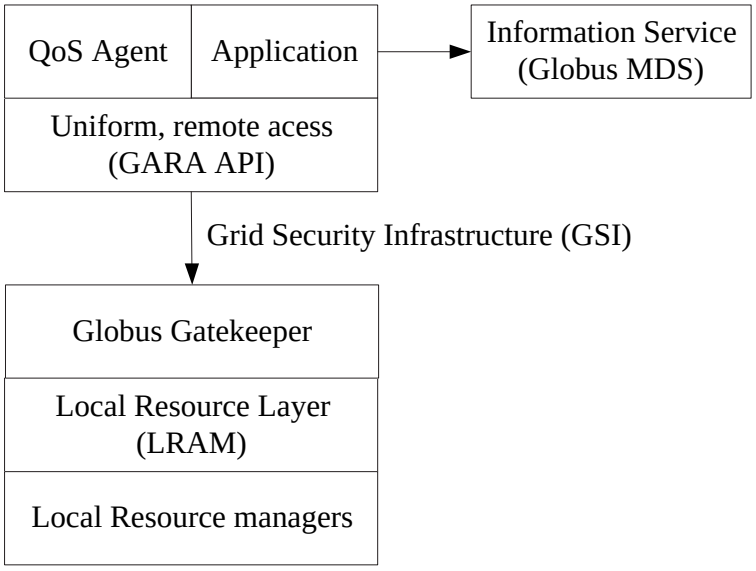

Figure 7.3: The GARA architecture from Figure 7.1 as it is implemented within Globus.

Because the protocol for communication with the gatekeeper and the security mechanisms were already completely existing within the Globus Toolkit, we were able to easily leverage them without any loss of generality or flexibility in the overall architecture of GARA.

Security

Globus uses the Grid Security Infrastructure (GSI) . The GSI normally uses public key cryptography. Users have private keys that they never share, and public keys (called certificates) that anyone can view. An important aspect of GSI is that it allows users to delegate credentials. To do this, a user can create a proxy certificate which has a new public and private key and is signed by the user's private key. However, it usually has a much shorter life time, generally on the order of twelve to twenty-four hours. This proxy certificate can then be used for authentication. If the proxy should happen to be compromised, it will be useful for a much shorter time than the user's private key.

The Gatekeeper Protocol

GARA uses GSI to authenticate with the gatekeeper. After authentication, the gatekeeper passes the network connection to another program called the GARA service. This GARA service uses the Local Resource Manager (LRAM) API to interact with the local resource managers. Each GARA API call is a transaction with the gatekeeper, so each call benefits from the security and remote access capability.

The GARA API allows users to request callbacks that inform the user when changes to the reservation occur. These do not use the gatekeeper for callbacks, but retain the connection originally opened to the gatekeeper by redirecting the sockets to another program that provides the callbacks.

Mapping of Service Requests to Resource Managers

As mentioned above, the GARA service uses the LRAM API. This is similar to the GARA API, but it does not provide remote access or security. It does provide an abstract interface to the resource managers so that the GARA service does not require intimate knowledge of different resource managers. Instead, the LRAM API knows the details of speaking to the resource managers.

The LRAM is implemented as a series of libraries that can be used to communicate with different resource managers. While it was written in C, it is essentially an object-oriented framework that allows for abstract interfaces to a variety of implementations.

Existing Resource Managers

Several resource managers have been built:

- **Network Resource Manager:** This uses Differentiated Services, as described above.
- **Computer Resource Manager:** This resource manager was implemented just in the LRAM layer because it was an interface to the Portable Batch System (PBS) which provides advance reservations. Because of the semantics of PBS advance reservation matched those of GARA reservations, PBS itself is used as a resource manager. To use the reservation, GARA extended the Globus GRAM interface to aid in claiming the reservation: this allowed users to merely provide the reservation handle when submitting a job, and the reservation was correctly used.
- **CPU Resource Manager:** This uses the Dynamic Soft-Real Time (DSRT) scheduler that allows us to guarantee a percentage of the CPU to an application. At the time, DSRT did not provide advance reservations, so we built a hybrid resource manager that provides advance reservations to DSRT.
- **Graphic Pipeline Resource Manager:** This resource manager provides advance reservations for graphic pipelines on a Silicon Graphics system.

- **Prototype Resource Managers:** Various prototype resource managers exist. This includes a disk space resource manager, a resource manager that provides exclusive access to a remote cache service, and a CPU resource manager that used priorities.

7.2 MPLS-related Improvements

We now briefly describe the steps performed for prototyping an MPLS-aware resource manager for GARA.

7.2.1 Adding topology information

Admission control is an essential building block for service provisioning based on the Differentiated Services architecture. GARA uses a State Repository that is responsible for the bookkeeping of reservations. In detail, it uses a slot-table model where all reservations are placed in a co-ordinate system based on the start- and end-time (x-axis) and their requested amount of the service class (y-axis). The latter is bandwidth, as an absolute value that is related to the maximum amount of bandwidth allowed for this class (configuration parameter). Reservations are thus stored in forms of rectangles in a co-ordinate system. Whenever a reservation request arrives, it must be able to place the related rectangle in the coordinate system without any overlap; otherwise the reservation exceeds the dedicated service amount for at least a small period of time.

In its simplest form, a single slot-table is used to control the admission of service requests. This State Repository is extended by a model in such a way that the Bandwidth Broker maintains a slot-table per link. Every reservation request that has a particular ingress and egress point to the controlled domain must now somehow be consistently mapped to the related path in the topology. To support this dynamic mapping, an additional abstraction called "traffic trunk" was introduced. A traffic trunk connects an ingress and an egress router with a particular level of service, i.e. a particular Per-Domain Behavior (PDB). Hence, a traffic trunk is the persistent representation of an accepted service request. Note that a trunk is always uni-directional. It only provides guarantees for packets traversing from the source downstream to the destination. Any upstream guarantees have to be requested separately. This vision of a traffic trunk nicely fits to the view of a Label Switched Path in MPLS.

For the State Repository the introduction of traffic trunks means that it has to persistently store their attributes. Trunks represent the unit of interest for Bandwidth Brokers, as they represent the granularity of interest in both environments: end- and transit-domains. Network reservations are always represented by traffic trunks. Whenever the reservation is instantiated or the State Repository is updated, traffic trunks are dynamically mapped to the topology using an admission control procedure which follows the shortest-path routing decisions done by the IP control plane.

We incorporated the PAB-topology into the modified GARA-codebase by using the following configuration file:

```
Router zam576 192.168.32.241
RouterLogin login_pw enable_pw1
Interface ATM1/0/0.1 192.168.32.9/30 1
InterfaceSpeed 150M full-duplex1
Interface FastEthernet0/0 192.168.31.9/301
InterfaceSpeed 100M full-duplex1
Interface FastEthernet0/1 192.168.32.241/301
InterfaceSpeed 100M full-duplex1

Router zam577 192.168.35.9
RouterLogin login_pw enable_pw
Interface FastEthernet0/0 192.168.32.242/30
InterfaceSpeed 100M full-duplex
```

```
Interface FastEthernet0/1 192.168.35.9/30
InterfaceSpeed 100M full-duplex
Interface ATM1/0.1 192.168.32.17/30
InterfaceSpeed 150M full-duplex
Interface ATM4/0.1 192.168.32.25/30
InterfaceSpeed 150M full-duplex

Router zam578 192.168.34.9
RouterLogin login_pw enable_pw
Interface FastEthernet0/0 192.168.34.9/30
InterfaceSpeed 100M full-duplex
Interface FastEthernet0/1 134.94.173.106/16
InterfaceSpeed 100M full-duplex
Interface ATM1/0.1 192.168.32.10/30
InterfaceSpeed 150M full-duplex
Interface ATM4/0.1 192.168.32.18/30
InterfaceSpeed 150M full-duplex
Interface GigabitEthernet2/0 192.168.32.233/30
InterfaceSpeed 1000M full-duplex
Interface POS3/0 192.168.32.33/30
InterfaceSpeed 150M full-duplex

Router zam579 192.168.33.9
RouterLogin login_pw enable_pw
Interface FastEthernet0/0 192.168.33.9/29
InterfaceSpeed 100M full-duplex
Interface FastEthernet0/1 134.94.204.17/28
InterfaceSpeed 100M full-duplex
Interface GigabitEthernet2/0 192.168.32.234/30
InterfaceSpeed 1000M full-duplex
Interface POS3/0 192.168.32.34/30
InterfaceSpeed 150M full-duplex
Interface ATM1/0.1 192.168.32.26/30
InterfaceSpeed 150M full-duplex

# Links between routers
Link full-duplex zam576 zam577 100M 1
Link full-duplex zam576 zam578 155M 1
Link full-duplex zam577 zam578 155M 1
Link full-duplex zam577 zam579 155M 1
Link full-duplex zam578 zam579 155M 1
Link full-duplex zam578 zam579 1000M 1
```

Note that an additional (legacy) configuration file is used to describe the association between ingress-egress links and the related interface name.

7.2.2 The Explicit-Route object

So far, the described modifications do not allow the incorporation of traffic engineering mechanisms as they are analyzed with the PAB-project. The resource manager performed admission control by following the routing decisions of the network. Combining this with the bandwidth reservation mechanisms offered by Cisco's automatic bandwidth adjustment mechanism, the admission control can be done in advance.

The active use of the explicit route object in the RSVP-TE messages, however, allows for an incorporation of flexible and efficient traffic engineering algorithms. Here, the resource manager is not just following the routing decisions of the network. Instead, it enforces a particular route by the use of the explicit route object.

Whenever a new service request arrives, it is dynamically mapped to the underlying topology. A particular function was introduced to GARA that handles this mapping. In detail, the function searches the network graph for a valid path, validates for each link that the requested capacity is still available over the particular time interval, and finally creates an explicit route object encoding used during service provisioning.

7.2.3 Service Provisioning

GARA's service provisioning interface uses the command line interface of the edge router. A particular TCL/Expect script is used to automate a telnet session that performs the related configuration steps. Incorporating MPLS functionality to GARA means that this interface must be updated according to the CLI steps to be performed.

In detail, the updated script:

- Creates an explicit route object
- Creates a tunnel interface
- Creates a route-map entry
- Creates appropriate packet classifiers
- Creates an appropriate packet marker
- Sets the tunnel interface as path for traffic in the route-map
- Assigns the route-map to the related ingress interface

Chapter 8

Advanced Network Management

The network resource manager in a prototype version uses a static configuration file containing a description of the network topology. This can lead to unwanted effects when the network map changes and this fact is not recognized by the running resource manager. It is then of great importance to keep the network topology and other network related information up-to-date. The network topology can be deduced from the link state database of the Open Shortest Path First (OSPF) protocol which is usually included in a Management Information Base (MIB) maintained by a router. This information from the MIB can be remotely accessed using the Simple Network Management Protocol (SNMP). These standards are supported by the routers in the DFN network as well by the CISCO 7200 routers in the testbed of the PAB project. The network monitoring was implemented in Java classes. In the future it is planned to extend the functionality of the software by the ability to configure the routers and not only to monitor them. Integration of the passive and active controlling functions in one class will simplify the implementation of advanced mechanisms for QoS provisioning in IP/MPLS networks.

8.1 SNMP-based Monitoring

8.1.1 MIB and SNMP

A MIB is a database organized in a hierarchical way. All items in this database are addressed by the sequence of numbers separated by dots, similarly to a file disk location, which is given by a sequence of directories (separated by slash or back-slash - depending on the operating system) terminated by the name of the file. Thus all items from one thematic group of the MIB database have addresses with the same prefix (like 1.3.6.1 for the Internet, 1.3.6.1.2.1.4 for IP).

The MIB database can be accessed remotely with the help of SNMP. In its basic version 1 "SNMP v1", which was implemented in the developed software, five operations are defined, see Figure 8.1:

- `get`
- `get-next`
- `get-response`
- `set`
- `trap`

The first two operations `get` and `get-next` are used to request data. An answer comes with the `get-response` message. The `set` operation is foreseen for modifying the variables in the MIB database. It is only possible when the access to the database is not write-protected. In fact,

the CISCO implementation of the MIB does not support modification of the items by the `set` command.

Sending of `trap` messages is initiated by the system where a MIB database is installed. It helps the SNMP manager to be up-to-date after any change in the network status. It should be stated, however, that the SNMP messages are transported on UDP which does not guarantee the delivery of packets.

The newer version of the protocol "SNMP v2c" introduced the new `get-bulk` command which allows to send a big number of MIB items in a single SNMP message. However, the simple authentication procedure, which is based on sending an unencrypted password - so called "community string" - is directly adopted from "SNMP v1". It should be noted that there are also other variants of SNMP referred to as version 2. However "SNMP v2c" described here is the most widely spread and popular one.

The latest version "SNMP v3" is the most advanced one. It implements sophisticated security algorithms for user authentication and data encryption.

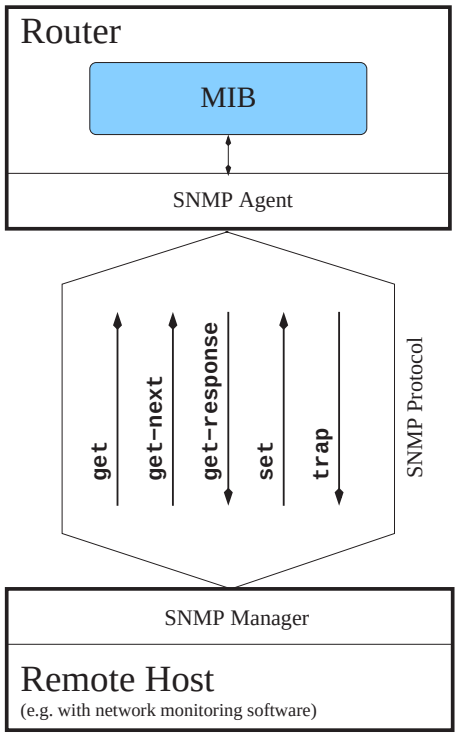

Figure 8.1: The communication between the SNMP manager on a remote host and the MIB database via SNMP. Five types of messages: `get`, `get-next`, `get-response`, `set` and `trap` can be exchanged in case of the protocol version 1.

8.1.2 Java Implementation

The monitoring code was written in Java (version 1.4.1) for portability reasons. The software has a layered structure, see Figure 8.2. At the bottom there is Westhawk's Java SNMP Stack class library (free-ware software downloaded from the www.westhawk.co.uk site) which was used to organize the communication with the routers via SNMP. The stack supports all three versions of SNMP, described in the previous section. Although the current tests were made with the most simple one, SNMP version 1, the final product also supports the functionality of the most advanced, SNMP version 3. On top of Westhawk's Stack the Network Class Library (current

version: 2.1) was developed which abstracts the elements like MIB databases, routers, network topology, MPLS tunnels, etc. in Java classes. This library offers building blocks for creating higher level applications controlling network state. In the next section the Network Monitoring program is presented as an example of an application using the Network Class Library.

The content of the MIB database is represented by the `MIB` object which is responsible for regular updates and which can pass the information to the higher level objects: `Router` and `Topology`. They contain lists of interfaces, MPLS tunnels, active links, etc. The `Testbed` object contains all `Router` objects and the `Topology` object and serves the information relevant to the network domain, like a global list of links and their states, a list of all MPLS tunnels, the topology of the network, etc. Applications built on the top of the Network Class Library will use the `Testbed` object as a basic source of information about the network domain. At this level all the details concerning SNMP communication with hardware are hidden in lower layers.

Due to the nature of the OSPF routing protocol, all routers in the domain should have the full and consistent information about the network topology within the domain. So, the choice of the router delivering this information is arbitrary and in the performed tests the router *zam576* was selected. Since the addresses of all other routers in the domain can be obtained from the topology database, the only parameters which the Testbed class requires as input data is the IP address of *zam576* and the access passwords (community strings) for the SNMP service (which are chosen to be common for all routers in the domain).

One of the methods of the Testbed object can create the configuration string for the resource manager in the format described in Section 7.2.1.

The objects from the Network Class Library are updated periodically with the programmed refresh time but this process can be also triggered by the application or by receiving a trap message. Thus, the software can quickly react on changes of the network parameters.

The current version of the Network Class Library makes use of the following MIB sub-databases:

- 1.3.6.1.2.1.1 - general system information like, router's name, location, administrator, etc.,
- 1.3.6.1.2.1.2 - information related to physical properties of router's interfaces,
- 1.3.6.1.2.1.4.20 - information related to the router's IP layer,
- 1.3.6.1.2.1.14.4 - OSPF link state database,
- 1.3.6.1.3.95 - CISCO information about MPLS tunnels,
- 1.3.6.1.4.1.9.9.109.1.1.1.1.3 − 5 - three values characterizing the CPU load of the router's processor.

8.1.3 Network Monitoring Application

Based on the Network Class Library, described in the previous section, the application was created for visualization of the network topology and MPLS tunnel layout, see Figure 8.3. In the left panel the list of MPLS tunnels is given. The bright grey color denotes the active tunnels while the dark grey color denotes the tunnels which were configured but for some reasons are not operational, e.g. due to link failure. The panel next to right shows the details for the selected tunnel. Here, the content of the Explicit Route Object is shown as well as the list of links and interfaces used by the tunnel. The most right panel displays the network topology and the path used by the selected tunnel. The operational interfaces are shown as grey spots near the routers. Other colors are reserved for not operational or administratively down interfaces. Links are represented by lines connecting interfaces. If a given link is advertised by the routing protocol, then the line ends with an arrow. The selected tunnel path is shown by arrows placed along the lines representing links. Additionally, the CPU load of the routers is given. The interface related information appears in a rectangular field if the mouse pointer is close to the interface spot.

This example application shows some possibilities of using the designed Java classes to monitor the network domain. It can be easily extended to monitor other parameters of the network.

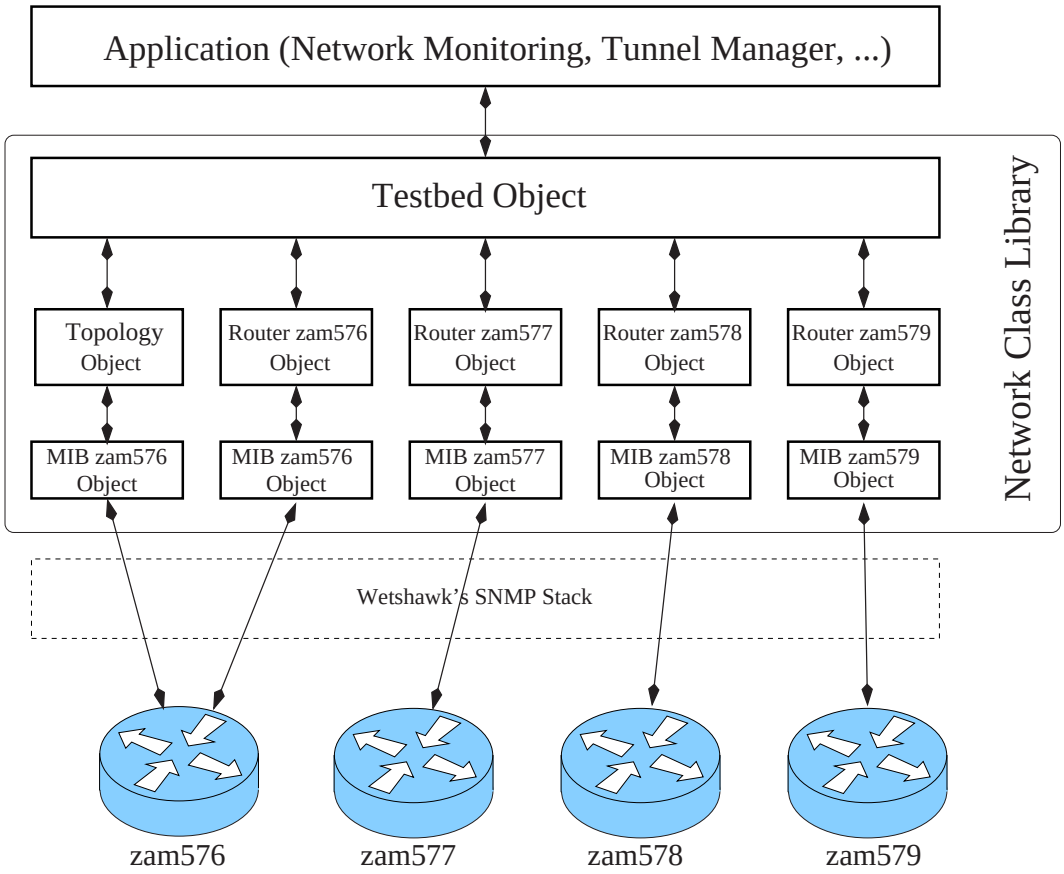

Figure 8.2: The architecture of the software. For low level communication via SNMP Westhawk's SNMP Stack is used. The data from MIB databases is represented by objects shown as rectangles. The application at the highest level uses the `Testbed` as a basic source of information about the network

8.1.4 Software Integration

The presented software is still being developed for the PAB project to include the possibility of the automatic network configuration, including MPLS tunnel set-up. Then the network monitoring functions will be completed with management possibilities. However, already at the current stage the monitoring system can be integrated with the existing network resource manager since the latter can read the topology information prepared by the `Testbed` object. The new version of the Java class library will allow to build much more sophisticated applications on top of it, which will be able to dynamically reconfigure the network according to user requests or to control and optimize it automatically.

8.2 Architecture

The *Network Class Library* is a set of Java classes but only few of them are intended to be created directly by a programmer. These special classes can be grouped into following categories:

- **Communication with routers.** There are two Java classes in this group:
  - `MIB`. It represents a set of entries in a router's MIB database. The meaning of those entries is transparent to the `MIB` class. Important are only their MIB addresses and

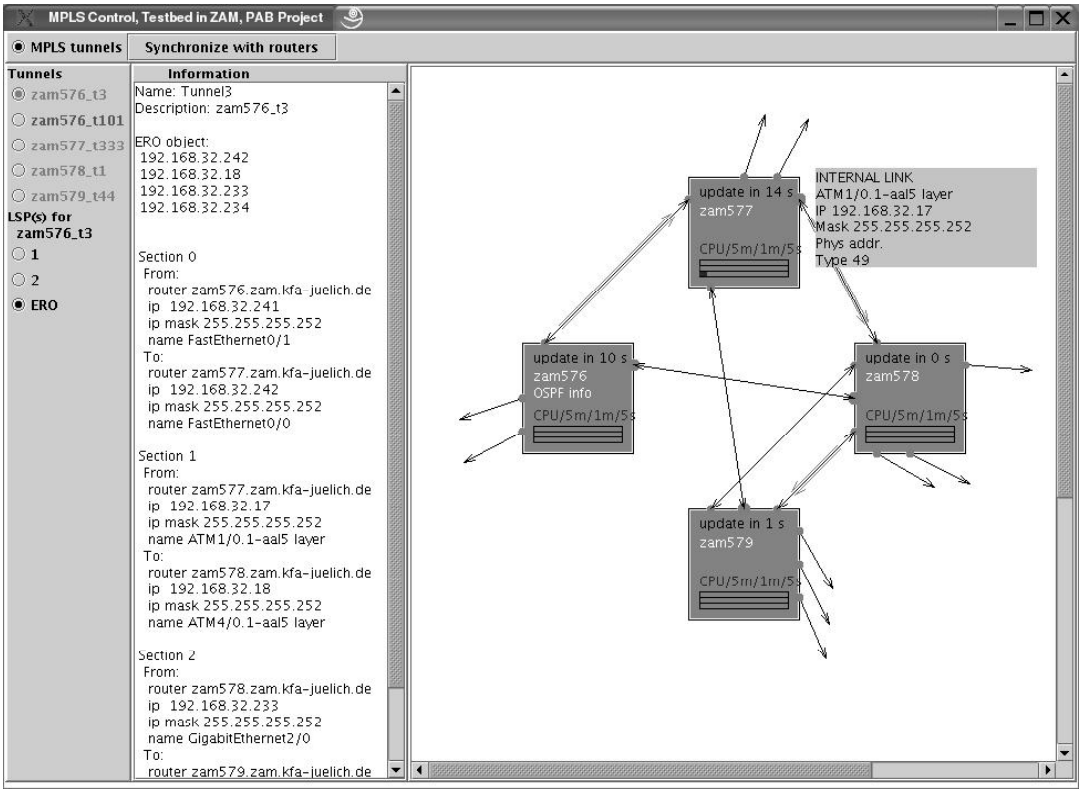

Figure 8.3: A screen shot of the network monitoring application. See description in the text.

assigned values. All SNMP communication details are hidden from a programmer in the implementation of the `MIB` class which uses the *Westhawk's SNMP Stack* software, distributed as a free-ware.

- `CLI`. It represents a *Command-Line Interface* (CLI) communication process. This class has methods to perform commands in the routers *EXEC* and *CONFIGURATION* modes. Its implementation uses the *The Java Telnet Applet* being a free-ware software which is distributed under terms of the *GNU General Public License* as published by the Free Software Foundation.

- **Network representation.** There are three Java classes in this group as listed below. All of them use an object of the `MIB` class as a basic source of information about a network. Additionally, a `CLI` object is used by some `Router` methods.

  - `Topology`. This class represents a network's topology. It contains lists of objects describing OSPF *Link State Advertisement* (LSA) messages of different types.
  - `Router`. This class represents a router. It allows to extract a lot of router-related information like e.g. the number of interfaces, their states and configurations. One of its methods gives access to a `MIB` object associated with the router.
  - `Testbed`. This class represents a network domain, i.e. a set of routers, links, MPLS tunnels, etc. Some of its methods return `Router` or `Topology` objects related to particular routers in a testbed or the topology respectively.

- **Tools.** Classes from this group can be useful for a programmer using the *Network Class Library*.
  - `TaskManager`. This class can be helpful in performing tasks, represented by the `Task` class and its subclasses, which block the execution of a current thread. For this purpose the `TaskManager` uses its own thread for executing tasks. Moreover, it implements a task waiting list.
  - `Task*`. These classes describe tasks to be performed by the `TaskManager`.
  - `OutputQoS`. This class can be useful in obtaining QoS parameters from a policy map configuration object of the `PolicyMapCLI` class. See the Java documentation for details.

Objects of other classes, i.e. those objects which cannot be created directly by a programmer, can be returned by methods of some other objects, e.g. `Interf` object representing a router interface is returned by the `Router.getInterf()` method.

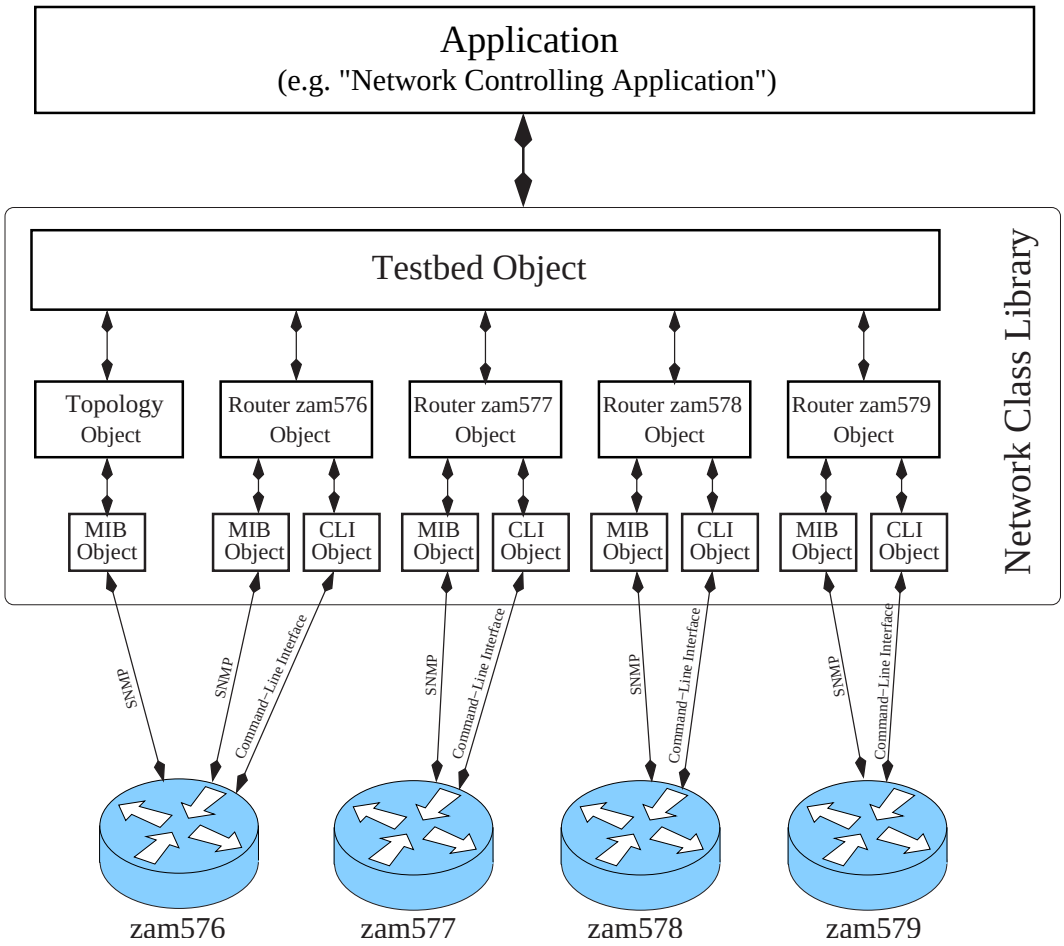

Figure 8.4: Schematic drawing showing relations between objects from the *Network Class Library*, a network testbed installed at the Research Center Jülich and a user application, see Section 8.4. `MIB` and `CLI` objects are used as building blocks for SNMP and CLI communication. `Topology` and `Router` objects represent OSPF messages and routers respectively.

The relations between *Network Class Library* objects are shown in Figure 8.4. In the next subsections the classes from this library will be described in detail.

8.2.1 `MIB` class

The `MIB` class implements an interface to a MIB database maintained by a given router. It has two constructors: one for SNMP version 1 and the second for SNMP version 3. In both cases, the object communicates at first with the router using the IP address passed to its constructor. However, it can happen later that this address is not reachable due to e.g. link failure. To be prepared for such cases the `MIB` object maintains a list of alternative router's addresses. Every time the current address is not accessible a new address from the list is tested until an accessible one is found. The access address which is currently used is returned by the `getAccessInterfaceAdress()` method.

A programmer is usually interested only in some subset of the whole MIB database maintained by the router. Thus, the `lookAtOid()` method is used to specify which MIB entries should be monitored by the software. The mentioned method takes two arguments: a common prefix of the MIB addresses and a time interval between periodically contacting the router.

As mentioned above the router is being contacted periodically with a predefined frequency. Additionally, a programmer can force an immediate update of the object by invoking the `setUpdateNow()` or the `refresh()` method. The first one exits before the update is done (what can be monitored by the `isUpdateNow()` method) while the second blocks the execution of the current thread until all MIB entries are re-read from the router's MIB.

A programmer can extract the MIB entries with the `getMIBItemList()` method which returns the `MIBItemList` object. The latter class implements two useful methods: `getMIBItem()` and `getMIBSublist()` which return a value for a complete MIB address or a list of values for a given MIB address prefix.

The `MIB` class implements recording of SNMP trap messages and sending of trap events notifications to other objects[1]. A single trap message can contain many MIB entries which are represented by an object of the `MIBItemListOffline` class. The list of received trap messages is returned by the `getTrapMIBList_c()` method and the oldest trap message is returned by the `getTrapMIBSubList()` method.

8.2.2 `CLI` class

The `CLI` class allows communication with routers via the Cisco *Command-Line Interface*. An object of this class is used in the `Router` object implementation. The `CLI` constructor expects an IP address to contact a router and two passwords necessary to access the system.

The class implements two groups of methods for executing commands in the *EXEC* and *CONFIGURATION* modes respectively: `execCommand2()` and `execCommand3()`. These methods return strings with the history of telnet sessions. The `isError()` method can be applied to check whether a given command was successfully executed or not.

A timeout for communication can be set with the `setTimeout()` method (the default value is $10\,s$).

8.2.3 `Topology` class

An object of the `Topology` class contains lists of OSPF *Link State Advertisement* messages of different types, received by a given router. A `MIB` object associated with this router is passed as an argument to the `Topology` constructor.

It should play no role, which router in the network domain serves the topology information. This is because of the flooding mechanism of OSPF advertisements - every router should see the same set of messages.

The `getLSA1_c()`, `getLSA2_c()`, and `getLSA5_c()` methods return lists of objects representing *LSA* messages of type 1, 2, and 5 respectively. This is sufficient to build the topology information of the network domain.

[1] The mechanism of sending trap event notifications uses a `Observable-Observer` model from the standard Java library

In the *Network Class Library* a `Topology` object is one of the members of the `Testbed` class, see Section 8.2.5.

The detailed description of the OSPF protocol and the format of LSA messages can be found in *RFC 1583*.

8.2.4 `Router` class

The `Router` class represents information related to a given router. An appropriate `MIB` object has to be passed to the `Router` constructor. A programmer can access this object later by invoking the `getMIB()` method. The `MIB` object constitutes a basic source of information for the `Router` class except the router's configuration text which is obtained with the help of a `CLI` object. The latter is created dynamically in `execCommand2()` and `execCommand3()` methods.

Most of the `Router` content is created in the `update()` method[2]. The most relevant information related to the router can be returned by the following methods:

- `getSysName()` - returns the name of a router,
- `getSysDescr()` - returns a text containing the system description,
- `getInterf_c()` - returns a list of `Interf` objects representing the router's interfaces,
- `getTunnel_c()` - returns a list of `Tunnel` objects representing the MPLS tunnels configured at this router,
- `getLSP_c()` - returns a list of `LSP` objects representing the *Label Switched Paths* passing through this router,
- `getConfigFile()` - returns a string with the complete router's configuration.

See the Java documentation for the full set of available methods.

8.2.5 `Testbed` class

This class represents a network domain. To create a `Testbed` object one needs to know the access address of only one router in the domain together with the appropriate parameters for SNMP and CLI access. These parameters are passed to one of the constructors of the `Testbed` class.

Below one can find a brief description of the various tasks performed by the `Testbed` constructor.

- **Learning the network topology.** An access address of one of the routers (one of the constructor's arguments) is used to create a `MIB` object which is then passed to a `Topology` object.
- **Creating a list of routers.** IP addresses of all routers in a network domain are known from the `Topology` object. Associated `MIB` objects are created and finally passed to the constructors of `Router` objects.
- **Invoking the `update()` method**.

In turn, the `update()` method, which can be invoked by a programmer at any time, performs the following tasks:

- **Updating topology and router information.** The `update()` method of the `Topology` object and the `update()` methods of all `Router` objects are invoked. It is also checked if any new routers are advertised by the `Topology` object. In this case the list of routers is extended to include new machines.

[2] Note that the constructor does not invoke `the update()` method.

- **Creating a list of internal links.** A list of `Router2Router` objects, describing internal links in the network domain, is created.
- **Creating a list of external links.** A list of `Router2External` objects, describing external links going out of the network domain, is created.
- **Creating a list of MPLS tunnels.** A list of `Tunnel2` objects, describing MPLS tunnels established in the network domain, is created.

Below one can find a list of the most relevant methods implemented in the `Testbed` class:

- `getRouter_c()` - returns the list of `Router` objects representing routers in a network domain,
- `getTopology()` - returns the `Topology` object representing the OSPF messages flooded in the network domain,
- `getInternalLink_c()` - returns the list of `Router2Router` objects representing the links between routers in the network domain,
- `getExternalLink_c()` - returns the list of `Router2External` objects representing the links going outside the network domain,
- `getTunnel_c()` - returns the list of `Tunnel2` objects representing the MPLS tunnels configured within the network domain,
- `makeTunnel()` - creates an MPLS tunnel,
- `update()` - invokes the `update()` method of the `Topology` and `Router` objects and creates updated lists of internal links, external links and MPLS tunnels.

See the Java documentation for the full set of available methods and their arguments.

8.3 Simple examples

In this section the basic functionality of the $\widehat{\text{MIB}}$, `Router` and `Testbed` classes is presented and illustrated with simple examples, which were tested with the network testbed installed at the Research Center Jülich. These examples, however, are not intended to replace the full Java documentation of the *Network Class Library*.

8.3.1 Configuration of Cisco routers

The presented software communicates with routers using SNMP in version 1 or 3. Both options are supported by the latest versions of the Cisco operating system, however the remote systems must be first appropriately configured for both accepting SNMP requests and sending SNMP traps. The configuration steps are partially different for SNMP version 1 and SNMP version 3. See Cisco OS documentation for the full description of the configuration commands and their parameters.

SNMPv1

SNMP version 1 is based on a very primitive security mechanism with the *community string* being a kind of password used in MIB access authorization. However, this string can be easily extracted from the SNMP packets since it is sent not encrypted. The following command executed in the router *CONFIGURATION* mode enables responding to SNMPv1 requests.

```
zamxyz(config)#snmp-server community xxx ro
```

The `xxx` string above is a *community string* which should be identical with a string from an SNMP request. Only then the request will be answered. The `ro` option causes that the access to the MIB database will be read-only.

On the other hand the command

```
zamxyz(config)#snmp-server host 10.10.10.10 yyy
```

causes that a router is ready to send traps to the IP address `10.10.10.10` with the `community string yyy`. An `snmp-server enable traps (...)` command specifies what kind of traps will be sent. For example:

```
zamxyz(config)#snmp-server enable traps snmp linkdown linkup
```

initiates trap sending in situations when any link is shut down or restored. See Cisco documentation for the full list of available options.

SNMPv3

SNMP version 3 offers much more advanced security than SNMP version 1 with its *community string* based concept. In SNMP version 3 every message sent over a network can be encrypted and signed with a special hash code. Thus, a potential intruder cannot decrypt a message or modify it without knowing appropriate secret keys.

A router which offers MIB access through SNMP version 3 must maintain a database of users which are authorized to contact the MIB. Each user belongs to a group and for every group the access rights to the MIB database are defined. The secret keys used by security procedures (one for encryption/decription and one for calculating hash codes) are calculated from the users's password and the SNMP identity number assigned to the router.

First of all the router should have assigned an `engineID` number. One does it with the command as shown in the example below:

```
zamxyz(config)#snmp-server engineID local 0000000902000005DD531800
```

Here, the parameter standing behind the word `local` is the `engineID` number which identifies the router in the SNMP communication. Now, a new user can be introduced. The command below executed in the router *CONFIGURATION* mode introduces a new user to the user database.

```
zamxyz(config)#snmp-server user userx groupx v3 auth md5 authpassword
                           priv des56 privpassword
```

For security reasons above command will not be shown in the routers configuration file. Here, the user name is `userx` and it belongs to a group `groupx`. The `authpassword` is used to calculate a secret key for MD5 (option `md5`) authentication procedures (option `auth`). Similarly, the `privpassword` is used to calculate a secret key for DES56 (option `des56`) encryption/decryption procedures (option `priv`)[3].

For a complete configuration one has to define a group, the same as typed above during user definition.

```
zamxyz(config)#snmp-server group groupx v3 auth
```

For a group definition one specifies the security level. Here `auth` means that authentication procedures must be applied when sending/receiving SNMP messages.

Next, to allow sending SNMPv3 traps one has to configure the router similarly as in this example:

```
zamxyz(config)#snmp-server host 10.10.10.10. version 3 auth userx
```

where `10.10.10.10` indicates the IP address of the trap receiver. To define which kind of trap messages should be sent use the same command as described in the previous subsection related to the SNMPv1 configuration.

[3]Some versions of Cisco IOS support only authentication without message encryption

8.3.2 Getting a list of MPLS tunnels - the `Router` object

The Java program presented here creates a `Router` object representing one of the routers in the testbed. This router has been configured to be a head of two MPLS tunnels. The code below shows how to use the *Network Class Library* to get certain information about those tunnels.

```
import fzjuelich.testbed.*;                                          //  0
import java.lang.Thread;
import java.util.*;

class TestRouter{
  static public void main(String[] args){

    MIB mib = new MIB("192.168.34.9", 161, "xxx", "Standard", true);  //  1
    Router router = new Router(mib);                                  //  2

    while(true){
      try{
        router.update();                                              //  3

        List list = router.getTunnel_c();                             //  4
        Iterator it = list.iterator();                                //  5

        String routerName = router.getSysName();                      //  6

        System.out.println("======================");                //  7
        System.out.println("Router: " + routerName);

        while(it.hasNext()){                                          //  8
          Tunnel tunnel = (Tunnel)it.next();                          //  9
          String tunnelSource = tunnel.getSrc();                      // 10
          String tunnelDestination = tunnel.getDst();                 // 11
          System.out.println("Tunnel from " + tunnelSource +          // 12
                             " to " + tunnelDestination);
        };

        Thread.sleep(15000);                                          // 13

      }catch(Exception e){                                            // 14
        System.out.println("Exception!!!");
      };
    }
  }
}
```

Below some lines of the program are commented.

- **// 0** - The **import** commands specify packages which should be loaded. One of them is the **fzjuelich.testbed** package containing the *Network Class Library*.
- **// 1** - The `MIB` object is created. Its constructor needs 5 arguments:
  - **"192.168.32.241"** is the IP address of the router, where a MIB database is maintained.
  - **161** is the IP port number for SNMP communication.
  - **"xxx"** is the community string, necessary when accessing the MIB database via SNMP.

  - **"Standard"** is the string defining the MIB access type.
  - **true** informs that this object will monitor SNMP traps.

- // **2** - The **Router** object representing the router is created. Its constructor takes the **mib** object as an argument.
- // **3** - The **update()** method builds the internal data structures, like e.g. a list of interfaces or MPLS tunnels.
- // **4** - The list of **Tunnel** objects representing MPLS tunnels is extracted from the **router** object.
- // **5** - The **Iterator** object is created. It simplifies programming a loop over the tunnels, see // **8**.
- // **6** and // **7** - The system name is extracted from the **router** object and printed.
- // **8** - The loop over the **Tunnel** objects.
- // **9** - The **Tunnel** object is extracted.
- // **10**, // **11** and // **12** - The source and destination address for the current tunnel are extracted and printed.
- // **13** - The thread waits for 15 s.
- // **14** - The **catch()** function processes possible `InterruptedException` exceptions which can be thrown by the **Thread.sleep()** function.

The execution of the `TestRouter` program results in the following output. Next to the messages generated by the `MIB` constructor one can see the lines which are printed inside the program's infinite loop. During the first iteration the requested data are still not mapped from the MIB database to the `MIB` object, what is indicated by the `"??"` string and the empty list of tunnels (tunnel information is not printed at all). Only the next iterations bring the more informative printouts: the name of the router and the end points of the tunnels.

```
user@zamxyz:~/zam/rcontrol> java TestRouter
MIB: MIB()SnmpContext[host=192.168.34.9, port=161, socketType=Standard,
community=xxx]

=======================
Router: ??
=======================
Router: zam578
Tunnel from 10.10.10.78 to 10.10.10.76
Tunnel from 10.10.10.78 to 10.10.10.77
=======================
Router: zam578
Tunnel from 10.10.10.78 to 10.10.10.76
Tunnel from 10.10.10.78 to 10.10.10.77
(...)
```

8.3.3 Executing arbitrary commands - the `Router` object

This subsection presents a way of execution *Command-Line Interface* commands by means of the `execCommand2` and `execCommand3` methods of the `Router` class. They give access to the *EXEC* and *CONFIGURATION* modes of the CLI respectively. Below is the printout of an example code. The program prints the status of the `FastEthernet 0/0` interface, shuts it down, prints the status again and finally restores the interface from the shutdown state.

```
import fzjuelich.testbed.*;                                              //  0

class TestRouter2{
  static public void main(String[] args){

    MIB mib = new MIB("192.168.32.241", 161, "xxx", "Standard", true);//  1

    Router router = new Router(mib);                                     //  2

    router.setCLIPassword1("yyy");                                       //  3
    router.setCLIPassword1("zzz");

    System.out.println("");
    System.out.println(".........................step_1...");
    String out1 = router.execCommand2("show interfaces " +               //  4
                                      "FastEthernet 0/0 description");
    System.out.println(out1);

    System.out.println("");
    System.out.println(".........................step_2...");
    String out2 = router.execCommand3("interface FastEthernet 0/0",     //  5
                                          "shutdown");
    System.out.println(out2);

    System.out.println("");
    System.out.println(".........................step_3...");
    String out3 = router.execCommand2("show interfaces " +               //  6
                                      "FastEthernet 0/0 description");
    System.out.println(out3);

    System.out.println("");
    System.out.println(".........................step_4...");         //  7
    String out4 = router.execCommand3("interface FastEthernet 0/0",
                                          "no shutdown");
    System.out.println(out4);
  }
}
```

The course of the program is analyzed below. The most important lines are commented.

- `// 0` - The **import** command specifies the **fzjuelich.testbed** package containing the *Network Class Library*.
- `// 1` - The `MIB` object is created. Its constructor needs 5 arguments:
  - **"192.168.32.241"** is the IP address of the router, where a MIB database is maintained.
  - **161** is the IP port number for SNMP communication.
  - **"xxx"** is the community string, necessary when accessing the MIB database via SNMP.
  - **"Standard"** is the string defining the MIB access type.
  - **true** informs that this object will monitor SNMP traps.
- `// 2` - The **Router** object representing the router is created. Its constructor takes the **mib** object as an argument.
- `// 3` - The two passwords for CLI communication are set.

- `// 4` and `// 5` - The **execCommand2()** method executes a command in the *EXEC* mode of the *Command-Line Interface*. This particular command prints the status of the `FastEthernet 0/0` network interface.
- `// 6` and `// 7` - The **execCommand3()** method executes a command in the *CONFIGURATION* mode of the *Command-Line Interface*. The commands from the discussed example cause that the `FastEthernet 0/0` network interface is administratively shut down or reactivated respectively.

The result of the program execution is seen below. The first four lines come from the constructor of the `MIB` object. Then a history of the communication with the router is printed for every step.

In the first step the command `"show interfaces FastEthernet 0/0 description"` is executed and the text describing the state of the interface is printed. One can see that the `FastEthernet 0/0` interface (`Fa0/0`) is currently active (`Status: up`) and the software process handling the line protocol of this interface confirms that the interface is usable (`Protocol description: up`).

In the second step the `FastEthernet 0/0` interface is administratively shut down. One can verify it by looking at the printout of the third step: the information `Status: admin down` indicates that the `shutdown` command was executed on this interface (in the third step) and `Protocol description: down` means that the line protocol concerns the interface as not usable.

In the last step the `shutdown` command is removed from the `FastEthernet 0/0` configuration.

```
user@zamxyz:~/zam/rcontrol> java TestRouter2
MIB: MIB() SnmpContext[host=192.168.32.241, port=161, socketType=Standard,
community=xxx]


.........................step_1...

zam576 line 3


User Access Verification

Password:
zam576>enable
Password:
zam576#terminal length 0
zam576#show interfaces FastEthernet 0/0 description
Interface Status         Protocol Description
Fa0/0     up             up
zam576#

.........................step_2...

zam576 line 4


User Access Verification

Password:
zam576>enable
Password:
zam576#terminal length 0
```

```
zam576#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
zam576(config)#interface FastEthernet 0/0
zam576(config-if)#shutdown
zam576(config-if)#

.........................step_3...

zam576 line 3


User Access Verification

Password:
zam576>enable
Password:
zam576#terminal length 0
zam576#show interfaces FastEthernet 0/0 description
Interface Status         Protocol Description
Fa0/0     admin down     down
zam576#

.........................step_4...

zam576 line 4


User Access Verification

Password:
zam576>enable
Password:
zam576#terminal length 0
zam576#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
zam576(config)#interface FastEthernet 0/0
zam576(config-if)#no shutdown
zam576(config-if)#
```

8.3.4 Network topology information - the `Testbed` object

The program discussed in this subsection uses the `Testbed` object to extract network topology information which is then printed to the standard output. Note that the format of this output can be immediately used by the modified GARA version.

```
import fzjuelich.testbed.*;                                          //  0
import java.lang.Thread;
import java.util.*;

class TestTestbed{

  public static void main(String[] args){

  Testbed tb = new Testbed("192.168.32.241", "xxx", "yyy", "zzz");    //  1
```

```
while(true){
  try{
    Thread.sleep(1000);                                          //  2
  }catch(InterruptedException e) {
    System.out.println("Exception");
  };
  tb.update();                                                   //  3
  System.out.println("=========================================");
  System.out.println("# Routers");

  List routerList = tb.getRouter_c();                            //  4
  synchronized(routerList){                                      //  5
    Iterator it = routerList.iterator();                         //  6
    while(it.hasNext()){                                         //  7

      Router rr = (Router)it.next();                             //  8

      System.out.println("Router \t" + rr.getSysName());         //  9
      System.out.println("RouterLogin \t" + "yyy zzz");

      List interfList = rr.getInterf_c();                        // 10
      synchronized(interfList){                                  // 11
        Iterator it2 = interfList.iterator();                    // 12

        while(it2.hasNext()){                                    // 13
          Interf interf = (Interf)it2.next();                    // 14
          String ip = interf.getIpAdEntAddr();                   // 15
          if(!ip.equals("???")){

            if(interf.getIfOperStatus().equals("1")){            // 16

              System.out.println("Interface \t" +                // 17
                                 interf.getIfDescr() + " " +
                                 interf.getIpAdEntAddr() + "/" +
                                 MIB.mask(interf.getIpAdEntNetMask()));
              String speedStr = interf.getIfSpeed();             // 18
              long speed = 0;
              if(speedStr!=null)
                speed = (Long.valueOf(speedStr)).longValue();
              else
                speed = -1;
              System.out.println("InterfaceSpeed \t"
                                 + speed/(1000*1000) + "M");     // 19

            };
          };
        };
      };
      System.out.println("");
    };
  };

  List internalLinkList = tb.getInternalLink_c();                // 20
```

```
    synchronized(internalLinkList){                                    // 21
      Iterator it = internalLinkList.iterator();                       // 22
      System.out.println("");
      System.out.println("# Links between routers");
      while(it.hasNext()){                                             // 23
        Router2Router r2r = (Router2Router)it.next();                  // 24
        Interf srcInterf = r2r.getSrcInterf();                         // 25
        Interf dstInterf = r2r.getDstInterf();                         // 26

        // if both interfaces up
        if(srcInterf.getIfOperStatus().equals("1") &&                  // 27
          dstInterf.getIfOperStatus().equals("1")){

          System.out.print("Link ");
          System.out.print(r2r.getSrc().getSysName() + " " +           // 28
                           r2r.getDst().getSysName());
          String speedStr = r2r.getSrcInterf().getIfSpeed();           // 29
          long speed = 0;
          if(speedStr!=null)
            speed = (Long.valueOf(speedStr)).longValue();
          else
            speed = -1;
          System.out.println(" " + speed/(1000*1000) + "M");           // 30

        }
      }
    }
  }
}
```

The program is analyzed below and the most relevant lines of the code are commented.

- // **0** - The **import** commands specify the packages which should be loaded. One of them is the **fzjuelich.testbed** package containing the *Network Class Library*.
- // **1** - The **Testbed** object is created. Its constructor takes two arguments: **"192.168.32.241"** is the IP address of one of the routers in the testbed and **"xxx"** is the community string which is used for authentication in SNMP connections. **"yyy"** and **"zzz "** are the two passwords for CLI communication.
- // **2** - Every iteration in the infinite loop starts with the **Thread.sleep(1000)** command which stops the execution of the current thread for 1 *s*.
- // **3** - The **update()** method causes that the **tb** object is refreshed with the current content of the `MIB` objects.
- // **4** - The **routerList** object represents a list of **Router** objects in the testbed.
- // **5** - This synchronization guarantees that only one thread is allowed to work with the list at any particular time.
- // **6** and // **7** - The **Iterator** object extracted from the **routerList** helps to program a loop over **routerList** elements.
- // **8** - **List** objects work only with **Object** elements. Since the elements are in fact **Router** objects, the explicit type projection operator **(Router)** must be applied.

- // **9** - The router's name and the passwords for the CLI communication are printed.
- // **10** - The **interfList** object represents a list of **Interf** objects assigned to the **rr** object.
- // **11** - This synchronization guarantees that only one thread is allowed to work with the list at any particular time.
- // **12** and // **13** - The **Iterator** object extracted from the **interfList** helps to program a loop over **interfList** elements.
- // **14** - **List** objects work only with **Object** elements. Since the elements are in fact **Interf** objects the explicit type projection operator **(Interf)** must be applied.
- // **15** - The IP address of the interface is extracted.
- // **16** - This block is executed only if the interface is active.
- // **17** - Printout of the interface name, IP address and the mask length (number of 1-bits in the mask).
- // **18** - **speedStr** is a link speed in bits per second.
- // **19** - Printout of the link speed in megabits per second.
- // **20** - The **internalLinkList** object represents a list of internal links (**Router2Router** objects) in the testbed.
- // **21** - This synchronization guarantees that only one thread is allowed to work with the list at any particular time.
- // **22** and // **23** - The **Iterator** object extracted from the **internalLinkList** helps to program a loop over **internalLinkList** elements.
- // **24** - **List** objects work only with **Object** elements. Since the elements are in fact **Router2Router** objects the explicit type projection operator **(Router2Router)** must be applied.
- // **25** and // **26** - Source and target interface objects are extracted: **srcInterf** and **dstInterf**.
- // **27** - The condition is checked whether the source and target interfaces are both active.
- // **28** - Printout of the source and target router names.
- // **29** - Extracting the speed of the source interface in bits per second.
- // **30** - Printout of the speed of the source interface in mega bits per second.

The course of the `TestTestbed` program is analyzed below. At first constructors of `MIB` objects send information about addresses of the contacted interfaces, IP port numbers used, socket types and community strings for SNMP communication. There are four routers in the testbed thus five `MIB` object are created, compare with Figure 8.4.

Next, the router sections (`# Routers`) include some router parameters like community strings (`RouterLogin`), interface names, addresses and speeds. Next, the list of links (`# Links between routers`) contains the basic link parameters: the name of the source router, target routers and their speed in mega bits per second.

```
PAB@zam233:~/zam/rcontrol> java TestTestbed
MIB: MIB() SnmpContext[host=192.168.32.241, port=161, socketType=Standard,
community=xxx]
MIB: MIB() SnmpContext[host=192.168.32.241, port=161, socketType=Standard,
community=xxx]
MIB: MIB() SnmpContext[host=192.168.32.242, port=161, socketType=Standard,
community=xxx]
MIB: MIB() SnmpContext[host=192.168.32.233, port=161, socketType=Standard,
community=xxx]
MIB: MIB() SnmpContext[host=192.168.32.234, port=161, socketType=Standard,
community=xxx]
==========================================
# Routers
Router  zam576
RouterLogin     yyy zzz
Interface       FastEthernet0/1 192.168.32.241/30
InterfaceSpeed  100M
Interface       Loopback0 10.10.10.76/32
InterfaceSpeed  4294M
Interface       ATM1/0.1-aal5 layer 192.168.32.9/30
InterfaceSpeed  149M

Router  zam577
RouterLogin     yyy zzz
Interface       FastEthernet0/0 192.168.32.242/30
InterfaceSpeed  100M
Interface       FastEthernet0/1 192.168.35.9/29
InterfaceSpeed  100M
Interface       Loopback0 10.10.10.77/32
InterfaceSpeed  4294M
Interface       ATM1/0.1-aal5 layer 192.168.32.17/30
InterfaceSpeed  149M
Interface       ATM4/0.1-aal5 layer 192.168.32.25/30
InterfaceSpeed  149M

Router  zam578
RouterLogin     yyy zzz
Interface       POS3/0 192.168.32.33/30
InterfaceSpeed  155M
Interface       FastEthernet0/0 192.168.34.9/30
InterfaceSpeed  100M
Interface       FastEthernet0/1 134.94.173.106/21
InterfaceSpeed  100M
Interface       GigabitEthernet2/0 192.168.32.233/30
InterfaceSpeed  1000M
Interface       Loopback0 10.10.10.78/32
InterfaceSpeed  4294M
Interface       ATM1/0.1-aal5 layer 192.168.32.10/30
InterfaceSpeed  149M
Interface       ATM4/0.1-aal5 layer 192.168.32.18/30
InterfaceSpeed  149M

Router  zam579
RouterLogin     yyy zzz
```

```
Interface       POS3/0 192.168.32.34/30
InterfaceSpeed  155M
Interface       FastEthernet0/1 192.168.33.17/30
InterfaceSpeed  100M
Interface       GigabitEthernet2/0 192.168.32.234/30
InterfaceSpeed  1000M
Interface       Loopback0 10.10.10.79/32
InterfaceSpeed  4294M


# Links between routers
Link zam576 zam577 100M
Link zam576 zam578 149M
Link zam577 zam576 100M
Link zam577 zam578 149M
Link zam578 zam576 149M
Link zam578 zam577 149M
Link zam578 zam579 155M
Link zam578 zam579 1000M
Link zam579 zam578 155M
Link zam579 zam578 1000M
(...)
```

8.3.5 Support for SNMPv1 and SNMPv3

SNMP, first introduced in the late eighties, quickly became the most widely-used network management protocol in TCP/IP based networks. In its initial version, named SNMPv1, it has some deficiencies of security mechanisms. Here, authentication is based on a single password, known as a *community string* which is included in SNMP packets so a potential intruder can get it by monitoring network traffic, see Figure 8.5

The lack of more advanced security features in SNMPv1 resulted in intensive works on the next version of the protocol: SNMPv2. However, there was no agreement about how security should be assured. The two competing solutions were called SNMPv2u and SNMPv2*. Finally, the currently most advanced and general version SNMPv3 was launched.

All examples discussed so far used the simplest, version 1 of SNMP, however, the presented software supports both version 1 and 3. The choice is determined by the constructors used. Thus, the `MIB` object can be initialized in two ways:

- ```
  MIB(String _host,
      int _port,
      String _community,
      String _socketType,
      boolean _useTrapListener)
  ```

  - The constructor for SNMPv1, where `_host` is the IP address of the SNMP agent to be contacted, `_port` is the UDP port to be used (usually `161`), `_community` is a kind of password in the SNMPv1 security model, `_socketType` determines the kind of socket used (usually `Standard`), and the logical variable `_useTrapListener` can activate trap listening functionality of the `MIB` object.

- ```
  MIB(String host,
      int port,
      String _socketType,
      boolean _useTrapListener,
      String _contextName,
  ```

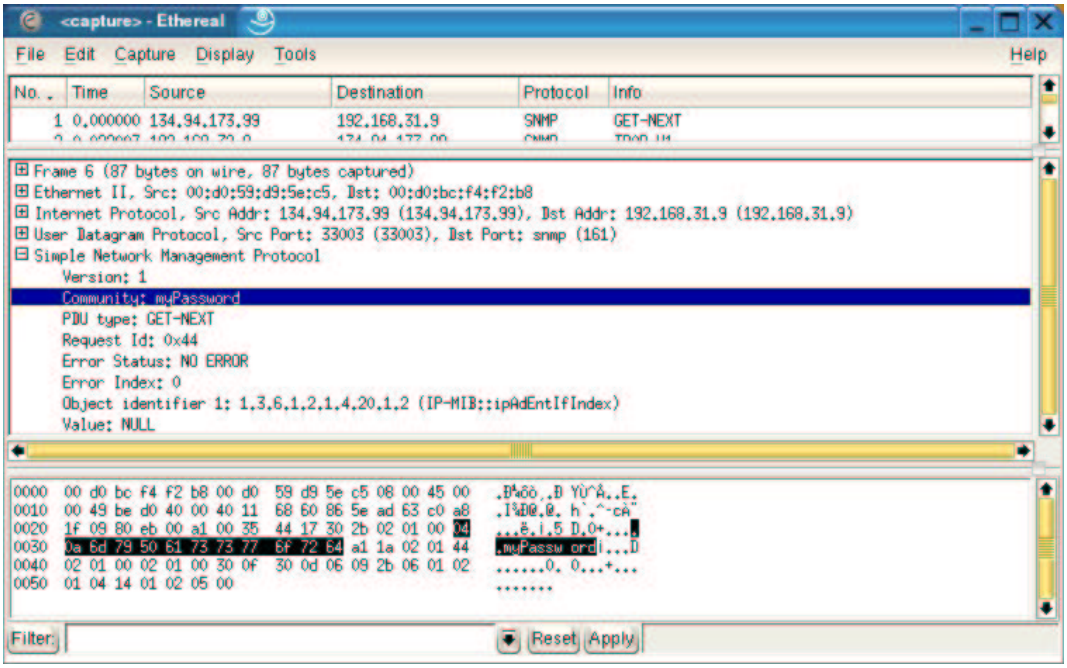

Figure 8.5: The screen dump from the *ethereal* program used to monitor network traffic. The SNMPv1 *community string* can be easily identified as `myPassword` string.

```
String _userName,
int _authenticationProtocol,
String _userAuthenticationPassword,
String _userPrivacyPassword,
boolean _useAuthentication,
boolean _usePrivacy)
```

- The constructor for SNMPv3, where the parameters `_host`, `_port`, `_socketType` and `_useTrapListener` have the same meaning as for the SNMPv1 `MIB` constructor. Additionally, `_contextName` defines the SNMPv3 context name (usually an empty string), `_userName` should be identical with the user name defined during SNMPv3 configuration of the Cisco router, `_authenticationProtocol` chooses a protocol used by authentication procedures, `_userAuthenticationPassword` and `_userPrivacyPassword` passwords are used to generate secret keys used by authentication and encryption procedures, logical variables `_useAuthentication` and `_usePrivacy` are used to choose an appropriate security mechanisms which should be applied.

Also the `Testbed` object, which uses internally `MIB` objects has two constructors:

- ```
  Testbed(String _routerAddress0,
          String _community,
          String _pass1,
          String _pass2)
  ```

  - the constructor for SNMPv1, where `_routerAddress0` is any valid IP address of any router in the network domain represented by `Testbed` object, `_community` is a *community string* used by SNMPv1 to communicate with all routers in the domain, `_pass1` and `_pass2` are two passwords giving access to the router via the CLI interface.

- ```
  Testbed(String _routerAddress0,
  ```

```
String _contextName,
String _username,
int _authenticationProtocol,
String _userAuthenticationPassword,
String _userPrivacyPassword,
boolean _useAuthentication,
boolean _usePrivacy,
String pass1,
String pass2)
```

- the constructor for SNMPv3, where `_routerAddress0`, `_pass1` and `_pass2` parameters have the same meaning as for the SNMPv1 `Testbed` constructor. The meaning of the other parameters: `_contextName`, `_username`, `_authenticationProtocol`, `_userAuthenticationPassword`, `_userPrivacyPassword`, `_useAuthentication` and `_usePrivacy` is the same as for the SNMPv3 `MIB` constructor described above in this subsection.

8.4 Network Controlling Application

The *Network Class Library* offers building blocks for creating higher level libraries or applications. A `Display` program is an example of an application built on the top of the *Network Class Library*. It can be used for:

- visualization of the network topology,
- presentation of the current status and configuration of routers,
- administration of MPLS tunnels,
- configuration of the *Differentiated* environment.

The `Display` program must be invoked with two parameters: an IP address of the access interface and a string identifying the version of SNMP to be used (`v1` for SNMPv1 or `v3` for SNMPv3). The program's main window contains a set of radio buttons for selecting type of display. The **MPLS tunnels** button is used to display a list of tunnels, a text with their basic parameters and a panel with the graphical representation of the network topology including tunnels' courses. By selecting other radio buttons placed in the same raw one invokes panels with different content. All of them contain a text field with an appropriate information depending on the section chosen. Thus, the **Router configurations** panel allows to retrieve router configurations, the **Trap messages** panel reports received SNMP trap messages, the **Logs** panel contains the history of the *Command-Line Interface* communication with routers and the **MIBs** panel allows to monitor the content of MIB databases installed on different routers.

Chapter 9

Conclusions

The provision of well-defined QoS guarantees in IP-based backbone networks is a challenge, particularly with respect to scalability and manageability. The PAB-project successfully addressed this issue proposing scalable service provisioning mechanisms together with advanced network management functions.

The MPLS architecture offers the basis for advanced traffic engineering mechanisms that extend the concepts of the Differentiated Services architecture. The conducted experimental evaluation can be used as a basis for deploying this technology. However, the project revealed a significant complexity that needs to be solved by appropriate operational tools.

Our Java-based management framework offers a first starting point to address this issue. A particular benefit of the implementation is that it offers opportunities for integration into future Grid environments that implement the emerging Open Grid Services Architecture (OGSA) standard. While the project successfully demonstrated the integration of on-demand services in current Grid environments the migration towards OGSA has been identified as a relevant future research and development target.

In addition, the PAB-project developed a new analytical approach to calculate hard service guarantees that can be used in Service Level Agreements. The implemented framework facilitates admission control procedures that are similar to those applied by the Integrated Services model, however, taking scalable aggregate scheduling into account.

In proposing solutions for managing service guarantees in IP-based networks, the outcomes of the PAB-project can be used as a building block for future network environments. The applicability of MPLS concepts also in optical networks fosters this.

Acknowledgements

We gratefully appreciate the support we have received from various colleagues, particularly from: Sascha Dröttboom, Gerrit Einhoff, Dietmar Erwin, Ian Foster, Friedel Hoßfeld, Frank Imhoff, Thomas Lippert, Olaf Mextorf, Ralph Niederberger, Alain Roy, Otto Spaniol, Michael Wallbaum, Sabine Werner, Linda Winkler, and Cisco Systems.

Bibliography

[1] Adler, H.-M., *10 Gigabit/s Plattform für das G-WiN betriebsbereit*, DFN Mitteilungen, Heft 60, November 2002.

[2] Allcock, B. et al., *Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing*, Proceedings of IEEE Mass Storage Conference, 2001.

[3] Allman, M., Paxson, V., and Stevens, W., *TCP Congestion Control*, RFC 2581, 1997.

[4] Almes, G., Kalidini, S., Zekauskas, M., *A One-way Delay Metric for IPPM*, RFC 2679, 1999.

[5] Andersson, L. et al., *LDP Specification*, RFC 3036, 2001.

[6] Awduche, D. et al., *Requirements for Traffic Engineering Over MPLS*, RFC 2702, 1999.

[7] Awduche, D. et al., *RSVP-TE: Extensions to RSVP for LSP Tunnels*, RFC 3209, 2001.

[8] Awduche, D. et al., *Applicability Statement for Extensions to RSVP for LSP-Tunnels*, RFC 3210, 2001.

[9] Bajaj, S., Breslau, L., Shenker, S., *Uniform versus Priority Dropping for Layered Video*, Proceedings of ACM SIGCOMM, 1998.

[10] Blake, S., Black, D., Carlson, M., Davies, M., Wang, Z., Weiss, W., *An architecture for Differentiated Services*, RFC 2475, 1998.

[11] Braden, R. et al., *Resource ReSerVation Protocol (RSVP)*, RFC 2205, 1997.

[12] Cardwell, N., Savage, S., and Anderson, T., *Modeling TCP Latency*, Proceedings of IEEE Infocom, 2000.

[13] Chang, C.-S., *Performance Guarantees in Communication Networks*, Springer, TNCS, 2000.

[14] Charny, A., and Le Boudec, J.-Y., *Delay Bounds in a Network with Aggregate Scheduling*, Springer, LNCS 1922, Proceedings of QofIS, 2000.

[15] Charny, A. et al., *Supplemental Information for the New Definition of EF PHB (Expedited Forwarding Per-Hop-Behavior)*, RFC 3247, 2002.

[16] Chimento, P.F., et al., *QBONE Signalling Design Team: Final Report* http://qos.internet2.edu/wg/documents-informational/20020709-chimento-etal-qbone-signaling/, 2001.

[17] Côté, G., Erol, B. and Gallant, M., *H.263+: Video Coding at Low Bit Rates*, IEEE Transactions on Circuits and Systems for Video Technology vol. 8, no. 7, 1998.

[18] Cruz, R. L., *A Calculus for Network Delay, Part I: Network Elements in Isolation*, IEEE Transactions on Information Theory, vol. 37, no. 1, pp. 114-131, 1991.

[19] Cruz, R. L., *A Calculus for Network Delay, Part II: Network Analysis*, IEEE Transactions on Information Theory, vol. 37, no. 1, pp. 132-141, 1991.

[20] Cruz, R. L., *SCED+: Efficient Management of Quality of Service Guarantees*, Proceedings of IEEE Infocom, 1998.

[21] Davie, B., et al., *An Expedited Forwarding PHB* RFC 3246, 2002.

[22] DeFanti, T. and Stevens, R., *Teleimmersion*, in Foster, I. and Kesselman, C., *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan-Kaufmann, 1998.

[23] Demichelis, C. and Chimento, P., *IP Packet Delay Variation Metric for IPPM*, RFC 3393, 2002.

[24] Ferrari, T., Pau, G. and Raffaelli, C., *Priority Queuing Applied to Expedited Forwarding: A Measurement-Based Analysis*, Proceedings of QofIS, 2000.

[25] Fidler, M., *Differentiated Services based Priority Dropping and its Application to Layered Video Streams*, Proccedings of IFIP-TC6 Networking, 2002.

[26] Fidler, M. and Einhoff, G., *Routing in Turn-Prohibition based Feed-Forward Networks*, Proceedings of IFIP-TC6 Networking, Springer, LNCS 3042, pp. 1168-1179, 2004.

[27] Fidler, M. and Sander, V., *A Parameter Based Admission Control for Differentiated Services Networks*, Elsevier Computer Networks Journal, vol. 44, no. 4, pp. 463-479, 2004.

[28] Fitzek, F. and Reisslein, M., *MPEG–4 and H.263 Video Traces for Network Performance Evaluation*, IEEE Network, vol. 15, no. 6, pp. 40-54, 2001.

[29] Floyd, S., Jacobson, V., *Random Early Detection Gateways for Congestion Avoidance*, IEEE/ACM Transactions on Networking, vol. 1, no. 4, pp. 397-413, 1993.

[30] Foster, I. et al., *A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation*, Proceedings of IWQoS, 1999.

[31] Foster, I., Fidler, M., Roy, A., Sander, V. and Winkler, L., *End-to-End Quality of Service for High-End Applications*, Elsevier Computer Communications Journal, vol. 27, no. 14, pp. 1375-1388, 2004.

[32] Foster, I., Roy, A. and Sander, V., *Quality of Service Architecture that Combines Resource Reservation and Application Adaptation*, Proceedings of IWQoS, 2000.

[33] Fumagalli, A. and Valcerenghi, L., *IP Restoration vs. WDM Protection: Is There an Optimal Choice?*, IEEE Network, vol. 14, no. 6, pp. 34-41, November-December, 2000.

[34] Grossman, D., *New Terminology and Clarifications for Diffserv*, RFC 3260, 2002.

[35] Guo, Y., Kuipers, F. and Van Mieghem P., *Link-disjoint paths for reliable QoS routing*, Internationl Journal of Communication Systems, vol. 16, pp. 779-798, 2003.

[36] Hasegawa, G., Murata, M., and Miyahara, M. *Performance Evaluation of HTTP/TCP on Asymmetric Networks*, International Journal of Communication Systems, vol. 12, no. 4, pp. 281-296, 1999.

[37] Haverkort, B. R., *Performance of Computer Communication Systems: A Model-Based Approach*, Wiley, 1998.

[38] Heinanen, J. et al., *Assured Forwarding PHB Group*, RFC 2597, 1999.

[39] Hoffmann, G., *G-WiN - the Gbit/s Infrastructure for the German Scientific Community*, Proceedings of TNC, p. 1, 2001.

[40] Hurley, P., Le Boudec, J.Y., Thiran, P., Kara, M., *ABE: Providing a Low-Delay Service within Best Effort*, IEEE Network Magazine, May/June, 2001.

[41] Jamoussi, B. et al., *Constraint-Based LSP Setup using LDP*, RFC 3212, 2002.

[42] Kar, K., Kodialam, M. and Lakshman, T.V., *Minimum Interference Routing of Bandwidth Guaranteed Tunnels with MPLS Traffic Engineering Applications*, IEEE Journal of Selected Areas in Communications, vol. 18, no. 12, pp. 2566-2579, 2000.

[43] Kodialam, M. and Lakshman, T.V., *Minimum Interference Routing with Applications to MPLS Traffic Engineering*, Proceedings of IEEE INFOCOM, pp. 884-893, 2000.

[44] Kodialam, M. and Lakshman, T.V., *Dynamic Routing of Locally Restorable Bandwidth Guaranteed Tunnels using Aggregated Link Usage Information*, Proceedings of IEEE Infocom, pp. 376-385, 2001.

[45] Law, A. M., and Kelton, W. D., *Simulation, Modeling, and Analysis*, McGraw-Hill, 3rd edition, 2000.

[46] Le Boudec, J.-Y. and Thiran, P., *Network Calculus Made Easy*, Technical Report EPFL-DI 96/218, Ecole Polytechnique Federale, Lausanne (EPFL), 1996.

[47] Le Boudec, J.-Y. and Thiran, P., *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, Springer, LNCS 2050, 2002.

[48] Li, G., Wang, D., Kalmanek, C. and Doverspike, R., *Efficient Distributed Path Selection for Shared Mesh Restoration*, IEEE/ACM Transactions on Networking, vol. 11, no. 5, pp. 761-771, 2003.

[49] Ma, Q., Steenkiste, P. and Zhang H., *Routing High-bandwidth Traffic in Max-min Fair Share Networks*, Proceedings of SIGCOMM, pp. 206-217, 1996.

[50] Ma, Q. and Steenkiste, P., *On path selection for traffic with bandwidth guarantees*, Proceedings of IEEE International Conference on Network Protocols, pp. 191-202, 1997.

[51] Mathis, M., Semke, J., Mahdavi, J. and Ott, T., *The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm*, Proceedings of ACM SIGCOMM, 1997.

[52] Melon, L., Blanchy, F. and Leduc, G., *Decentralized Local Backup LSP Calculation with Efficient Bandwidth Sharing*, Proceedings of IEEE ICT, pp. 929-937, 2003.

[53] Nichols, K. and Carpenter, B., *Definition of Differentiated Services Per Domain Behaviors and Rules for their Specification*, RFC 3086, 2001.

[54] Nichols, K., Jacobson, V. and Zhang, L., *A Two-bit Differentiated Services Architecture for the Internet*, RFC 2638, 1999.

[55] Osborne, E. and Simha A., *Traffic Engineering with MPLS*, Cisco Press, 2002.

[56] Padhye, J., Firoiu, V., Towsley, D. and Kurose, J., *Modeling TCP Troughput: A simple model and its empirical validation*, Proceedings of ACM SIGCOMM, 1998.

[57] Parekh, A. K. and Gallager, R. G., *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case*, IEEE/ACM Transactions on Networking, vol. 1, no. 3, pp. 344-357, 1993.

[58] Parekh, A. K. and Gallager, R. G., *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple-Node Case*, IEEE/ACM Transactions on Networking, vol. 2, no. 2, pp. 137-150, 1994.

[59] Rosen, E. et al., *Multiprotocol Label Switching Architecture*, RFC 3031, 2001.

[60] Rosen, E. et al., *MPLS Label Stack Encoding*, RFC 3032, 2001.

[61] Roy, R. and Sander, V., *GARA: A Uniform Quality of Service Architecture*, in *Grid Resource Management - State of the Art and Future Trends*, edited by Jarek Nabrzyski, Jennifer Schopf and Jan Weglarz, Kluwer Academic Publishers, 2003.

[62] Sander, V., Adamson, W., Foster, I. and Roy, A., *End-to-End Provision of Policy Information for Network QoS*, IEEE Symposium on High Performance Distributed Computing, 2001.

[63] Sander, V. and Fidler, M., *Evaluation of a Differentiated Services based Implementation of a Premium and an Olympic Service*, Springer, LNCS 2511, Proceedings of QofIS, 2002.

[64] Sander, V. Fidler, M., *A Pragmatic Approach for Service Provisioning Based on a Small Set of Per-Hop Behaivors*, Proceedings of IEEE ICCCN, 2002.

[65] Sander, V., Foster, I., Roy, A. and Winkler, L., *A Differentiated Services Implementation for High-Performance TCP Flows*, Terena Networking Conference, 2000.

[66] Schroeder, M. et al., *Autonet: a High-speed, Self-configuring Local Area Network Using Point-to-point Links*, IEEE Journal on Selected Areas in Communications, vol. 9, no. 8, October, 1991.

[67] Shenker, S., Patridge, C. and Guerin, R., *Specification of Guaranteed Quality of Service*, RFC 2212, 1997.

[68] Starobinski, D., Karpovsky, M. and Zakrevski, L., *Application of Network Calculus to General Topologies using Turn-Prohibition*, IEEE/ACM Transactions on Networking, vol. 11, no. 3, pp. 411-421, 2003.

[69] Taft-Plotkin, N., Bellur, B. and Ogier, R., *Quality of Service Routing Using Maximally Disjoint Paths*, Proceedings of IWQoS, 1999.

[70] Thomas, B. et al., *LDP Applicability*, RFC 3037, 2001.

[71] Trimintzios, P. et al., *An Architectural Framework for Providing QoS in IP Differentiated Services Networks*, Proceedings of IFIP/IEEE International Symposium on Integrated Network Management, 2001.

[72] Wang, S., Xuan, D., Bettati, R. and Zhao, W., *Providing Absolute Differentiated Services for Real-Time Applications in Static-Priority Scheduling Networks*, Proceedings of IEEE Infocom, 2001.

[73] Wang, Z. and Crowcroft, J., *Quality-of-Service Routing for Supporting Multimedia Applications*, IEEE Journal of Selected Areas in Communications, vol. 14, no. 7, pp. 1228-1234, 1996.

[74] Wroclawski, J., *The Use of RSVP with IETF Integrated Services*, RFC 2210, 1997.

[75] Wroclawski, J., *Specification of the Controlled-Load Network Element Service*, RFC 2211, 1997.

[76] Wu, Q. and Williamson, C., *Improving Ensemble-TCP Performance on Asymmetric Networks*, Proceedings of MASCOTS, 2001.

[77] *QBone Scavenger Service*, www.internet2.edu/qbss/.