QoS for real-time IP traffic

Graduate Thesis

Siv.ing. degree in
Information and Communication
Technology

by
Helge Gundersen and Frode Trydal

Grimstad, May 2001

Abstract

This thesis focuses on three service models, known as Best Effort, IntServ and DiffServ, and their ability of providing QoS in IP networks. The objective is to evaluate the performance of real-time traffic based on theory and testing. In addition, mechanisms that may improve the QoS performance further, like MPLS and Bandwidth Brokers, are evaluated to find a comprehensive solution for different types of IP networks.

Tests are set up to focus on typical behavior of the service models, and disclose differences in their efficiency together with problems concerning fairness in the bandwidth sharing. Results obtained show the capability of IntServ and DiffServ to supply QoS guarantees for real-time traffic in contrast to Best Effort.

Best Effort is fit for networks with low input load where traffic does not need service guarantees. IntServ reserves resources end-to-end, guarantees service quality and is suitable for small networks. DiffServ classifies traffic in behavior-aggregates and is very scalable, thus is fit for use in large networks. For the Internet, the results suggest that IntServ should be used in edge networks and DiffServ over MPLS in core networks.

Preface

This thesis is written for Ericsson's design unit in Grimstad, Norway and is a part of the Graduate degree (Siv.ing.) in Information and Communication Technology (ICT) at Agder University College.

We started the thesis January '01 by getting an overview of the technologies to be used during the process. We had to find out early how to configure our testbed and how to perform the actual testing. This had to be done to order and get the test equipment in time. Early in February we presented the preliminary study.

The preliminary study introduced a progress schedule that we followed during the rest of the thesis. We performed a literature study, configured the testbed and performed the tests. The testing took more time than presumed, since the evaluation of the results was comprehensive. We kept on studying the literature to fill some holes in the thesis, and in the end we spent a lot of time to evaluate the results.

Documentation was performed in parallel with the work during the whole process, and we have been in regular meetings with our supervisors. During the meetings we have presented results and discussed difficulties that have occurred.

Our supervisors have been Jon Mjellekås (Department Manager, Ericsson), Berner Vegge (Master Product & System Manager, Ericsson) and Magne Arild Haglund (Assistant Professor, Agder University College). They have inspired us to perform our best and reach our goal. We would like to thank them for giving us advice and guidance throughout the project.

Other persons that also have contributed to our thesis are Dr. K. Cho (the creator of the router software we have used) and Peter Frame (Senior SE, NetIQ Corporation). In addition we would like to thank the NetIQ Corporation for letting us have an extended evaluation version of the software package Chariot to perform our tests.

The report can be used to get an overview to what QoS for real-time IP traffic implies, and what mechanisms that can be used to achieve QoS in IP networks. We have learned a lot about how to approach the problem of handling real-time traffic in an IP network, and hope that the thesis can help the design unit we worked for at Ericsson to understand more about this issue.

Helge Gundersen Frode Trydal

Grimstad, Spring 2001

Contents

1 Introduction 1

1.1 Thesis introduction 1
1.2 Task description 2
1.3 Literature review 3
1.4 Report outline 4

2 A basis for evaluating real-time IP traffic 5

2.1 Overview 5
2.2 What is Quality of Service? 6
2.3 QoS on different layers of the OSI model 7
2.4 Real-time applications 10

3 The service models 12

3.1 Overview 12
3.2 QoS forwarding mechanisms 13
3.3 Best Effort 23
3.4 Integrated Services 25
3.5 Differentiated Services 30

4 Other QoS technologies 37

4.1 Overview 37
4.2 IntServ over DiffServ 38
4.3 MPLS 40
4.4 Bandwidth Broker 47

5 Testing the service models 50

5.1 Overview 50
5.2 Testbed description 51
5.3 Preliminary tests 54
5.4 The power of a network 59
5.5 Fair resource allocation 68

6 QoS for different network topologies 76

6.1 Overview 76
6.2 A small-scale network 77
6.3 A mobile network 79
6.4 A large backbone 84

7 Discussion 86

7.1 Overview 86
7.2 Preliminary discussion 87
7.3 Best Effort 88
7.4 Integrated Services 89
7.5 Differentiated Services 90
7.6 Future work 91

8 Conclusion 92

Abbreviations 93

References 95

Appendixes 99

1 Introduction

1.1 Thesis introduction

With the increased processing capability of personal computers and the appearance of new multimedia applications, the Internet experiences a tremendous growth. The traffic has not only increased in volume, it has also changed in nature, due to large amounts of multimedia traffic. The Best Effort model, adopted by the Internet for the transport of information, does not support the real-time requirements that these new applications require. Proposals are being studied to find a way to define different service levels, i.e. services on the Internet with a specific Quality of Service (QoS).

The Integrated Services (IntServ) [1] approach provides service discrimination through the explicit allocation and reservation of resources. This model requires a large amount of signaling between nodes and maintains state information at each node, for individual flows.

The Differentiated Services (DiffServ) [2] approach intends to define a simple group of mechanisms to treat packets with different priorities according to the information carried in the DSCP field of the IP packet header. This approach requires neither state information per flow nor signaling to each node, which increases the scalability potential.

A detailed and thorough evaluation of the service level offered by these mechanisms needs to be accomplished prior to their deployment. In addition, mechanisms that may improve weaknesses, such as Multi-Protocol Label Switching (MPLS) [3], Bandwidth Brokers [4], or a combination of IntServ and DiffServ [5], must be considered.

The objective of this thesis is to evaluate the performance of real-time flows in IP networks that implement the service models.

1.2 Task description

Title: QoS for real-time IP traffic

A new breed of applications, including audio and video streaming, demand high data throughput capacity (bandwidth) and have low-latency requirements when used in two-way communications (i.e. conferencing and telephony). The Internet Engineering Task Force (IETF) has a lot of promising technologies under development to provide increased Quality of Service of the Internet Protocol and its infrastructure. The thesis shall evaluate different technologies that provide QoS in IP networks, with focus on real-time application behavior.

A series of models have been proposed to provide different types of QoS in IP networks, and at least the following service models have to be evaluated: Best Effort, Integrated Services and Differentiated Services.

Compare the use of the service models for real-time IP traffic, especially IP telephony. In addition, try to find other QoS technologies that may be combined with the service models.

Test the service models by using an IP testbed. Load the testbed with IP traffic and measure how well real-time traffic is transferred using different service models.

Make a proposal, based on theory and the testbed results, to what kind of QoS technologies that are appropriate for at least one network topology.

1.3 Literature review

This section explains where information on the Internet that is relevant for this thesis can be found. The most commonly used sites are mentioned first.

The most interesting site must be the Internet Engineering Task Force (IETF) [6]. The IETF is a large open international community of network designers, operators, vendors and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet. It is open to any interested individual, and we can find “Requests For Comments” (RFCs) and drafts for new Internet standards at the site.

The MPLS Resource Center [7] was founded in January 2000 to provide a clearinghouse for information on the IETF's Multiprotocol Label Switching Standard. The MPLS Resource Center is owned and operated by ITPRC.COM [8] and has neither relation to the Internet Engineering Task Force nor any hardware vendor. As we will see during the thesis, MPLS render possibilities such as traffic engineering on the Internet.

Internet2 [9] is another interesting site being led by over 180 universities working in partnership with industry and government to develop and deploy advanced network applications and technologies, accelerating the creation of tomorrow's Internet. Internet2 is recreating the partnership among academia, industry and government that fostered today’s Internet in its childhood. The QoS working group [10] and the Qbone (a testbed of the Bandwidth Broker) [11] is of particular interest concerning this thesis.

Recently written articles are considered more valid because of the rapid development of new Internet communication technologies.

1.4 Report outline

We make one assumption to the task description of this thesis. As IP telephony is a part of real-time traffic, we choose to focus on real-time traffic as a whole. The term real-time traffic includes applications such as videoconferences, IP telephony and streaming of different types. We will not interpret the task description further, instead we elaborate how we structure the thesis to answer the task description in the best possible way.

The target group for this report is students and network engineers with basic knowledge of IP networks. Readers with interest in QoS in IP networks and the development of the Internet in the future may benefit from reading this report.

First, chapter 2 is background information to introduce the term Quality of Service and describe why we look at QoS at the network layer. Then we explain the nature of real-time applications and what demands they have for an IP network.

Chapter 3 and 4 of the thesis is theoretical parts where we give basic knowledge on QoS technologies and discuss their behavior. We try not to go too deep into the substance, since there is no reason making this a textbook in QoS technologies. At the same time, we must gain sufficient knowledge to understand the argumentation in the discussions following each new technology introduced.

The next chapter is a practical part where we build a testbed to look at some of the features described in the theoretical part of this document. We measure how well real-time traffic is transferred using different service models in a small-scale testbed. The results found from the testbed are presented to offer the possibility of making own conclusions from the results.

In chapter 6, we make a proposal to what QoS mechanisms that are appropriate for three different network topologies. We focus on real-time IP traffic, and IP telephony is a very important part of this kind of traffic. The suggestions are based on experiences gathered from the three former chapters described.

Finally, we discuss and suggest an overall solution for IP networks and the Internet where different service models and QoS mechanisms play together to give real-time traffic the best possible treatment.

2 A basis for evaluating real-time IP traffic

2.1 Overview

To make an introduction to Quality of Service (QoS), section 2.2 has an explanation of the term QoS, a short introduction to the three service models and a description of factors that make it possible to measure QoS.

QoS mechanisms may be introduced on different layers of the OSI reference model. The reason why this report focuses on QoS at the IP layer is explained in section 2.3 by discussing the QoS features at each OSI layer. Traffic management mechanisms such as ATM, Frame Relay and IEEE 802.1p are discussed.

Section 2.4 characterizes real-time applications and the requirements such applications have to networks.

2.2 What is Quality of Service?

Quality of Service (QoS) is the quality a user or customer can expect from a given service. In this report, we think of QoS as the quality a service experiences over an IP network. When specifying the QoS, a number of factors are taken into account:

- Latency - the time from a packet is sent until it is received at another point. Response time is another term concerning latency, and refers to the round-trip time, i.e. twice the latency. For IP telephony, this is a very important factor.
- Jitter (timing jitter)– timing variations from an ideal position in time, caused by packets arriving either out of order or at an inconsistent rate. This is particularly damaging in multimedia applications where timing inconsistencies of the data may be viewable as shaky images in real-time video applications.
- Packet Loss - the percentage of packets lost in the transmission. Different applications will have different tolerance of packet loss.
- Throughput - the amount of data transferred between two given nodes during a given amount of time. This reflects the bandwidth of the network and is a significant factor to QoS for e.g. videoconferences.

Quantifying the above parameters allows us to find out how efficiently the traffic in an IP network is being managed and whether the network is suitable for the data we wish to transmit or not. Different kinds of applications have different requirements for the parameters listed above.

There are primarily three possible QoS architectures, referred to as service models in this report:

- Best Effort can only provide QoS by over-provisioning the network. If there were infinite bandwidth available for everyone to use all the time, there would be no problem with any type of communication over IP. Obviously this is not possible, and the closest we can get would be to provide excess capacity at points in the network that are frequently busy, or to add bandwidth to a section that becomes busy at a given time. The restricting factor here is cost.

- The Integrated Services Architecture (IntServ), also referred to only as resource reservation, allocates network resources according to a QoS request from a user. The resources remain allocated for the duration of the transmission, and will not be affected by normal Best Effort IP traffic. Real-time traffic like voice and video can use resource allocation to make sure it gets the service needed. RSVP is the name of the reservation protocol initially made for use with IntServ, but it has also been utilized in other signaling contexts.

- A Differentiated Services (DiffServ) network provides Quality of Service for groups of microflows, called behavior-aggregates. A bit-pattern in each packet is used to mark a packet in order to receive a particular forwarding treatment, or per-hop behavior, at each network node. The intelligence is in the edge nodes that mark the packets, while the core nodes only forward the packets based on the marked bit-pattern. A common understanding about the use and interpretation of this bit-pattern is required for consistent reasoning about expected aggregate behaviors in a network. Preferential treatment is given to applications that are specified as more demanding.

Most networks tend to combine the above protocols to implement the best performance, and they have been designed such that no architecture is given exclusive control of the network.

2.3 QoS on different layers of the OSI model

2.3.1 The OSI model

We start looking at Figure 1, the OSI reference model [12], to visualize where we can give service guarantees in a network. QoS mechanisms may be introduced in the network layers of the OSI reference model: the physical layer, the data link layer, and the network layer. In addition, end-user functions like the transport protocols may improve network performance.

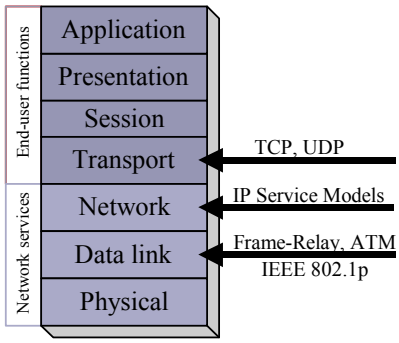

Figure 1. QoS on different layers of the OSI model.

2.3.2 Physical layer

The physical layer is the transmission media in the network usually consisting of physical wiring or fiber optics. Diverse paths may be a method for providing increased service quality at this layer. If one path through a network is congested, it is a good idea to build another path if increasing the capacity of the existing one means high costs. However, sharing an input load between two diverse paths across a network can in certain circumstances lead to decreased performance.

Take an example where some arbitrary amount of network traffic takes the primary low-delay, high-bandwidth path, and the bulk of traffic takes another path, which may have different delay and bandwidth properties. Such a configuration may cause packets sent from the same application, but in different paths, to arrive in the wrong order. This may lead to increased jitter within the network unless the routing profile has been carefully constructed to stabilize the traffic segmentation between the two paths.

Two paths may be used to provide differentiated services. In Figure 2, the routers along the low-speed path could forward non-timely traffic, while real-time traffic could be forwarded along the high-speed path.

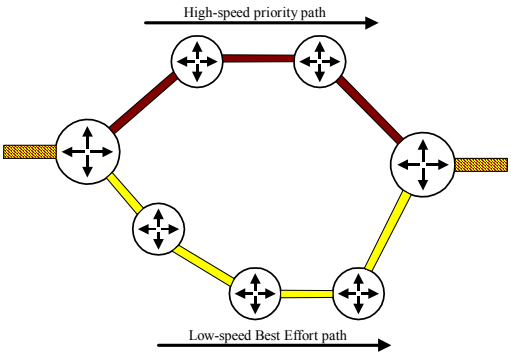

Figure 2. Flows may be forwarded through different paths in a network.

2.3.3 Data Link layer

2.3.3.1 Introduction

This section describes that interaction of QoS mechanisms within various levels of the OSI model is chaotic. Without coherence between signaling at the data link layer and the higher-level protocol stack, the result, in terms of consistency of service quality, is chaotic.

Traditionally, differentiation of traffic at the link layer has been associated with Asynchronous Transfer Mode (ATM) [13] and Frame Relay [14]. A brief overview is given to show how each of these technologies can provide service differentiation, and additionally, we give an overview of the newer IEEE 802.1p mechanism [15].

2.3.3.2 ATM

ATM provides high-speed data-transport together with a complex subset of traffic-management mechanisms. It has Virtual Circuit (VC) establishment controls, and various associated QoS parameters for these VCs. ATM has the capability of providing predictive and dynamic real-time services. Examples may be dynamic allocations of resource guarantees, virtual circuit rerouting, and virtual circuit path establishment to accommodate subscriber QoS requests. There are however problems associated with relying solely on ATM to provide QoS in a network.

Higher-layer protocols, such as TCP/IP, provide the end-to-end transportation service in most cases, so although it is possible to support QoS in a lower layer of the protocol stack, ATM covers only parts of the end-to-end data path. If ATM is not generally deployed end-to-end in the data path, efforts to deliver QoS using ATM can be unproductive. It is difficult to fully exploit the QoS parameters available in ATM, and a problem with IP over ATM is that the flow control of ATM simply pushes the congestion to the edges of the network, i.e. the routers, where performance degradation or packet loss may occur as a result.

Aside from traditional data services that may use ATM, it is clear that ATM provides the QoS necessary for interactive applications like telephony. However, delivering voice services on virtual digital circuits using circuit emulation is quite different than delivering packet-switched data. It is considerably more difficult to deliver QoS for packet-switched data, because the higher-layer applications and protocols do not provide the necessary links to utilize the QoS mechanisms in the ATM network. As a result, an intervening router must make the QoS request on behalf of the application, and thus the ATM network really has no way to determine what type of QoS the application may truly require.

2.3.3.3 Frame Relay

Frame Relay was originally developed for use as a packet service technology in ISDN. It was selected for end-to-end signaling at the transport layer of the protocol stack to perform error detection, retransmission, and flow control. Frame Relay allows the network switches to forward data-link frames without waiting for positive acknowledgment from the next switch. This in turn allows the switches to operate with less memory and to drive faster circuits. Frame Relay is a good example of what is possible with relatively sparse signaling capability. However, the match between Frame Relay as a link layer protocol, and QoS mechanisms for an IP network, is not a particularly good one.

Frame Relay networks has its own ways to discard frames and enforce rate limits on traffic as it enters the network. This is done as the primary response to congestion, but Frame Relay

does not pay any respect to hints provided by the higher-layer protocols. The end-to-end TCP protocol uses packet loss as an indication of network congestion, and Frame Relay offers no great advantage over any other link layer technology in addressing problems when the network starts to reach a congestion state.

2.3.3.4 IEEE 802.1p

The IEEE 802.1p signaling technique is an OSI Layer 2 standard for prioritizing network traffic at the data link/MAC sublayer. 802.1p traffic is simply classified and sent to the destination, thus no bandwidth reservations are established. 802.1p is a spin-off of the 802.1q standard, which specifies a tag that is appended to a MAC frame. The tag in 802.1q consists of a 12-bit VLAN ID, and 802.1p defines an additional 3-bit priority tag that was never defined in the 802.1q standard.

802.1p establishes eight levels of priority, and network adapters and switches route traffic based on the priority level. This will provide a way to transport a prioritized frame across a subnetwork in a consistent method for Ethernet, token ring, or other MAC-layer media types. 802.1p support QoS, but is not able to classify packets itself, thus it needs an overlay end-to-end technology to do this.

2.3.4 Network layer

The network layer and the Internet Protocol (IP) operate end-to-end of the network. IP usually operates in a combination with the transport protocols TCP or UDP. The best QoS technologies are implemented at the network layer, simply because IP can control the data flow end-to-end. Some of the end-to-end QoS features are actually implemented at the transport layer. One example is the TCP congestion control.

It is possible to provide QoS on lower layers of the protocol stack. However, we have seen that such services only cover parts of the end-to-end data path, and the overall outcome of a partial QoS structure is inefficient. An IP packet may traverse an uncertain number of link-layer paths, and each may possess its own characteristics to provide traffic differentiation. However, the packet also traverses link layers that cannot provide traffic differentiation, picturing that providing QoS solely at the link layer is an inadequate solution.

The most dominating part of the OSI model is clearly the network and transport layer, which makes a perfect interaction between network services and end-user functions (see Figure 1). A single link-layer media will never be used end-to-end across all possible paths, though it is possible in smaller private IP networks, and perhaps in smaller peripheral networks on the Internet.

We have shown that QoS mechanisms should be implemented at the network layer, and they will be thoroughly presented and discussed throughout this thesis.

2.4 Real-time applications

There is more to transmitting audio and video over a network than just providing sufficient bandwidth. We refer to applications that are sensitive to the timeliness of data as real-time applications. The characteristics of real-time applications are that they need some sort of assurance from the network that data is likely to arrive on time. Non-real-time applications focus more on the correctness of the data that are transmitted. This means retransmission when data arrives too late or is corrupted. Retransmission means increased latency, but no harm is done as long as the data arrives within reasonable time limits.

The Best Effort model tries to deliver data, but makes no promises neither for timeliness nor guaranteed delivery. This is not sufficient for real-time applications, thus new service models with better guarantees are introduced in this section. We make a summary of different kinds of applications to better understand how complex the need for QoS guarantees are.

We can divide applications in two types: non-real-time and real-time. Non-real-time applications are also called elastic and include common applications like Telnet, FTP, email, Web browsing, and so on. They are often bursty, i.e. they have unpredictable delivery of "blocks" of data at a variable bit rate (VBR). All of these applications can work without the guarantees of timely deliver of data, but the delay requirements may vary from interactive applications like Telnet to more asynchronous ones like email.

Real-time applications can be divided into two groups, interactive applications and one-way streaming applications. Both have predictable delivery at a relatively constant bit rate (CBR). Two or more people that talk together on the Internet typically use an interactive application. They have strict demands to delay and the amount of data that are transferred is small. Today, such data gets delayed by other traffic on the Internet and may arrive too late at the receiver. IP Telephony, also referred to as Voice over IP (VoIP), is today's most well known example. One-way streaming services are less delay sensitive, since the data is sent only in one direction. Streaming usually aims at giving a live audio or video experience at the receiver. The service uses an adaptive playback buffer to limit variations in delay. Big Brother live is an example of one-way streaming. Table 1 summarizes QoS requirements for some common application types.

Table 1. QoS requirements for common application types.

Application Types	QoS requirements			
	Bandwidth	**Latency**	**Jitter**	**Packet Loss**
E-Mail	Low to Moderate	-	-	-
File Transfer	Bursty High	-	-	-
Telnet	Bursty Low	Moderate	-	-
Streaming Media	Sustained Moderate to High	Sensitive	Sensitive	Sensitive
Videoconferencing	Sustained High	Critical	Critical	Sensitive
Voice over IP	Sustained Moderate	Critical	Critical	Sensitive

Playback time is the point in time at which the data from the sender is needed at the receiving host. Recommendations from ITU-T [16] show that a playback time less than 150 ms is acceptable for most user applications. 150 to 400 ms is acceptable provided that we are aware of it, and delays above this is unacceptable. Data that arrives after the playback time is completely worthless.

Usually, a playback buffer (Figure 3) is used to make sure that arrived data is played back at a steady rate in an application. It is used to minimize jitter introduced when packets are traversing a network, and as long as the playback time is after packet arrival and within acceptable time limits, jitter is never noticed by the application. Network delay may be very variable, and usually a small percentage of the packets arrive very late in comparison with the

rest, therefore it is always smart to set the playback point in such a way that we have some packet loss.

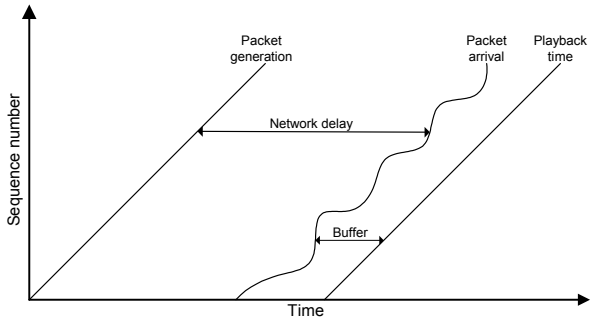

Figure 3. The role of a playback buffer.

If the packet loss varies with time, the playback point may be shifted to play out samples at an increased or decreased rate for some period of time. With a voice application, this can be done in a way that is barely perceptible, simply by shortening or increasing the silences between words. Applications that can adjust their playback point are called delay-adaptive.

We also have rate-adaptive applications, which are used in videoconferencing. Many video-coding algorithms can trade-off bit rate versus quality, so if the network only supports a certain bandwidth, the picture is compressed harder. If more bandwidth becomes available later, we can lower the compression to increase the quality. Intolerant applications that do not tolerate the distortion of delay adaptivity may be able to take advantage of rate adaptivity.

Real-time applications are used in many different areas. We are focusing on QoS in IP networks for communication tools like telephony and conferencing, which is expected to have an enormous growth on the Internet in the next few years. Real-time applications are also widespread in factory automation, especially applications that monitor and control equipment. Other examples are medical equipment, pay-TV systems for hotels and signaling systems in new cars. Today, applications with critical demands for delivery within a certain time (intolerant applications) often use proprietary network standards, and there are a lot of them. In the future the bandwidth allocation guarantees from an IP network will get more reliable, and intolerant applications hopefully will be able to use IP as the common network platform.

An example of a network that is adjusting to the IP standard is the global mobile telephone network. Third generation mobile networks will have their own IP backbones connected to the Internet, and there will be a demand for QoS guarantees for real-time applications in the same way as in fixed networks. The demands for throughput and delay are difficult to fulfill, and it is important that resources are shared in the best possible way.

As we have seen, new applications have new service requirements, and this thesis is going to evaluate how an IP network can satisfy the needs of these new applications. We look at service models with not just one Best Effort class, but with several classes, each available to meet the needs of some set of applications.

3 The service models

3.1 Overview

To increase the knowledge about how packets are handled in a router, section 3.2 describes the forwarding process in a router and how it provides QoS. The forwarding of packets is described by dividing it into different parts, and different queuing and queue management techniques are described and discussed. Later, when we describe the service models, we refer to these techniques, thus they are fundamental to understand how the service models are implemented in the routers.

In the succeeding sections, the three service models (Best Effort, IntServ and DiffServ) are described and discussed individually.

3.2 QoS forwarding mechanisms

3.2.1 The forwarding engine

Forwarding is the process of taking a packet, looking at its header and giving it a proper treatment before it is sent to the correct output interface. The forwarding mechanisms in a router may be divided into different parts as Figure 4 shows. These mechanisms are basic to understanding how the service models influence real-time traffic and other traffic in an IP network.

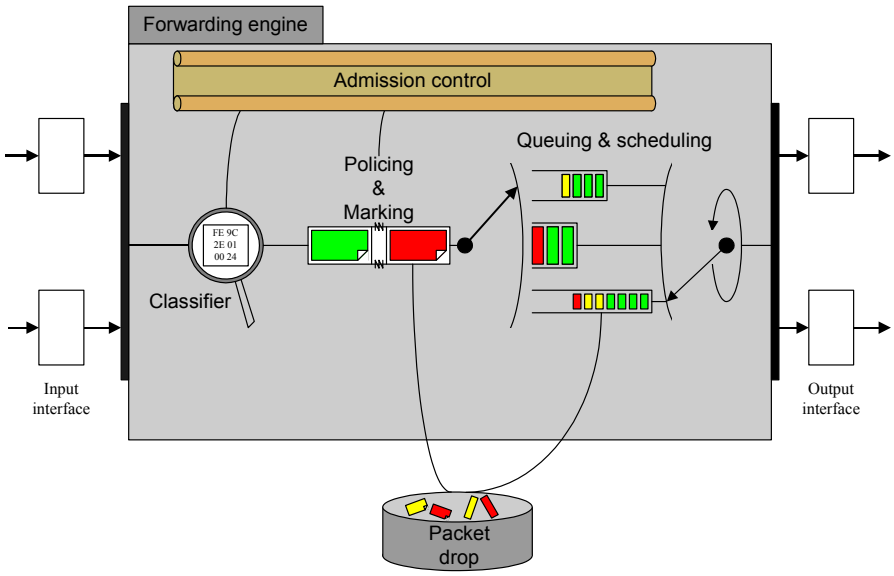

Figure 4. Packet handling in a router that supports QoS.

Admission control and policing & marking are not possible in a Best Effort router, though classification and queuing & scheduling concern routers independent of the service model supported. Each element in Figure 4 will be described in the subsequent sections.

3.2.2 Admission Control

A real-time service supported with some QoS-level depends on setting up a state in the router and making commitments to certain classes of packets. In order to ensure that these commitments can be met, it is necessary that resources are explicitly requested, so that the request can be refused if resources are not available. The decision about resource availability is called admission control.

Admission control requires that the router understands the demands that are currently being made on its resources. The approach traditionally proposed is to remember the service parameters of past requests, and make a computation based on the worst-case bounds on each service. A recent proposal, which is likely to provide better link utilization, is to program the router to measure the actual usage by existing packet flows and to use this measured information as a basis of admitting new flows. This approach is subject to higher risk of overload, but may prove much more efficient in using bandwidth.

Note that while the need for admission control is a part of the global service model, the details of the algorithm run in each router is a local matter. As a result, vendors can compete by

developing and marketing better admission-control algorithms, which lead to higher link loadings with fewer service-overloads.

3.2.3 Packet Classification

Today, routers look at the destination address in the IP-packet and select a route through the network. The destination address alone is not always sufficient to select the service a packet must receive, so we need more information. Packet classification is about determining a packet's needs and rights for treatment by the router. A packet can flexibly be classified according to e.g. the transport protocol (TCP, UDP), incoming interface, packet size, source or destination address.

For DiffServ, the DSCP field is used to mark packets according to a given classification. The DSCP field can also be set by the transmitting application to indicate what QoS the packets demand. To avoid corrupt applications setting QoS demands that are not justified, it is suggested to have some form of control of who is allowed to use the different service levels that again forces the classification algorithm to look deeper into the packets.

As we can see from Figure 5, classification will have influence on policing & marking and queuing & scheduling. With this classification as a basis, the routers can choose the proper handling for the packets.

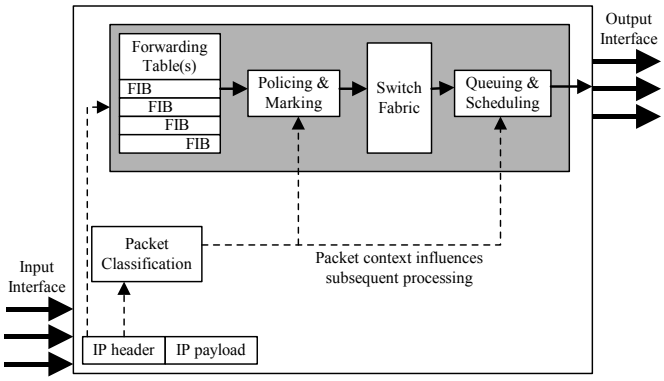

Figure 5. An overview to how classification influences forwarding [17].

How thoroughly a packet will be analyzed depends on what speed the router is able to analyze the packets in. While routers serving slow links will be able to analyze the packets more thoroughly, routers on high bandwidth links may not be able to do so. The accuracy at which the routers will analyze packets is dependent on what the network administrators find suitable for that exact network type.

3.2.4 Policing & marking

There is a limit on how fast packets may arrive in a router and policing & marking are actions taken to find out if packets are out of profile or not. Policing is the process of ensuring that incoming traffic belonging to a given class is conforming to the traffic profile defined for that class. A policy may decide packets out of profile to be dropped, marked with lower priority in the scheduler, or just passed on to the queuing and scheduling stage.

A related function is traffic shaping, which modifies the temporal characteristics of a traffic class by selectively delaying local packet forwarding. Shaping is covered in subsection 3.2.5.

Before performing policing & marking, we need a meter to monitor traffic patterns of traffic belonging to a given class against a given traffic profile. A simple example is the classic token bucket meter, which enforces an average rate limit within a particular traffic class, only allowing a chosen degree of burstiness. Tokens are added to a bucket at a fixed rate of tokens per second, and are removed from the bucket whenever a packet arrives. A bucket also has a finite depth, which prevents it from ever containing more than a given number of tokens. Figure 6 shows three token buckets providing a simple metering function.

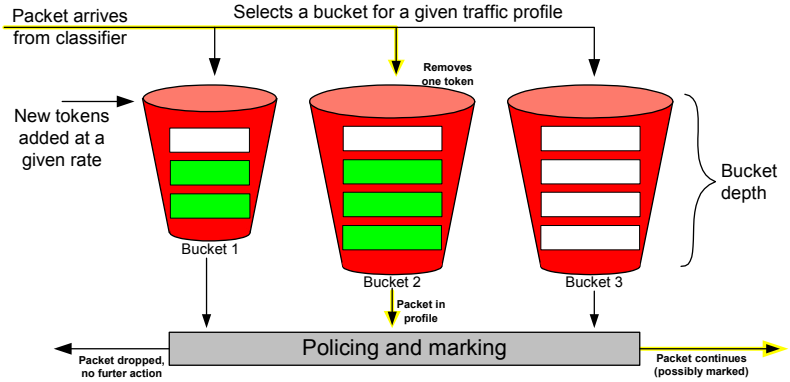

Figure 6. Token buckets providing a simple metering function.

Policing & marking is critically important to the DiffServ service model, where marking is often referred to as "coloring". Two examples are "A single rate Three Color Marker" [18] and "A two rate Three Color Marker" [19].

The two rate three color marker (trTCM) marks packets "red" if they are completely out of profile, "yellow" if they are just out of profile, and "green" otherwise. The colors correspond to various drop precedence levels, with red being the highest precedence and green the lowest. For example, a heavily congested network may discard all red packets because they have exceeded the peak rate, forward yellow packets with Best Effort and green packets with low drop probability. trTCM preserves the order of packets that are not dropped, independent of the drop probability.

3.2.5 Queuing & Scheduling

3.2.5.1 Introduction

Queuing controls in what manner the packets are passed on to the output queue. Some network elements enable "fair queuing" algorithms so that a misbehaving application, one that continues to send during times of congestion, will not be destructive to other better-behaved applications like TCP applications. Basically, they determine how packets are dropped when congestion occurs in a router (e.g. when a queue is full).

Nodes also need to have the closely related scheduling mechanisms to ensure that different "connections" obtain their promised share of the resources (e.g., processing and link bandwidth). This mechanism also ensures that any spare capacity is distributed in a fair manner.

A scheduler can also provide rate shaping by enforcing a packet stream to be conformant to a traffic profile. A bursty interactive flow could be shaped to a constant bit rate in a scheduler. Shaping is often based on the leaky bucket mechanism, since packets leak out of the queue at a fixed rate (should not be confused with a token bucket meter). Thus, the bursts of a stream will be smoothed and non-conforming packets will be delayed.

The basic function of packet scheduling is to reorder the output queue. There are many possible ways to manage the output queue and the resulting behavior. Perhaps the simplest approach is a priority scheme where packets are ordered by priority, and highest priority packets always leave first. This has the effect of giving some packets absolute preference over others. If there are enough of the higher priority packets, the lower priority class can be completely prevented from being sent. Priority Queuing (described in 3.2.5.5) is an example of such a mechanism.

An alternative scheduling scheme is round-robin, which gives different classes of packets access to a share of the link. As an example Weighted Fair Queuing (described in 3.2.5.4) has the ability to allocate the total bandwidth of a link into specified shares.

3.2.5.2 First In, First Out Queuing (FIFO)

FIFO is a simple tail-drop queuing mechanism. This is the simplest and most common interface queuing technique and works well if links are not congested. The first packet to be placed on the output interface queue is the first packet to leave the interface, as illustrated in Figure 7.

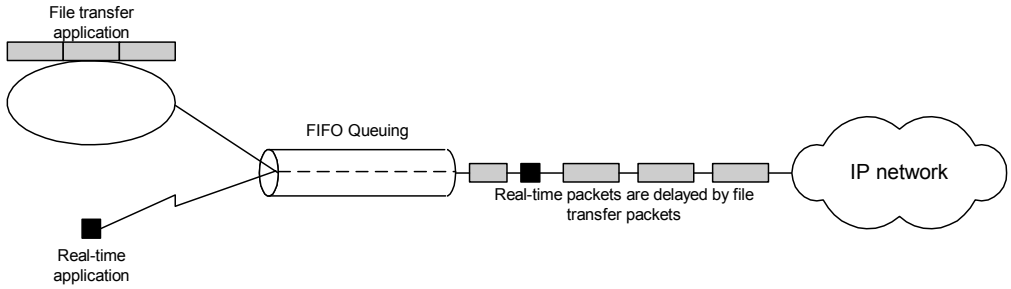

Figure 7. Potential impact of a file transfer on Interactive Traffic with FIFO queuing.

The problem with FIFO queuing is that when a host starts a file transfer, it can consume all the bandwidth of a link to the disadvantage of real-time traffic. The phenomenon is referred to as packet trains because one source sends a “train” of packets to its destination and packets from other hosts get caught behind the train. First in, first out queuing is efficient for large links that have little delay and minimal congestion.

3.2.5.3 Fair Queuing (FQ)

One of the main setbacks with FIFO queuing is that it does not separate packets according to the flow that they belong. Fair Queuing is an algorithm that has been proposed to address this problem. The idea of FQ is to maintain a separate queue for each flow currently being handled by the router. The router then services these queues in a round-robin manner as illustrated in Figure 8. When one flow sends packets too fast, the queue will fill up and packets will be discarded when a certain limit is reached. However, this action does not have any influence on the service delivered to other flows on the router.

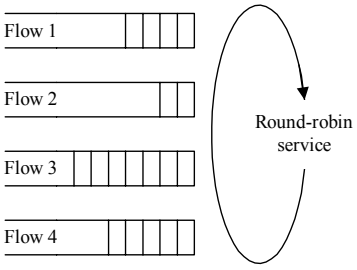

Figure 8. Fair Queuing at a router.

FQ should be used in combination with an end-to-end congestion control mechanisms. It will not give any feedback to the sender, besides lost packets or increased delay, whether the router is about to be congested or not. There is also no possibility to give a larger share of bandwidth to a flow. Because of this, FQ can be thought of as the basis for some of the more sophisticated queuing mechanisms used in routers. An example of this is weighted FQ.

3.2.5.4 Weighted Fair Queuing (WFQ)

WFQ is the best known and the most studied queuing discipline. It assigns a queue for each flow and applies priority (or weights) to identified traffic to determine how much bandwidth each flow is allowed relative to others. As a result, WFQ can prevent other flows to have direct influence on one specific flow.

Flows are broken into two categories: those requiring large amounts of bandwidth and those requiring a relatively small amount of bandwidth. The goal is to always have bandwidth available for the low throughput flows and allow the high throughput flows to split the rest proportionally to their weights.

In Figure 9, packets arrive at the router in the order indicated on the left hand side. Then, the packets are reordered according to their size and the amount of similar packets so that packets from the Voice over IP flow are moved forward in the queue.

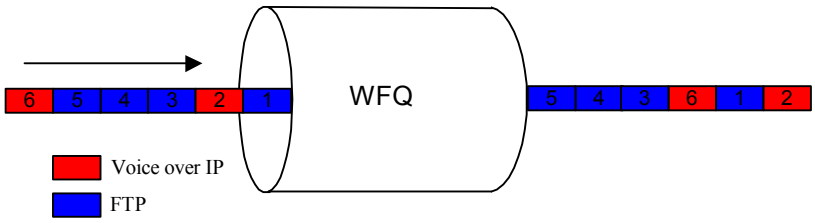

Figure 9. An example on how Weighted Fair Queuing reorders packets.

In WFQ, packets are reordered so that low-volume flows are moved forward and high-volume flows are moved towards the tail of the queue. This reordering results in packet trains being broken up and low-volume flows receiving preferential service. The high-volume flows share the delay by equal reordering, where no flow is affected more than another.

WFQ can guarantee an even distribution for classes of packets with similar behavior, but the guarantees for individual connections disappear through aggregation.

3.2.5.5 Priority Queuing (PQ)

PQ ensures that important traffic gets the fastest handling at each point where it is used. It is designed to give strict priority to important traffic according to what is set by the classifier.

In PQ, each packet is placed in one of four queues; High, Medium, Normal, or Low based on an assigned priority. Packets that are not classified by this priority-list mechanism fall into the Normal queue. During transmission, the algorithm gives higher-priority queues absolute preferential treatment over low-priority queues.

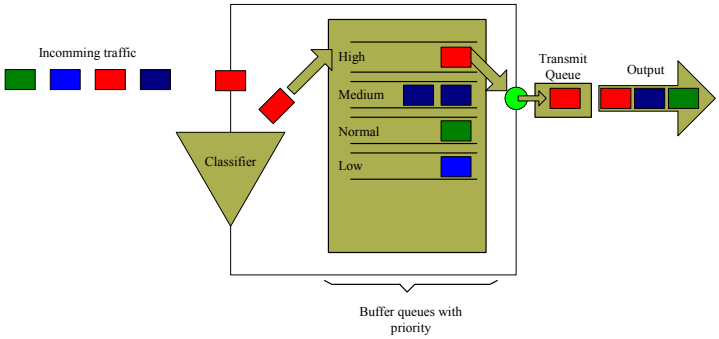

Figure 10. Priority Queuing.

PQ is useful for making sure that mission-critical traffic, such as real-time VoIP traffic, traversing various links gets priority treatment. PQ uses static configuration and does not automatically adapt to changing network requirements.

Although PQ provides a simple and intuitive approach to scheduling, it can cause queuing delay and increased jitter on the lower priority traffic. Depending on the bandwidth used by higher-priority packets, this could result in lower priority traffic never being transmitted.

To avoid inflicting these conditions on lower priority traffic, we can use traffic shaping or Committed Access Rate (CAR) [20] to rate-limit the higher priority traffic.

3.2.5.6 Class Based Queuing (CBQ)

Class Based Queuing reserves a portion of the link bandwidth for each selected traffic type. It works almost the same way as PQ, but while PQ always prioritize the most important queue CBQ can share the bandwidth among several classes. To configure CBQ, the network manager must determine how much bandwidth to reserve for each traffic type. If a particular type of traffic is not using the bandwidth reserved for it, then other traffic types may use the unused bandwidth.

CBQ works by cycling through the series of queues in a round-robin order and sending the portion of allocated bandwidth for each queue before moving to the next queue. If one queue is empty, the router will send packets from the next queue that has packets ready to be sent. The queuing of packets is still first in, first out in each classification, but bandwidth sharing can be achieved between the different classes of traffic.

In Figure 11, CBQ is configured to give 50 % bandwidth to file-transfer traffic, 25 % to real-time traffic, and 25 % to default traffic. If the file-transfers use all their allocated bandwidth, the other queues can utilize this bandwidth until more file-transfer traffic arrives.

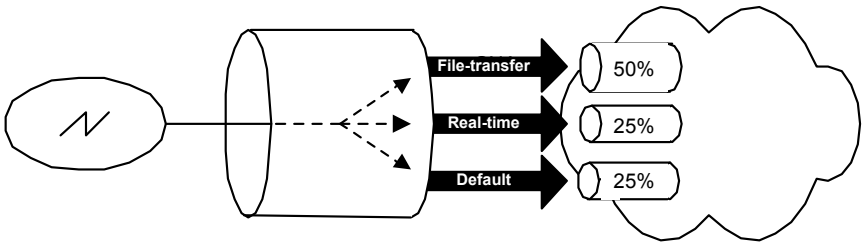

Figure 11. Class Based Queuing.

CBQ does not fragment packets, and therefore situations will arise where large packets occupy more bandwidth than the stated percentage of the total bandwidth. This happens when the round-robin distributor "rotates" at higher speeds than the rate of incoming packets. While file-transfers will try to send the packets as fast as possible, real-time packets have to send its packets at the specified rate. A solution could be to reduce the "rotation" speed of the round-robin distributor so that the data will have to wait longer before being handled. However, this solution results in that small real-time packets often experience larger delay and jitter than presumed.

3.2.5.7 Hierarchical Fair Share Curve (HFSC)

In hierarchical link-sharing there is a class hierarchy associated with each link that specifies the resource allocation policy for the link. A class represents a traffic stream or some aggregate of traffic streams that are grouped according to the classifier. Figure 12 shows an example class hierarchy for a 45 Mbps link that is shared by two organizations, Ericsson and HiA. Each class is associated with its resource requirements, which is the minimum amount of service that the traffic of the class should receive when there is enough demand.

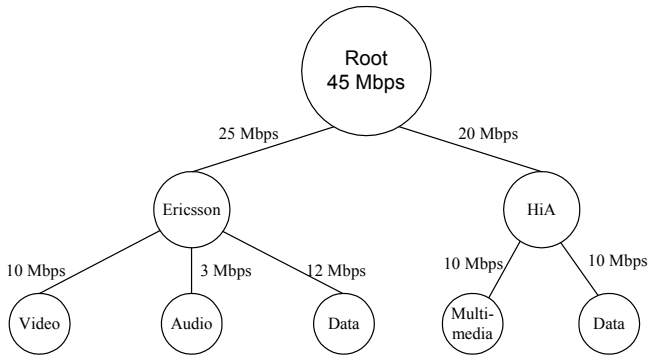

Figure 12. An example of a link-sharing hierarchy.

If there are only audio and video streams from Ericsson, the streams should receive all the bandwidth that is allocated to Ericsson if the demand is high enough. The policy set for the system will distribute the excess bandwidth among the active classes.

The scheduling is based on two criteria; the real-time criterion that ensures the service curves of all leaf classes to get their guaranteed bandwidth and the link-sharing criterion that aims to satisfy the service curves of the interior classes and fairly distribute the excess bandwidth.

HFSC provides independent delay and bandwidth allocation in an integrated fashion. It can also improve real-time traffic's problems with high delay and jitter [21].

3.2.6 Active Queue Management

3.2.6.1 Introduction

A router must drop packets when its buffers are all full. This fact does not determine which packet should be dropped, and this is not an easy decision, since dropping the arriving packet may cause undesired behavior.

In the context of today's Internet, with TCP operating over a Best Effort service, dropping a packet is taken by TCP as a signal of congestion and causes it to reduce its load on the network. Thus, picking a packet to drop is the same as picking a source to throttle. This simple relation suggests that a dropping control should be implemented in the router to improve congestion control.

In the context of real-time services, dropping more directly relates to achieving the desired QoS. If a queue builds up, dropping one packet reduces the delay of all the packets behind it in the queue. The loss of one packet can contribute to the success of many. The problem is to determine when the delay bound is in danger of being violated. If there is a priority scheme in place, packets with low priority can be postponed indefinitely, so even a short queue may have very old packets in it.

The boundaries for dropping packets must be set according to the size of the in- and output buffers on the node. As an example, large buffers with a high threshold for dropping increases the performance for TCP flows that are bursty of nature. On the other hand, real-time flows with fixed transmission rate often experiences long delays with such a configuration. With smaller buffers, TCP packets are going to be dropped more often, but the delay will be reduced in favor of real-time flows that are dependent on minimal delays.

To reduce delays, it is important to avoid congestion in the routers. The congestion control algorithms described below addresses this issue.

3.2.6.2 Random Early Detection

Random Early Detection (RED) provides a mechanism to avoid congestion collapse by randomly dropping packets from arbitrary flows in an effort to avoid the problem of global synchronization and, ultimately, congestion collapse. The principal goal of RED is to avoid a “queue tail drop” situation in which all TCP flows experience congestion at the same time, and subsequent packet loss, thus avoiding global synchronization. RED also attempts to create TCP congestion signals using duplicate ACK signaling, rather than through sender timeout, which in turn produces a less catastrophic rate backoff by TCP [23].

RED monitors the mean queue depth, and as the queue begins to fill, it begins to randomly select individual TCP flows from which to drop packets, in order to signal the receiver to slow down. The threshold at which RED begins to drop packets is generally configurable by the network administrator, as well as the rate at which drops occur in relation to how quickly the queue fills. The more it fills, the greater the number of flows selected, and the greater the number of packets dropped. This results in an indication to a large number of senders to slow down and a more manageable congestion avoidance.

Although Figure 13 may indicate that the probability of drop $P(drop)$ only is a function of $AvgLen$ (average length of the queue), it also is a function of how long it has been since the last packet was dropped.

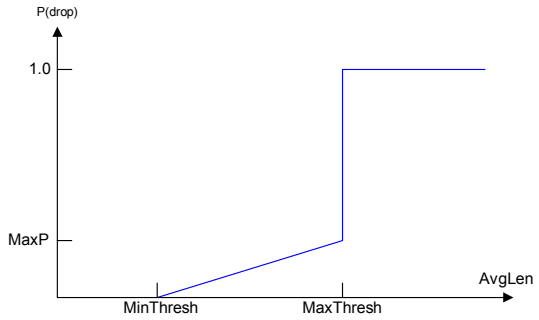

Figure 13. Drop probability function for RED.

Although RED will provide means to control the congestion, it is important to notice that it will only have effect on TCP packets. It is not recommended for protocols that respond to dropped packets by retransmitting the packets at the same rate. The RED routers also discard packets earlier than typical routers. In a network with both TCP and other protocols, this will result in unfairly reduced capacity for the TCP connections since they are "intelligent" enough to reduce the transfer rate [24][25].

There are some problems using RED. One of the problems is that the performance and behavior of RED is extremely sensitive to the parameters it is configured with. For example, a misparameterized RED queue can fail to control the transmission rates of low priority sources, allowing them to fill the queue with packets and cause priority packets to be dropped. Another problem is that RED's congestion detection mechanism is based on the calculation of an average queue length. Since the instantaneous queue length can change quite dramatically over short time intervals, queue overflow and loss of priority packets can occur before the increase in average queue length can trigger RED's congestion control mechanisms. In order to address the shortcomings of RED's active queue management algorithm, recent improvements labeled BLUE [26] have been proposed. While these improvements can potentially reduce the amount of packet loss observed, they alone still cannot ensure priority packets safe passage.

3.2.6.3 RED with In and Out (RIO)

RIO combines the capabilities of the RED algorithm with drop precedence. This combination provides preferential traffic handling for higher priority packets. It can selectively discard lower priority traffic when the interface begins to get congested, and provide differentiated performance characteristics for different services. RIO differs from other congestion management techniques such as queuing strategies because it attempts to anticipate and avoid congestion rather than controlling congestion once it occurs [27].

Figure 14 shows how RIO works and we can see the drop probability on the y-axis increases as average queue length increases along the x-axis. RIO has one curve for traffic that is conformant with the boundaries set in the router (in) and one that are not (out).

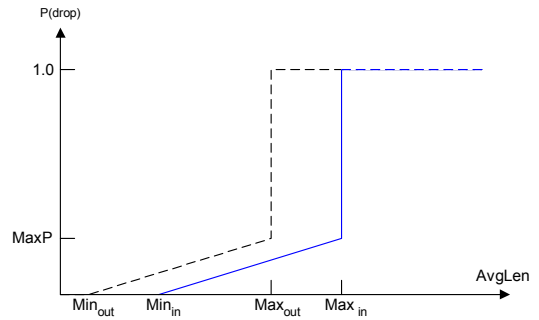

Figure 14. RED with In and Out drop probabilities.

From the figure, we can also see that traffic that are not conforming to its policy will reach a shorter queue length before packets are discarded.

RIO uses the ToS-field marked by the edge routers to determine how it treats different types of traffic. It provides separate thresholds and weights for different packet classes, allowing us to provide different QoS for different traffic. Standard traffic may be dropped more frequently than premium traffic during periods of congestion.

Just like RED, the efficiency of a mechanism like RIO depends to some extent on correct parameter choices, and there are considerably more parameters to set for RIO.

One interesting property of RIO is that it does not change the order of “in” and “out” packets. If a TCP connection is sending packets through a profile meter, and some packets are being marked “in” while others are marked “out,” those packets will receive different drop probabilities in the router queues, but they will be delivered to the receiver in the same order as they were sent.

The idea of RIO can be generalized to provide more than two drop probability curves, and this is the idea behind the approach known as Weighted RED (WRED) [25].

3.3 Best Effort

3.3.1 Introduction

Best Effort is about the simplest service we can ask for from an IP network. The “best-effort” part means that if something goes wrong and the packets get lost, corrupted, misdelivered, or in any way fails to reach its intended destination, the network does nothing to handle these problems. It does its best effort, and that is all it has to do.

3.3.2 Description

Best Effort offers no kind of quality control, this has to be done by the protocol stack at the end host. For example, TCP acknowledges packets to confirm that they have been received correctly. If the source of a flow does not receive an acknowledgement, it simply tries again. For real-time services such as VoIP, the receiver does not have time to wait for retransmission of lost packets. Since, even small losses in packets will result in decreased quality.

Figure 15 shows the simple queue management at which Best Effort handles incoming flows. If the buffer on the incoming interface is full, the excess packets will be dropped. For streaming applications it is important to have a more reliable transfer of packets than this.

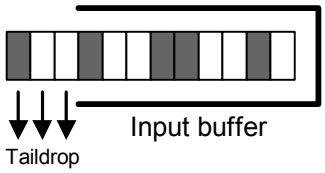

Figure 15. FIFO with taildrop.

The first attempt to increase the reliability of Best Effort was to add more sophisticated queuing algorithms. The implementation of queuing algorithms, such as CBQ and WFQ, give the streaming flows an increased QoS, as explained in the previous section.

A suggested improvement for Best Effort is the use of the 3-bit IP precedence field. By using this field, we can ask for a packet with a higher precedence to arrive no later than a packet with a lower precedence if both packets are passed into the network at the same time with the same destination and IP options. In addition, the IP precedence field is related to discard probability, meaning that if a sequence of marked packets is passed along a path, the packet sequence with the elevated precedence should experience a lower discard probability. However, network hardware and software have traditionally not been configured to use this field.

3.3.3 Discussion

As the Internet now is of a vast size, the transition from one service model to a new one is not easy. Best Effort is still viewed as a successful service model, and there have been several possible solutions to overcome problems concerning guarantees for real-time applications.

A possible solution to handle such traffic is to always have enough capacity available in the network and in this way limit the complexity. At the speed that the Internet is evolving today,

this would require an overdimensioning in capacity to handle the ever-increasing traffic, which again would result in very high costs for service providers.

For TCP traffic, the rate at which the capacity have to be increased can be kept down by the fact that TCP will retransmit the packets. As the amount of real-time traffic is increasing, even small losses will be noticeable. Therefore, the basic Best Effort service cannot properly support real-time connections except if the load level is so low that buffers are continuously almost empty.

In concern to fairness, Best Effort has two primary user groups: Ordinary users and skillful users with considerable ability to tune their computer system. The harmful part of the skillful user group always tries to exploit the network in order to receive the best throughput possible, and this will degrade the service delivered to the friendly users. If this happens in a large scale, it does not only deteriorate overall fairness but also deteriorates the service for all users.

As we can understand, Best Effort cannot deliver the Quality of Service that is necessary to be able to transfer real-time streaming at the required quality. In order to separate the time-dependent flows from the rest, the network have to promise service guarantees.

By implementing more sophisticated queuing algorithms, the reliability can be increased a little. However, even small losses of packets are highly visible and audible, so the real-time applications demand a higher Quality of Service. This leads us forward to QoS oriented network architectures such as Integrated Services (IntServ) and Differentiated Services (DiffServ).

3.4 Integrated Services

3.4.1 Introduction

Integrated Services (IntServ) [1] and the Resource Reservation Protocol (RSVP) [28] are strongly connected, thus we describe these two technologies together. While IntServ is handling the classification and marking of the packets, RSVP is used for signaling to make the actual reservations in the routers.

3.4.2 The Integrated Services framework

The Integrated Services framework offers applications to choose among multiple, controlled levels of QoS for their data packets. To support this capability, some changes compared with the network today have to be done. Routers along the path must support mechanisms to control the desired QoS. There must also be a way to communicate the application's requirements to the routers.

In the Integrated Services framework, the QoS control is provided by Controlled Load and Guaranteed Service. The communication between applications and routers may be provided in a number of ways, but is frequently implemented by the resource reservation setup protocol named RSVP. Figure 16 illustrates the functionality of IntServ and RSVP.

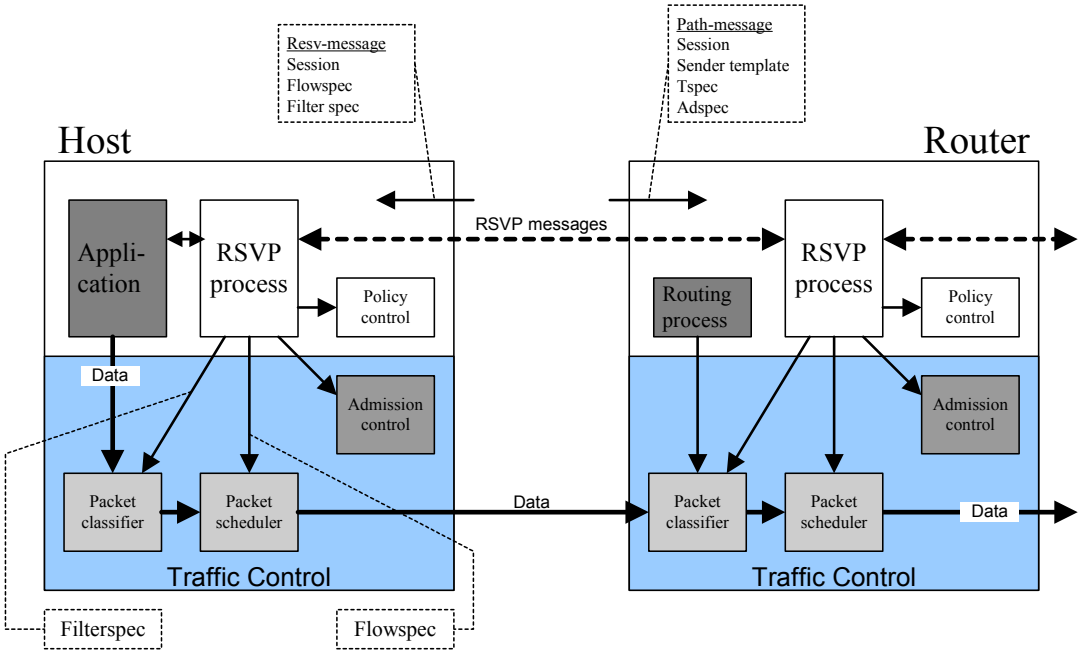

Figure 16. The Integrated Services framework [28].

In the IntServ framework, each QoS-capable node feeds all incoming packets to the packet classifier. The packet classifier decides to which route and QoS class each packet belongs. A packet scheduler then queues and forwards the packets according to the promised QoS.

In all the nodes, two modules, the admission control module and the policy control module, handle resource reservation requests. The admission control module makes sure that the network resources suffice for the request. The policy control module determines who can and who cannot reserve network resources on the node. The packet classifier, the packet scheduler

and the admission control module form the traffic control, shown in the blue boxes of Figure 16. They actually implement the QoS defined in the resource reservation.

There are differences between the RSVP process in a host and a router. In a network host, the RSVP process provides an application interface (API) to the application programs. It also receives and sends the RSVP messages, authenticates requests with the policy control module and relays the objects concerning flow classifying, scheduling and admission control to the appropriate traffic control modules within the local host. These objects are opaque to the RSVP process itself.

In a router, the RSVP process relays the appropriate objects to the traffic control. It also stores these objects in the flow state. The RSVP process uses the routing databases to route RSVP messages to appropriate destinations.

3.4.3 RSVP

Source applications supporting IntServ uses RSVP to tell the receiving application about its coming flow, and the receiving application makes the actual reservations (see Figure 17). Actually, there are other reservation protocols that can be used together with IntServ, but RSVP is the protocol being used. RSVP communicate end-to-end by signaling and is in this way behaving like a transport protocol, but RSVP does not transport data. As a result of this, it is more relevant to compare RSVP with routing protocols, since it is used only for signaling. RSVP can be transferred over IP using protocol number 46, but the most used solution is to transfer it using UDP.

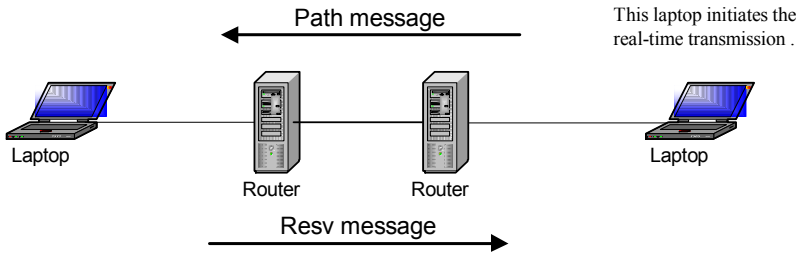

Figure 17. Signaling with RSVP.

An application wishing to make reservations in the data transmission path sends RSVP “path” messages along the route provided by the routing protocol. These “path” messages store “path state” in each node along the way. “Path state” includes at least the unicast IP address of the previous hop node, which is used to route the “Resv” messages hop-by-hop in the reverse direction.

A path message contains among other things a Session and a Sender Template object. Together they define which packets that belong to the flow. The sender includes the traffic parameters of the data flow it generates in the Sender TSpec object. The traffic parameters are defined according to the token bucket model, and they include maximum bandwidth, token bucket size and maximum packet size among others. The Adspec object contains the sender’s QoS capabilities. Each router along the path will merge the RSVP capabilities to the Adspec, thus the receiver gets a summary of the RSVP characteristics and the current QoS over the whole path.

Path messages are sent with the same source and destination addresses as normal data packets. Therefore, path messages will be routed correctly through non-RSVP supporting routers. However, Resv messages are sent hop-by-hop. Each RSVP-speaking node forwards a Resv message to the unicast address of the next RSVP-speaking node.

When the path message reaches the receiver, the belonging application will read the contents of the message. Then, with basis in the parameters read, the reservations needed for the flow is calculated. The receiver generates a Resv message and sends it upstream towards the sender. The Resv messages must follow exactly the reverse of the path the data packets will use, and they create and maintains a “Reservation State” in each router along the path, assuming the router has the available resources to initiate the reservation. If the reservation cannot be made, the router will replace the Resv message with an “error” message and send it further up the reservation three. Routers have no ability to negotiate the demands in the Resv message, so if the reservation fails the sender have to try again with reduced demands.

A Resv message contains among other things a FlowSpec and a FilterSpec that together form a flow descriptor. The FlowSpec defines the desired QoS and the FilterSpec defines the source of a data flow. The data flow is a simplex data stream from the sender specified by the FilterSpec of the session. The packet classifier uses the FilterSpec and session, i.e. source and destination addresses, to identify packets belonging to the flow. The FlowSpec describes the traffic in the flow and the resources that the flow needs. The admission control calculates from the FlowSpec how the resources should be allocated for the flow, and the packet scheduler uses it to define how the packets belonging to the flow are queued.

If all the reservations are successful, the Resv message is finally delivered to the sender host and the appropriate traffic control parameters can be set up. It is important to notice that the demands specified in the path message are not necessarily the same as the demands received in the Resv massage. An example of the change in the demands is when the receiving host does not have the bandwidth to accept the specified flow. Then, the receiving host can reduce the bandwidth in order to be able to receive the flow.

The reason why IntServ let the receiver decide the service quality, is to support multicast and to be able to change the reservations when the quality at the receiver is not acceptable.

As long as the link is active, the reservations at the routers will be kept by refreshing it at a specified rate (usually every 30 seconds). This implies that the reservations will terminate when the link is not being used. This is what we call a “soft state”. The reservation of an active link may also be removed by sending a teardown message.

3.4.4 Service classes

3.4.4.1 Introduction

IntServ offers two service classes in addition to Best Effort; Controlled Load and Guaranteed Service described respectively in [29] and [30].

3.4.4.2 Controlled Load

Controlled Load offers a service closely approximated to the treatment that a flow would receive from an unloaded network element. Controlled Load uses admission control to assure that this service is received even when the network element is overloaded.

To realize Controlled Load, appropriate resources are allocated inside the network to each flow. Traffic control is implemented to limit the amount of packets from each flow and their traffic variations. Controlled Load is able to offer a good, though not perfect, service with a relative simple mechanism, which is a prioritization mechanism that separates Controlled Load from pure Best Effort flows.

Users requesting Controlled Load give an estimation of the data traffic they will generate (TSpec). The service provider does not give any guarantees, rather an assurance that a very high percentage of transmitted packets are delivered successfully and that the delay does not greatly exceed the minimum for control parameters such as delay or loss. This way, the service philosophy is better than Best Effort, but without any hard guarantees.

If the traffic of a flow exceeds the limits specified by the TSpec, the flow obtains a similar service as Best Effort with the possibility of long delays and dropped packets. So the transition from Best Effort to Controlled Load is a relatively easy operation for users. Moreover, because the specification is quite open, the network implementation may either rely on low utilization, traffic measurements to predict traffic behavior or on strict traffic control and accurate calculations.

3.4.4.3 Guaranteed Service

"A guaranteed service shall provide firm, mathematically provable guarantees that the end-to-end delay experienced by packets in a flow will not exceed a set limit." [30]

The Guaranteed Service is specified by two sets of parameters: traffic parameters (TSpec) and service-level parameters (RSpec). The RSpec consists of a data rate and a slack term. In addition, two error terms describe the accuracy of the reservation compared to a perfect one. By combining the parameters from the various service elements in a path, the maximum delay can be obtained, but this is quite complicated and will not be discussed further.

If the QoS control defined in RFC 2212 is deployed widely in an IP network, Guaranteed Service gives applications considerable control over their delay. Delay has two parts, fixed delay and queuing delay. Fixed delay is a property of the chosen path, which is determined not by Guaranteed Service but by the setup mechanism. Guaranteed Service, on the other hand, determines queuing delay. Therefore, an application can usually accurately estimate, in advance, what queuing delay Guaranteed Service will likely promise. If the delay is larger than expected, the application can modify the traffic's token bucket and data rate to achieve a lower delay.

3.4.5 Discussion

A function that IntServ offers, is the ability for applications to get explicit feedback from the network. This ability is non-existent in Best Effort, while an IntServ network can give a guarantee whether to deliver Controlled Load or Guaranteed Service. As we might understand, these guarantees do not cover malfunctions out of reach for the routing protocols.

As discussed in RFC 2208 [31], there are several areas of concern about the wide-scale deployment of RSVP. With regard to concerns of RSVP scalability, the resource requirements for implementing RSVP on routers increase in proportion to the number of separate RSVP reservations, or sessions, accommodated. Therefore, supporting a large number of RSVP reservations could introduce a significant negative impact on router performance. However, the suggestion that RSVP has poor scaling properties deserves additional examination. A 155 Mbps pipe can hold about 5000 32-Kbps audio sessions, and a backbone router can easily handle that amount of sessions. Since the deployment of RSVP has not been widespread enough it still is an open question how well RSVP will scale.

The scaling concerns results in that organizations with large, high-speed networks will be reserved to deploy RSVP, at least until these concerns are addressed. The underlying implications of this concern also suggest that without deployment by service providers, who own and maintain the high-speed backbone networks, the deployment of pervasive RSVP

services will not be forthcoming. However, Microsoft has in the latest releases of Windows implemented a Generic QoS that uses RSVP signaling to reserve resources.

The key recommendation contained in RFC2205 [28] is that given the current form of the RSVP specification, multimedia applications run within small-scale networks are the most likely to benefit from the deployment of RSVP. The inadequacies of RSVP scaling may be more manageable within the limits of a smaller, more controlled network environment than in a far-reaching, global IP network. It is certainly possible that RSVP may provide a good service in smaller networks, both in peripheral parts of the Internet and in the private arena, where these issues of scale are far less critical. In that lies the key to successfully deliver QoS using RSVP. After all, the purpose of the Integrated Services architecture and RSVP is to provide a method to offer QoS, not to degrade the service quality.

RSVP scaling considerations also present another important point. As the number of IntServ flows increases in the network, the more resources it will consume. Of course, this can be somewhat limited by defining how much of the network's resources that are available. Everything in excess of this value is treated as Best Effort. When all available resources are consumed, all further requests for QoS are rejected until allocated resources are released. This is similar in functionality to how the telephone system works, where the network's response to a flow request is commitment or denial, and such a service does not prove to be a viable method to operate a data network where services better than Best Effort always should be available.

Another open issue is security, or rather, the lack of security. The resource reservation creates a risk for new kinds of denial-of-service attacks. In addition, RSVP uses keyed MD5 to authenticate the messages [32] and the authentication scheme requires a key-distribution system before RSVP can become widespread. An alternative is to distribute the keys manually.

It is also important to mention that the IETF IntServ working group has been removed from the IETF website, which probably means that the group’s work has terminated. If this is the case, this means that the development of IntServ and RSVP is finished and that there may be doubts about the wide propagation of IntServ.

3.5 Differentiated Services

3.5.1 Introduction

The Integrated Services architecture allocates resources to individual flows. The Differentiated Services (DiffServ) model, however, allocates resources to a small number of traffic classes instead. The DiffServ architecture [2] arose as a counterpoint to the relative complexity and end-to-end nature of the IntServ architecture. It is said that the transition from the Best Effort service model to IntServ was too big, and that something in between would be fine for now.

3.5.2 The Differentiated Services architecture

DiffServ makes a clear distinction between edge routers and core routers based on a concept similar to the Internet end-to-end model, that is, the inside of the network should be simple, and complex functions should be placed at end nodes.

Edge routers are located at network boundaries, and are responsible for giving the incoming packets a Behavior Aggregate (BA). The edge router performs a classification on incoming packets and maintains per-flow states to meter the users' traffic. Based on this, the traffic conditioner marks each packet with a Differentiated Services Code Point (DSCP) given by the admission system. Traffic conditioning is configured to meet the users' contracts, and no signaling is required to provide service differentiation.

Core routers are located inside the network, and are responsible for providing different traffic classes. They always handle traffic as an aggregate and implement a limited set of scheduling classes called Per-Hop Behaviors (PHBs). The DSCP set by an edge router is used to select one of the PHBs. The simplicity of the core routers is what makes the DiffServ architecture to be considered scalable.

The concept of DiffServ is dependent on a fair and rational resource management. A coarse-grained control should work reasonably well as long as the network resources are well provisioned. Figure 18 shows the Differentiated Services Architecture.

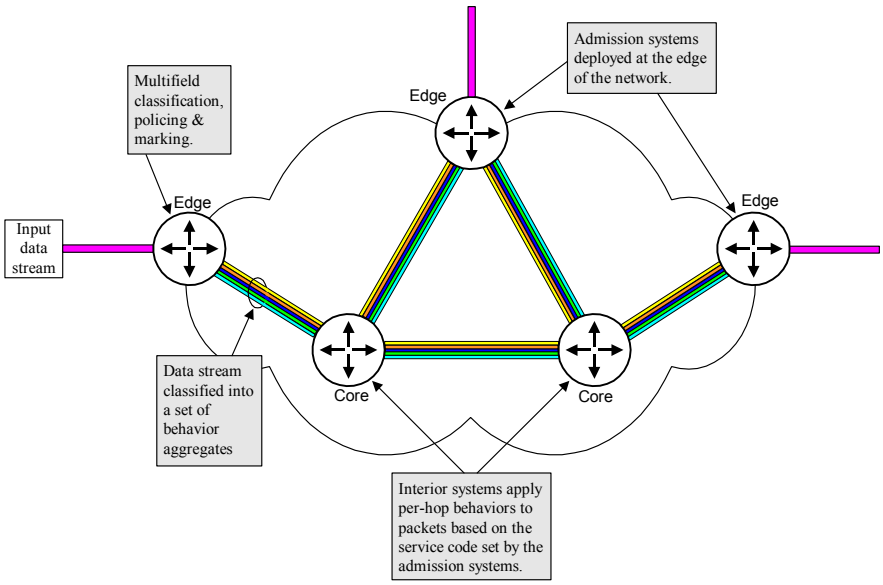

Figure 18. The Differentiated Services Architecture

3.5.3 DS Domain

A DS domain is a contiguous set of DS nodes, which operate with a common service policy and PHB definitions. A DS domain normally consists of one or more networks under the same administration, e.g. an organization's intranet or an ISP. The administration of the domain is responsible for ensuring that adequate resources are provisioned to support the differentiated services offered by the domain.

It is not certain that two neighboring DS Domains use the same PHBs and edge mapping rules. Thus, where an end-to-end path involves connection of DS Domains, called a DS region, it is up to the network operators to ensure that each DS Domain acts in a way that supports the end-to-end QoS goals. Edge routers surrounding a DS Domain are known as DS Boundary nodes with traffic entering a DS Domain at DS Ingress nodes and exiting at DS Egress nodes. Core routers providing a transit service are known as DS Interior nodes.

Figure 19 shows a DS region consisting of two DS domains. The customer Charlie establishes a connection with customer Lima. Charlie has a Service Level Agreement (SLA) with the service provider Orange, but not with Lavender. This example is a worst-case scenario, since Orange in real life would have an agreement with Lavender that provides Charlie good service end-to-end.

Packets sent from Charlie reach the edge router of Orange, functioning as an ingress node for Charlie. The packets are classified after a thorough examination, and the DSCP are marked. Then the packets are forwarded to a core router, known as an interior node, which provides a transit service for Charlie's packets. Now, the packets are classified based on the DSCP given by the ingress node. The packets are rapidly forwarded due to little processing. The packets then reach the border of the Orange domain, and have to be remarked and shaped by the edge router now acting as an egress node. Lavender has no SLA with Charlie, so when the packets arrive at the new DS domain, the packets are conditioned for Best Effort treatment. Therefore, in the Lavender domain, the packets are classified with another Behavior Aggregate (BA) than in the Orange domain.

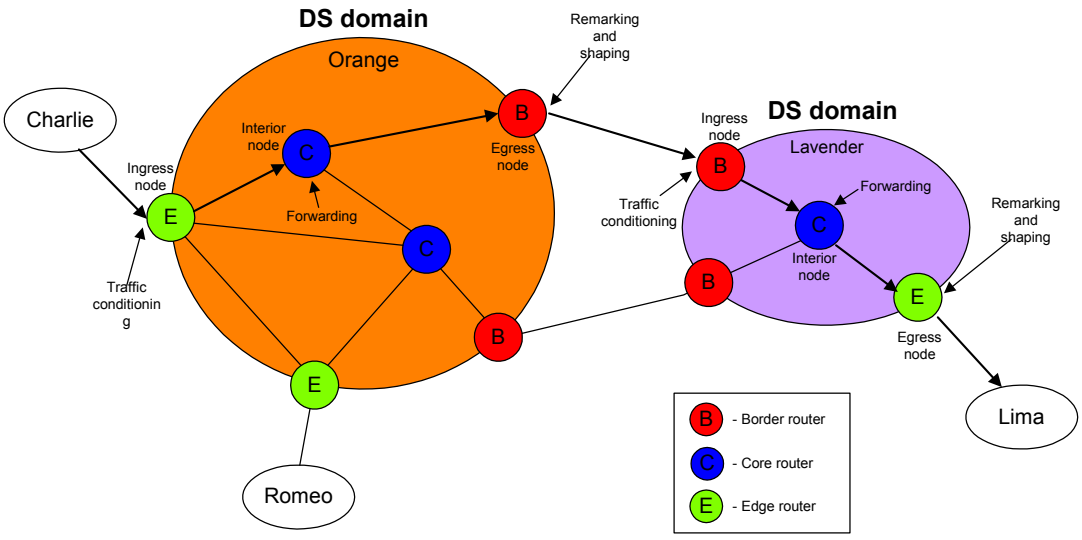

Figure 19. A DiffServ region with two DiffServ domains.

3.5.4 Differentiated Services Code Point (DSCP)

Each packet entering a DS domain is assigned a value called a Differentiated Services Code Point (DSCP). The assigned value is written into the DS field of the packet header. For IPv4, the DSCP field is the six most significant bits of the ToS field in the IP header. For IPv6, it is in the Traffic Class field in the IP header. The DS field is six bits long and the remaining two bits are currently unused.

To separate a small number of classes from each other, we need some way to figure out different characteristics of the packets and generalize them into a set of classes. Within a DS domain many individual application-to-application flows will share a given DSCP. The collection of packets sharing a DSCP is referred to as one Behavior Aggregate (BA), which result in that the core routers need to take only BAs into concern rather than particular flows.

The DSCP can be used to identify 64 different BAs, and IETF defines a small set of standard DSCPs for interoperability among different DS domains. However, a DS domain is free to use non-standard DSCPs within the domain as long as packets are remarked when they leave the DS domain.

3.5.5 Per Hop Behaviors (PHBs)

3.5.5.1 Introduction

A Per-Hop Behavior (PHB) indicates the behavior given by individual DS nodes, and has nothing to do with the end-to-end treatment. DS nodes decide how the forwarding is performed on a per-hop basis according to the DSCP values in packets. The concept of a PHB is to define the minimum requirements for forwarding a particular BA, even though router vendors are still allowed to use proprietary algorithms to realize it.

PHBs specify externally visible queuing, queue management and scheduling characteristics. This approach is intended to allow some freedom in the choice of algorithms in the routers. For example, it does not matter if WFQ, PQ, CBQ or HFSC are used to achieve the desired queuing and scheduling. Some PHBs are defined with closely related behaviors and are referred to as PHB groups. An example may be a set of PHBs that specify the same basic queuing and scheduling behavior, but indicate different drop probabilities to the queue manager.

PHBs are indicated by specific values in the DSCP. A given DS node can support at most 64 different PHBs, since there are 64 different combinations of the DSCP. Although each PHB's definition also provides a recommended DSCP value, DiffServ allows multiple DSCP values to map to the same PHB. PHB groups have multiple DSCP values, typically one for each specific PHB within the group. RFC 2474 provides broad guidelines for allocating particular DSCP values [33].

Best Effort behavior is referred to as the default PHB, which has a recommended DSCP value of 000000. Core routers select the default PHB when they do not recognize a DSCP value.

3.5.5.2 Expedited Forwarding (EF)

One of the simplest PHBs to explain is known as Expedited Forwarding (EF) [34]. Packets marked for EF treatment should be forwarded with minimal delay and loss by the router. The only way that a router can guarantee this to all EF packets, is if the arrival rate of EF packets

at the router is strictly limited to be less than the rate at which the router can forward EF packets.

The rate limiting of EF packets is achieved by configuring the ingress node of a DS domain to allow a certain maximum rate of EF packet arrivals into the domain. A simple, though conservative, approach would be to ensure that the sum of the rates of all EF packets entering the domain is less than the bandwidth of the slowest link in the domain. This would ensure that, even in the worst case where all EF packets converge on the slowest link, the domain is not overloaded and can provide desired behavior.

There are several possible implementation strategies for the EF behavior. One is to give EF packets strict priority over all other packets. Another is to perform weighted fair queuing (WFQ) between EF packets and other packets, with the weight of EF set so high that all EF packets can be delivered quickly. This has an advantage over strict priority: The non-EF packets can be assured of getting some access to the link, even if the amount of EF traffic is excessive. This might mean that the EF packets fail to get exactly the specified behavior, but it could also prevent essential routing traffic from being locked out of the network in the event of an excessive load of EF traffic.

RFC2598 [34] notes that it is possible to meet the scheduling requirements with various scheduling algorithms, but does not specify any particular approach. The main difference between queuing and scheduling mechanisms are the jitter characteristics experienced by a flow. For example, mapping EF with highest priority in CBQ provides lower jitter than mapping EF to WFQ.

Although EF packets in practice will be sent to a specific queue for appropriate scheduling, the definition of EF service is such that on average this queue ought to be small or empty. As a consequence, the EF PHB is suitable for low-loss, low-latency and low-jitter demanding services, such as IP telephony.

By definition packet drops within the network are meant to be rare for the EF service. Individual traffic flows using EF service are rate shaped or aggressively policed on entry to the DS domain. So, correctly configured core routers should never see EF traffic arriving with an aggregate rate exceeding their configured service intervals.

3.5.5.3 Assured Forwarding (AF)

Another PHB is known as Assured Forwarding (AF) [35]. Whereas EF supports services with strict demands to bandwidth and jitter characteristics, the AF group allows more flexible and dynamic sharing of network resources, supporting the soft bandwidth and loss guarantees appropriate for bursty traffic. RFC2597 actually defines four PHB groups, each independently supporting the AF service.

AF has its roots in an approach known as RED with in and out (RIO). RIO is described in section 3.2.6.3 and has the possibility of dropping packets out of profile with a higher possibility than packets in profile.

A combination of a profile meter at the edge of the network that give packets a drop precedence, and RIO in all the routers of the service provider's network should give the customer high assurance that packets within the profile can be delivered. In particular, if the majority of packets are "out" packets, including those sent by customers who have not paid extra to establish a profile, then it should usually be the case that the RIO mechanism will act to keep congestion so low that "in" packets are rarely dropped.

Two distinct classification contexts are encoded within the DSCP, a packet's service class and its drop precedence. The service class assigns the packet to an appropriate queue and, hence, a particular share of bandwidth from the scheduler. The drop precedence sees to make the queue management more or less aggressive depending on whether a packet's drop precedence is high or low. Active queue management on each queue keeps long-term congestion down while allowing short-term burstiness.

Although an AF DSCP identifies one of four queues (classes), it does not specify a queue's maximum size or the scheduler service interval associated with each queue. The network operator configures these parameters on a case-by-case basis depending on the edge-to-edge service desired from each AF class at the time. Each service class has a given amount of bandwidth and queue space it is master of, independent of the other three classes. To prevent possible re-ordering of packets belonging to application flows within a service class, an AF-compliant router must not map different service classes into the same queue and is not allowed to distribute packets belonging to a single service class across multiple queues.

Another approach than RIO, known as weighted RED (WRED) provides more than two drop-probabilities. In the case of assured forwarding, the value of the DSCP field is used to pick one of several drop probability curves, so that several different classes of service can be provided.

Specific drop probabilities for each precedence level are assigned by the network operator to meet the desired packet-loss characteristics for each class. The only requirements are that drop precedence 3 must have a more aggressive drop probability than precedence 2 has, and so on.

As with EF, an actual edge-to-edge service based on an AF PHB group requires coordination between the edge routers and the core routers. The edge routers limit the type of traffic mapped to each AF class and set the drop precedence, while the core routers ensure that appropriate resources and behaviors are provided to each class and drop precedence.

3.5.5.4 Class Selector Per-Hop Behaviors

Limited backward compatibility with the old IPv4 ToS field's Precedence classes [36] is achieved by defining class selector PHBs [33] as those that closely approximate the packet handling of various IPv4 precedence levels. The DSCP values xxx000 corresponds to the three bits of the original Precedence field. A Precedence of 000 results in the default PHB DSCP, and they are semantically and syntactically equivalent with both indicating Best Effort forwarding behavior. A router implementing Class Selector PHBs must implement at least two, and it may implement up to eight distinct PHBs.

3.5.6 Traffic conditioning

A traffic conditioner can perform policing & marking by enforcing the rules of the network administrators at the network node input. In order to deliver proper service agreements, each DiffServ enabled edge router implements a traffic conditioning function that performs metering, shaping, policing and marking of packets. This ensures that the traffic entering a DiffServ network conforms to the Traffic Conditioning Agreement (TCA).

Figure 20 shows the elements of a traffic conditioner. The different parts are described more thoroughly in section 3.2.4.

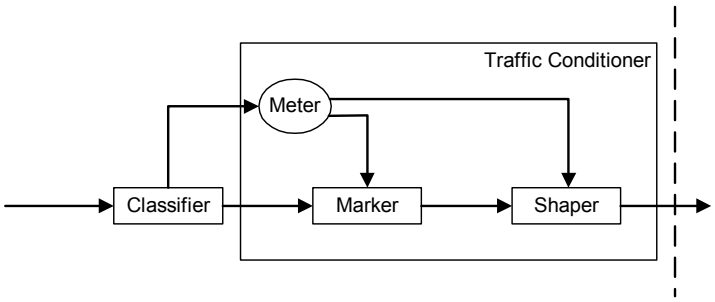

Figure 20. A traffic conditioner.

Some form of multifield (MF) classification is typically applied to establish a packet's context, and from this context the traffic conditioner knows which metering profile to apply, which DSCP to assign and whether to drop or mark the packet. The classification rules and associated actions are specified by one or more TCAs.

Implementations of traffic conditioners are usually not very flexible when it comes to handling packets. For example, EF service only requires a DS ingress node's metering stage to drop packets that are outside a profile represented by a single-stage token bucket. However, such traffic conditioning is not sufficient for AF service, which requires the ingress node to assign DSCP values based on both the classification and the drop precedence. For example, a single-stage meter might mark in-profile packets as AFx1 and out-of-profile packets as AFx2 (or AFx3), instead of dropping them. A multistage meter could differentiate between AFx1, AFx2, AFx3 or local drop. An example of a conditioner is "A two rate three color marker", described in section 3.2.4.

3.5.7 Discussion

We introduced this section saying that DiffServ arose as a counterpoint to the relative complexity and end-to-end nature of the IntServ architecture. DiffServ has minimized the scaling problem and complexity of IntServ performing complex functions at the edge routers, and a simple core network. DiffServ aggregates flows to a limited set of behaviors, to make the forwarding simple.

DiffServ has received a warmer welcome than IntServ, and is by many seen as a solution for the future. DiffServ is simple in replacing IntServ's end-to-end signaling with admission and policy functions at the network edges. In the absence of end-to-end signaling, DiffServ has problems guaranteeing QoS through the network, and the edge routers must carefully perform traffic conditioning to minimize the probability for other QoS-demanding traffic to experience congestion.

By the same reason, DiffServ cannot guarantee that the next DS domain the packets are going to traverse will offer the same QoS. QoS across DS domains is ensured by Service Level Agreements (SLAs) between service providers. However, there is no exchange of policy information between DS domains, and a lack of control of individual flows results in low dynamics during changes in input loads.

DiffServ's architectural simplicity is attractive, but actual deployment of edge-to-edge services requires filling in a lot of unspecified parameter values, and there are few guidelines for network operators to follow. Mapping hundreds or thousands of mostly unrelated microflows to a limited set of BAs requires a careful balancing act, an act that is much dependent on the network's topology at any point in time as it is on the ingress traffic conditioning and interior queuing and scheduling.

One of the benefits from the DiffServ approach is faster core-routers. By limiting their classification and queuing stage complexity and less state to signal, process and store, the packets are handled faster.

Relative to IntServ, a DiffServ network is far less tolerant of individuality among the microflows being carried. Two approaches are available to ensure that individual edge-to-edge services survive being aggregated as they pass through the core:

- Aggressively drop or rate-shape packets at the edges
- Overprovision the core

In the first case, the network as a whole appears fairly intolerant from the perspective of the external traffic sources. In the second case, the network will, on average, be underutilized. With DiffServ at hand, network designers must balance service flexibility against network efficiency.

Another issue relates to backward compatibility with the IPv4 ToS field. One problem during early DiffServ trials in 1999 was that some existing TCP implementations closed any connections where the IPv4 Precedence bits changed after the connection was established. This behavior is entirely compliant with RFC 793 [37], and so the "fault" could not necessarily be attributed to the TCP implementations. However, the IPv4 Precedence bits are the lowest three bits of the DS field. Any DS-based transit network between two TCP peers might well modify the DSCP so that the TCP peers perceive the precedence to have changed, triggering a shut down of the connection. Discussions about updating RFC 793 to say that TCP implementations should ignore the precedence fields have been performed to remove the problem.

DiffServ has its own form of "uncertainty principle". The fewer contexts a core router has about individual packets, the less tolerant the system is to burstiness of microflows within a behavior aggregate. Because core routers no longer see individual microflows, these routers are unable to mediate between aggressive and non-aggressive microflows making up a given behavior aggregate. Ingress rate- shaping bounds the interference between microflows making up a behavior aggregate and smoothes the burstiness of any given behavior aggregate, compensating for the lack of packet context available to core routers.

In principle, a DS domain might reach an agreement with its upstream traffic sources such that the packets arrives with a DSCP already appropriately set. However, even with such an agreement in place, traffic conditioning is still recommended on ingress to the local DS domain. At the very least, policing and shaping ought to be applied to guard against accidental or deliberate attempts to inject traffic outside the allowed profiles.

In some sense, the DiffServ approach represents the middle ground between absolutely minimal intelligence in the network and the rather significant amount of information processing that is required in an IntServ network. One important question is whether the DiffServ approach will meet the requirements of real-time applications. For example, if a service provider is trying to offer a large-scale telephony service over an IP network, the DiffServ technique may be insufficient to deliver the QoS that traditional telephone users expect. It seems likely that yet more options, with varying amounts of intelligence in the network, will need to be explored. The next chapter will introduce some other technologies that may be used in a combination with DiffServ to achieve even better quality for real-time applications.

4 Other QoS technologies

4.1 Overview

From the description and discussion of the service models in chapter 3, we have seen that all the service models have strengths and weaknesses in different situations. To improve the service models, there have been proposed other QoS technologies.

Section 4.2 describes how IntServ and DiffServ can be combined to utilize the good features from both architectures. Then, section 4.3 introduces label switching, known as MPLS, to improve routing efficiency and make traffic engineering possible. Finally, a solution to manage network resources more efficiently is introduced with Bandwidth Brokers in section 4.4. All the QoS technologies are discussed individually.

4.2 IntServ over DiffServ

4.2.1 Introduction

This section describes how IntServ and DiffServ can be combined in order to support the delivery of end-to-end QoS.

As mentioned earlier, both IntServ and DiffServ have strengths and weaknesses. The IntServ model provides a very high level of assurance of per-flow resource management, but has significant problems with scaling in large networks. The DiffServ model however, has a weak approach to resource management, though possessing significant scaling properties. To be able to deliver better QoS than the two service models can supply separately, there have been suggested a combination where IntServ is used in the smaller networks at the edges of the network and DiffServ within the network core.

4.2.2 The IntServ over DiffServ framework

In the framework we present, IntServ is applied end-to-end across a network with one or more DiffServ regions in the middle. The DiffServ regions may, but are not required to, participate in end-to-end RSVP signaling for the purpose of optimizing resource allocation and supporting admission control. Figure 21 shows the IntServ over DiffServ framework.

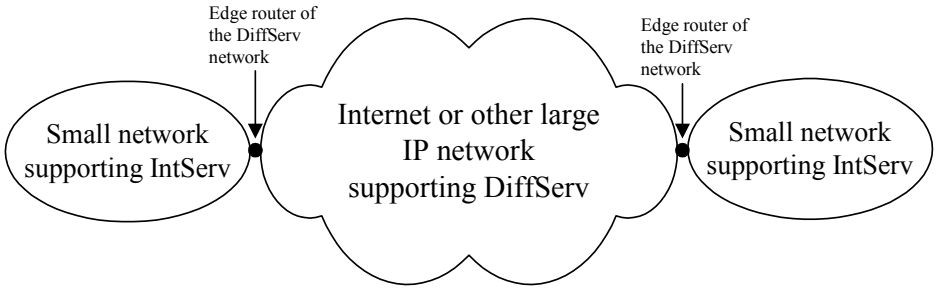

Figure 21. Example of an IntServ over DiffServ network.

Applications using RSVP for resource reservation receives a RESV message to imply that there is established a reservation within the IntServ-capable components of the network. This also implies that the service requirements have been mapped into an aggregate service class within the DiffServ-network. The DiffServ core must be capable of carrying the RSVP messages across the network, so that further resource reservations are possible within the IntServ network when leaving the DiffServ environment. As RSVP messages are transparent to non-IntServ networks, this is not a problem.

An IntServ-supporting DiffServ network uses multifield (MF) admission classifiers, together with a classification based on the IntServ flow specification. The specification is mapped to an aggregate DiffServ class, and the MF admission filter marks all such packets into the associated aggregate service class. It is also possible for the host to condition the traffic before sending it into the network, and to mark it with the associated DSCP values, so that the admission filter can operate as a behavior aggregate (BA) admission classifier.

The IntServ over DiffServ model requires that any admission failure within the DiffServ network are to be communicated to the IntServ network, and from there to the end application via RSVP. This allows the application to take corrective action to avoid failure of the service

itself. If the service agreement for the DiffServ networks is statically provisioned, this static information can be loaded into the IntServ boundary system, and IntServ can manage the allocation of available DiffServ BA resources. If the service agreement is dynamically variable, some form of signaling is required between the two networks to pass this resource availability information across to the IntServ environment.

4.2.3 Discussion

Combining IntServ and DiffServ will provide the strengths of both the service models where it is applicable. IntServ is used to supply QoS both internally in the small IP network as well as ensuring that QoS-dependent traffic are given priority all the way to the edge router of the DiffServ network. In the large network shown in Figure 21, the flow will be treated as specified for the DiffServ network.

By using IntServ only in smaller networks, we avoid the fact that IntServ does not scale very well, yet keeps the advantages that IntServ has to offer. In Figure 21, we assume that the office network has a dedicated link connecting it to a larger network. Once a flow using IntServ reaches the edge router of a DiffServ domain, it is treated with the rights given by the DiffServ network administrators. At the time the flow reaches the IntServ network on the other side of the DiffServ network, it will again use IntServ to supply QoS for the flow.

A well-configured IntServ over DiffServ network uses DiffServ routers that are IntServ compatible, in the sense that they can understand the signaling of RSVP. If a service provider wishes to use IntServ, it will not be a problem to do this in combination with a DiffServ backbone. Internally in the network, they will benefit from all the advantages of IntServ, and externally the DiffServ network understands IntServ and is able to offer resources in accordance to the reservations made in the local IntServ network.

By using IntServ and RSVP in the smaller networks, we can utilize the ability of receiving explicit feedback if the reservation was not completed of some reason. A useful area for the explicit feedback is for real-time applications such as “video on demand”, where the receiver’s link bandwidth and computing ability is essential. By using IntServ and RSVP for this purpose, there will be possibilities for some form of end-to-end QoS negotiation. The use of RSVP does also open the possibility of increasing the utilization of the network by signaling the resources available in the network.

An improvement in the DiffServ core network could be to let the routers be both IntServ and DiffServ capable. Then, applications with a critical demand for delivery-in-time can be allowed to create a reservation path end-to-end with IntServ. Interactive applications with less demand for low delay and non- real-time applications could still be mapped to DiffServ-treatment in the network core.

To avoid creating too many reservations in the core network, strict admission control before letting real-time applications run IntServ end-to-end must be performed. This service could be expensive or only be possible for applications that are not so common, such as videoconferencing.

The next section will deal with label switching, another technology that make IntServ and DiffServ more efficient.

4.3 MPLS

4.3.1 Introduction

MPLS is an abbreviation of Multi-Protocol Label Switching, and the term multiprotocol has been chosen to stress that the method applies to all network layer protocols, not only IP. MPLS is about “gluing” connectionless IP to connection-oriented networks. RFC 3031 specifies the architecture of MPLS [38] and refers to this as the “shim” layer. “Shim” refers to the idea that MPLS is somewhere between layer 2 and 3 and MPLS makes them fit better.

The principle of MPLS is that all packets are assigned a label, and packets are forwarded along a path where each router makes forwarding decisions based solely on the contents of the label. At each hop, the router strips off the existing label and applies a new, which tells the next hop how to forward the packet. Simple forwarding increases the speed of the forwarding process inside the MPLS network, which improves delay and jitter characteristics of traffic.

Network operators establish paths, called Label Switch Paths (LSPs), for a variety of purposes. Examples are to guarantee a certain level of performance, to route around congested networks, or to create IP tunnels for network-based virtual private networks. MPLS has the ability to create end-to-end circuits, similar to a virtual circuit in ATM. It also provides specific performance characteristics, such as traffic engineering across any type of transport medium. These opportunities reduce the need for overlay networks and layer 2 control mechanisms.

4.3.2 The MPLS architecture

The basic concept of label switching is very simple. E.g. let us assume an email message is sent from one user to another. In a Best Effort network, the method to send this email is similar to postal mail. The destination address is examined and this address determines how the email is sent to its final destination [39].

Label switching is different. Instead of using a destination address to make the router decision, a label is associated with the packet. In the postal service analogy, a label value is placed on the envelope as a Zip code and is thereafter used in place of the postal address to route the mail to the recipient. In computer networks, a label is placed in a packet header and is used in place of an IP address, and the label is used to direct the traffic to its destination. Figure 22 shows how an MPLS network works.

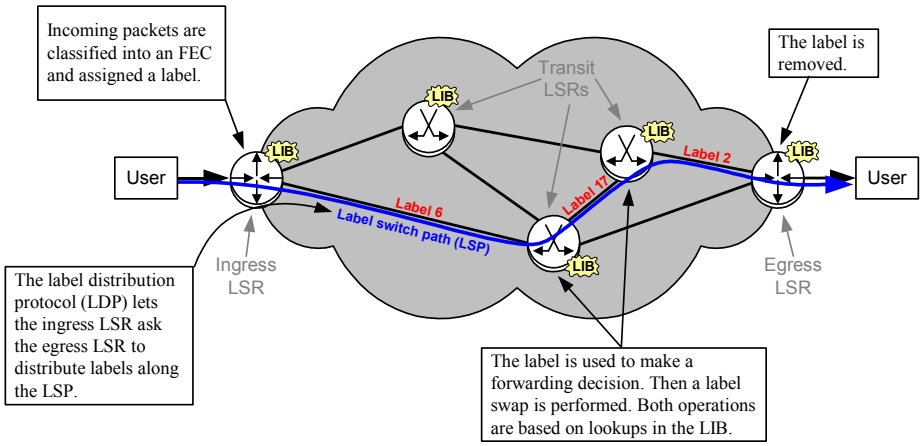

Figure 22. The MPLS functionality.

All the routers supporting MPLS is called Label Switch Routers (LSRs). The ingress LSR is the one by which a packet enters the MPLS network. It adds an MPLS header to the IP packet and assigns a label. If the packet is integrated with a QoS operation, e.g. DiffServ, the ingress LSR will condition the traffic in accordance with the DiffServ rules. The egress LSR is where a packet leaves the network, and the MPLS header is removed from the packet. Both ingress LSRs and egress LSRs are edge nodes connecting the MPLS network to other networks.

The transit LSR, also called an interior LSR, receives the packet between the MPLS edges and uses the MPLS header to make forwarding decisions. It will also perform label swapping.

Labels are small identifiers placed in the MPLS header. As traffic transits the MPLS network, label-swapping tables are consulted in each MPLS device. These are known as the Label Information Bases (LIBs). By looking up the inbound interface and label in the LIB, the outbound interface and label are determined. Then, the LSR can substitute the inbound label for the outbound, and forward the frame. The labels are locally significant only, meaning that the label is only useful and relevant on a single link, between adjacent LSRs. The adjacent LSR label tables, however, should end up forming a path through some or the entire MPLS network, a Label Switch Path (LSP).

The 32-bit MPLS “shim” label is located after the layer 2 header and before the IP header in a packet, and contains the following fields [40]:

- The label field (20-bits) carries the actual value of the MPLS label.
- The Exp/QoS (experimental) field (3-bits) can affect the queuing and discard algorithms applied to the packet as it is transmitted through the network.
- The Stack (S) field (1-bit) indicates the bottom of the stack when label stacking is being used.
- The TTL (time-to-live) field (8-bits) is a copy of the TTL field in the IP header, and is decremented for each hop.

A Label Switch Path (LSP) is a specific traffic path through an MPLS network, and is provisioned using distribution protocols. Packets will be forwarded in the same manner by all the routers belonging to an LSP.

MPLS allows a hierarchy of labels, known as a label stack. It is therefore possible to have different LSPs at different levels in the label stack. As an example consider the scenario shown in Figure 23:

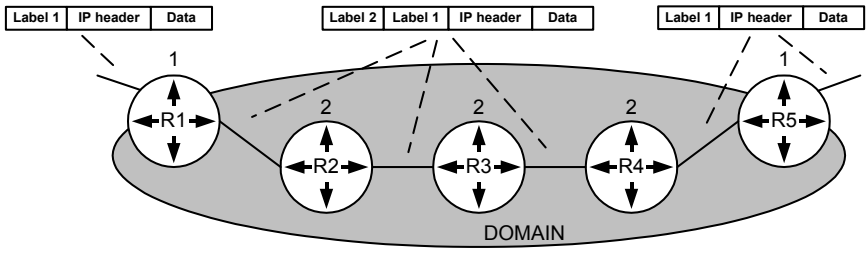

Figure 23. Example of a label hierarchy.

The routers R1 to R5 belong to two different LSPs. The numbers 1 and 2 are the label stack depth. R1 and R5 are border routers and R2, R3 and R4 are the interior routers. For the purpose of label forwarding, R1 and R5 are peers at the border level and R2, R3, R4 are peers at the interior level. When R1 receives the packet P with a label that is 1 level deep heading for R5, it will swap P's label with a corresponding label that will be used by R5. Also since the packet has to travel through R2, R3 and R4, R1 will push a new label, thus the label stack

level depth is now 2. Then we have two LSPs, one is at level 1 from R1 to R5 and the second is at level 2 from R2 to R4.

The path through a label switched network can be determined in two ways. Traditionally, routing protocols such as OSPF or BGP are used to discover IP addresses. Information telling the IP address of the next hop is correlated to a label, yielding a Label Switched Path (LSP). Another approach is to configure the LSP manually based on traffic engineering. Real-time traffic is a prime candidate for traffic engineering.

The Label Distribution Protocol (LDP) [41] is a major part of MPLS, a specification, which lets the LSRs distribute labels to its LDP peers. When an LSR assigns a label to a set of packets, it needs to let the peers along the LSP know about the label and its meaning. LDP establishes an LSP by using a set of procedures to distribute the labels among the LSR peers. Shortly, we can say that the LSRs come to a common understanding by using the label distribution protocol.

The last expression we need to explain is the Forwarding Equivalency Class (FEC). An FEC is a set of packets that will be forwarded in the same manner. Typically packets belonging to the same FEC will follow the same path in the MPLS domain. To assign a packet to an FEC, the ingress LSR may look at the IP header and other information such as the interface on which the packet arrived. A label identifies the FEC to which a packet is assigned. An example of an FEC is a set of packets whose Type of Service (ToS) bits are the same and whose IP destination address matches a particular prefix.

The routing example below illustrates the basic operation of MPLS in support of unicast routing. Using conventional IP routing protocols and LDP, the label switch routers (LSRs) build up routing tables supplemented with labels called label information bases (LIBs). In Figure 24, nodes A, B, F, and G are hosts not configured with MPLS. Node C is the ingress LSR, node D is a transit LSR, and node E is the egress LSR.

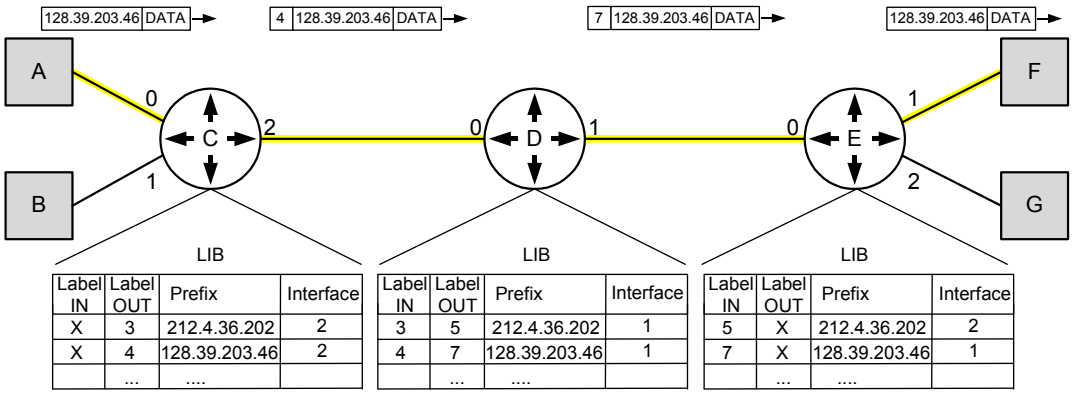

Figure 24. Label swapping and forwarding in MPLS.

LSR C receives an IP datagram from user node A on interface 0, destined for node F. LSR C performs the ingress LSR function, that is, it applies the initial label to the packet after performing a conventional longest-match lookup on the IP header. The datagram is encapsulated behind a label header, and sent to output interface 2. The OUT entry in LSR C's table directs it to place label 4 onto the label header in the packet. After the packet is labeled, subsequent LSRs can forward it using only the label. LSRs usually replace the label on an incoming packet with a new value as they forward it, as we can see from the label values in the LIBs.

At node D, the transit LSR, only the label header is processed, and its LIB is used to swap label 4 for label 7. Egress LSR E is configured to recognize label 7 on interface 0 as its own label, which means that there are no more hops. Notice that the OUT entry in E's table directs LSR E to send this datagram to F on interface 1. This implies removing the label from the packet.

4.3.3 Optical networks and MPLambdaS

Multiprotocol Lambda Switching (MPLamdaS) stands for optical label switching, and the technology concerns routing individual wavelengths (lambdas). Rather than swap labels on a per-packet basis at each hop, lambdas are swapped between the input and output port of optical cross connections (OXCs). MPLambdaS may be the key to unify the control plane for all optical elements, and this can aid in provisioning services and optical paths while enabling IP network fast re-establishment and traffic engineering.

Today, there is a clear desire to use MPLS to make IP fit to optical networks. MPLS is bounded to optical by using the idea that when a route to a specific destination, or group of destinations is propagated, a light path might also be set up. This light path could then be used by packets going to that destination or group of destinations, getting them faster to the destination than if every router or device along the path had to examine the layer 3 header. When a router examines the information on layer 3 the light has to be converted to and from electrical signals. If this has to be done at each step along the way, the result is increased delay in the routers.

4.3.4 MPLS and traffic engineering

An area where MPLS provides benefits to service providers, is in the field of traffic engineering [42]. Traffic engineering refers to the ability of controlling where traffic flows in a network, with the goal of reducing congestion and getting a better service from the available facilities. Traffic engineering is most important in networks where alternate paths exist, such as in large backbones.

A path from one given node to another must be computed, such that the path can provide QoS for IP traffic and fulfill other requirements the traffic might have. Once the path is computed, traffic engineering (a.k.a. constraint-based routing) is responsible for establishing and maintaining the forwarding state along the path.

For example, in the traffic-engineering example in Figure 25, there are two paths from Router C to G. If the router selects one of these paths as the shortest path, it will carry all traffic destined for G through that path. The resulting traffic volume on that path may cause congestion, while the other path is under-loaded. To maximize the performance of the overall network, it may be desirable to shift some fraction of traffic from one link to another. While one could, in this simple example, set the cost of Path C-D-G equal to the cost of Path C-E-F-G, such an approach to load balancing becomes difficult, if not impossible, in networks with a complex topology. Explicitly routed paths, implemented using MPLS, can be used as a more straightforward and flexible way of addressing this problem, allowing some fraction of the traffic on a congested path to be moved to a less congested path.

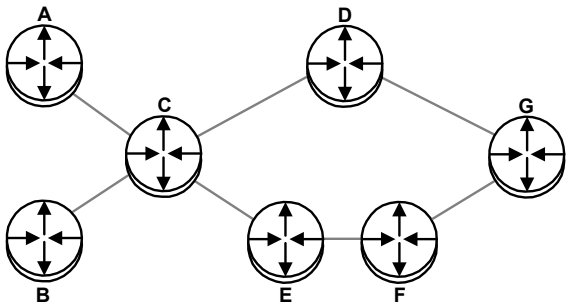

Figure 25. Traffic engineering example [44].

The solution to the traffic engineering problem relies on the fact that labels and label-switched paths can be established by a variety of different control modules, as discussed earlier. For example, with traffic engineering we can establish a label-switched path from A to C to D to G and another from B to C to E to F to G. By defining policies that select certain packets to follow these paths, traffic flowing across the network can be managed.

Traffic engineering requires a lot from network operators. In later releases of MPLS, establishment of traffic engineering policies will be implemented with constraint-based routing. In this environment, the operator only has to specify the amount of load expected to flow between various points in the network (a traffic matrix), and the routing system will calculate the best paths to carry that load and establish explicit paths as a result.

4.3.5 MPLS and IntServ

There is an obvious relationship between the functionality of MPLS and the requirements of the IntServ architecture. IntServ establishes a path end-to-end with RSVP. In MPLS, this path can correspond to an LSP.

The advantage for IntServ routers in an MPLS network is that the reservation state and forwarding directive can be interpreted from the incoming MPLS label, thereby reducing the overhead required to pass the IP header through a potentially very large classification filter. Because MPLS labels and RSVP paths both support unidirectional data flows and can support receiver-initiated path setup actions, there is great similarity between the two concepts. Within such a model, RSVP plays two roles; to set up and maintain the LSP, and to maintain an associated resource allocation and scheduling discipline associated with the virtual circuit.

To make RSVP function in an MPLS, two RSVP objects have to be added. Then, the operation of a unicast path reservation is quite straightforward. When the receiver receives an RSVP Path message, an RSVP Resv message is passed back towards the sender. At the egress point of the MPLS network, the egress router allocates a label and writes this into the RSVP message. At the upstream node, the label is stored as part of the reservation state for this flow. A new label is generated and written into the RSVP Resv message, and the message is passed upstream. In this way, RSVP undertakes the role of a label distribution protocol.

The issue with the use of MPLS LSPs as a mechanism of supporting IntServ reserved paths is relatively unchanged in terms of the scaling properties of this approach. In the interior of large networks, the amount of IntServ flows has the potential to create concentration points at which the number of labels required becomes a scaling problem. [43]

4.3.6 MPLS and DiffServ

The MPLS network can also be combined with Differentiated Services [45]. Behavior aggregates (BAs) can be mapped onto label switched paths (LSPs) in order to best match DiffServ traffic engineering and QoS objectives. The resulting capability to manage traffic flows on a DiffServ BA basis is a very powerful tool, enabling the network to manage traffic flows that share a common service requirement in a single managed unit.

The job of the MPLS network is to select how the DiffServ BAs are mapped in the MPLS LSPs. A DiffServ rule states that packets belonging to the same flow cannot be disordered if they differ only in drop precedence. They must maintain the same order from the ingress LSR to the egress LSR. The effect of this rule is that packets belonging to the same PHB, such as AF21, AF22, and so on, must be placed in a queue that maintains the order of the packets.

It is expected that both MPLS and Differentiated Services will be, and are being, deployed in service providers' networks. DiffServ can support up to 64 classes while the MPLS "shim" label only supports up to 8 classes, which is a problem. There are two alternatives that address the problem, but they introduce complexity into the network architecture. When a frame arrives at an LSR, the Exp/QoS field can be used to determine the queuing treatment, or MPLS can deal with traffic differentiation by storing treatment information as a part of the LIB. When a frame arrives at an LSR in this situation, a table lookup in the LIB determines the queuing treatment.

Combining MPLS and DiffServ provides a very good service differentiation with fast forwarding and aggregation of the traffic at the same time.

4.3.7 Discussion

MPLS is a general-purpose technology that enables a number of new features and services for service providers. Traditionally, IP forwarding is too slow to handle the large traffic loads on the Internet. Even with enhanced techniques, such as a fast-table lookup for certain datagrams, the load on the router is often more than the router can handle. The results may be lost traffic, lost connections, and overall poor performance in the IP-based network.

Label switching is much faster, because the label value that is placed in an incoming packet header is used to access the forwarding table at the router, that is, the label is used to index the table. This lookup requires only one access to the table, in contrast to a traditional routing table access that might require several thousand lookups. Faster packet delivery results in reduced delay and less jitter than with traditional routing operations.

Fast packet delivery is an important aspect of label switching, and processing traffic quickly in an IP network is very important. However, fast service is not all that label switching provides, it also can provide scalability. Scalability refers, in this case, to the ability or inability of an IP network to accommodate a large and growing number of users. Thousands of new users are signing on to the Internet each day. Imagine the task of a router that has to keep track of all these users. Label switching offers solutions to this rapid growth and large networks by allowing a large number of IP addresses to be associated with one or a few labels. This approach reduces further the table lookups, and allows a router to support more users.

The traffic engineering features of MPLS are useful as a way of managing traffic and link utilization in a routed network. It is a major advantage of MPLS compared to IntServ or DiffServ alone. An LSP does not need to follow the shortest path between any two edge LSRs. Although conventional IP routing protocols typically generate shortest-path routes,

external routing algorithms can be used to determine new routes for LSPs that result in more optimal distribution of loads around a network.

MPLS supports the same QoS as IP does. Since MPLS also supports reservation of layer 2 resources, MPLS combined with IntServ or DiffServ can deliver finely grained Quality of Service, much in the same manner as ATM and Frame Relay.

MPLS provides the flexibility to evolve control functionality without changing the forwarding mechanism, thus uniquely positioning MPLS to support the deployment of enhanced forwarding capabilities that will be needed for the Internet to continue its explosive growth.

Most MPLS standards are currently in the draft phase, though several have moved into the RFC standardization phase. An important step was taken April 10, 2001, when Cisco announced a complete solution for end-to-end QoS using DiffServ over MPLS. This is the industry's first QoS-enhanced MPLS traffic engineering solution.

4.4 Bandwidth Broker

4.4.1 Introduction

A Bandwidth Broker (BB) can manage network resources in a DiffServ network. It deals with queries from customers asking for QoS and adjusts Service Level Agreements (SLAs) dynamically according to the input load of the DiffServ network.

Static SLAs tend to be conservative in order to avoid service classes to be congested and result in failure of meeting the guarantees made for the specific class. They result in poor link utilization and no possibility to allow larger throughputs for a short amount of time, even if there are enough free bandwidth available on the link.

DiffServ can provide QoS support for real-time services, but it cannot specify the exact amount of resources a service needs. Van Jacobsen introduced the idea of BBs as part of the Differentiated Services architecture to address this issue [46]. By keeping track of current allocations and available resources, new requests can be handled with concern to the network policies and the available resources both at the edge and core of the domain.

A BB is in charge of both the internal affairs and external relations regarding resource management and traffic control. Internally in a DiffServ domain, a BB may keep track of QoS requests from individual users and applications, and allocate internal resources according to the domain's specific policies. Externally, a BB will be responsible for setting up and maintaining service agreements with the BBs of neighbor domains to assure QoS handling of data traffic crossing its border. By doing so, the edge router can dynamically change its policy in accordance to the state of the core network.

4.4.2 The Bandwidth Broker architecture

The Bandwidth Broker [11] is an agent responsible for allocating preferred services to users as requested, and for configuring the network routers with the correct forwarding behavior according to the defined service. It is associated with a particular trust region, one per domain, and has a policy database that keeps information about who can do what when, and a method of using that database to authenticate requesters. Only a BB can configure the ingress routers to deliver a particular service to flows and this is critical for deploying a secure system.

When an allocation is desired for a particular flow, a request is sent to the BB. Requests include a service type, target rate, maximum burst and the time period when the service is required. The request can be made from a user in the region or it might come from another region's BB. The BB first authenticates the permit of the requester, then it verifies if there exists unallocated bandwidth sufficient to the request. If a request passes the test, the flow is granted access.

The BB configures the appropriate ingress router with the information about the packet flow to be given a service at the time that the service is to begin. This configuration is “soft state”, so the BB will periodically have to refresh this state.

Regularly, the Bandwidth Broker has to configure a group of DiffServ capable routers in order to provide the desired level of service within its region. SLAs change continuously with incoming bandwidth allocation requests from clients, and service classes are reconfigured by sending parameters necessary for policing and marking a flow. A router must be refreshed whenever the Bandwidth Broker register changes in service allocations.

The BB plays several roles in administering resources in its DiffServ region. Two important aspects are Intra-domain resource management and Inter-domain resource management. This is illustrated in Figure 26.

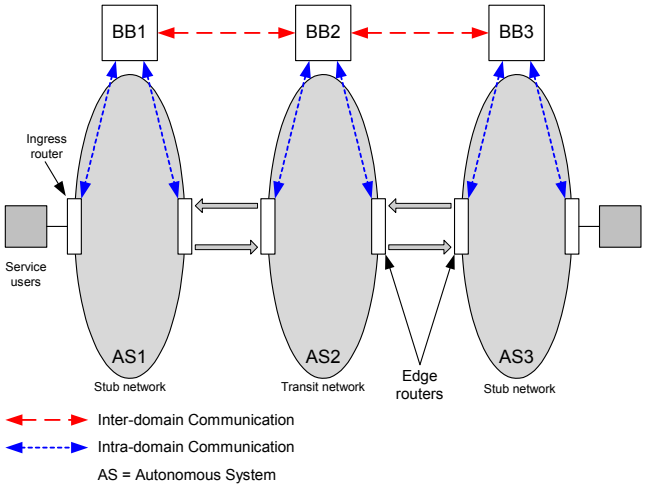

Figure 26. Bandwidth Broker communication [47].

Inter-domain resource management is concerned with provisioning and allocating resources at network boundaries between two domains. The amount and types of traffic at both sides, which a bilateral SLA agrees on transmitting, must be established on the boundary between two domains. Signaling messages are sent between BBs of neighboring domains to request the necessary resources from the network, and to communicate information about the resources required on the links connecting the domains.

In a stub network, a bandwidth broker manages the resources on the links connecting the stub network to its neighboring transit networks. It also provides information such as how to appropriately set the DSCP field of packets before forwarding them into a particular DiffServ-enabled transit network.

In a transit network, the BB keeps track of the DiffServ traffic that enters and leaves the domain across its boundaries, making sure that the bilateral agreements with adjacent domains are met. The BB communicates with the ingress and egress border routers to configure traffic conditioners within the routers, according to the bilateral agreements.

When a BB receives an inter-domain resource management message, the message contains two things: Information about an aggregate flow entering the domain at a particular ingress, and the DSCP requested for the flow. If there are any existing aggregate flows between the ingress and egress, the flows are updated with the egress profile flow information. The BB keeps track of the DS traffic that enters and leaves the domain across its boundaries, making sure that the bilateral agreements with neighboring domains are met. The BB communicates with the ingress and egress border routers to configure traffic conditioners, according to the bilateral agreements.

Intra-domain resource management deals with allocation of resources within a DS domain. The SLAs are negotiated with customers in the stub networks, and bandwidth allocation requests are granted or rejected depending on availability of resources. Information about the DSCP to be set is also passed on to the ingress router as well as the setting of classes to the egress router.

4.4.3 Discussion

Theoretically, the introduction of BBs in DiffServ networks will guarantee QoS for real-time applications, as well as it will increase the utilization of the network. With BBs, the edge routers can dynamically change their policy in accordance to the state of the core network and this will result in a service more similar to IntServ, but without the explicit reservations.

The solution seems quite promising, but not without complications. Increased complexity in the network can result in reduced scalability. A Bandwidth Broker must perform many tasks per query, as well as handling all the queries in the domain. Signaling and reservations per query is also a concept that from earlier experiences scales poorly.

To make the solution using BBs work, the domains should be kept at moderate sizes, avoiding a lot of small domains, which needs more signaling. Only real-time applications should be allowed to make reservations, reducing the number of requests and signaling in the BB regions. The remaining traffic should be handled without end-to-end signaling, thus the stress on the BBs are kept low.

5 Testing the service models

5.1 Overview

So far, we have discussed the service models theoretically. In this chapter, we will test the service models in order to strengthen our perception of their characteristics. We have studied two mathematical models that visualize real-time qualities of a network well.

Section 5.2 deals with the test setup, describing the equipment that we use and the testbed architecture. A validation of IntServ and DiffServ support in the testbed can be found in appendix A.

We have performed some simple tests to improve our understanding of the testbed and the behavior of the service models. The results are presented in section 5.3 and in appendix B. A mathematical model to find the power of a network is described in section 5.4. The section also discusses tests results concerning the efficiency of the service models. Fair resource allocation is also defined by a mathematical model, and we use it to measure the service models' fairness in section 5.5. All the test results are thoroughly discussed.

5.2 Testbed description

5.2.1 Introduction

This section describes the equipment that we use in the testbed and the topology of the testbed. We start with explaining the routers chosen and the software that generates and measures traffic. Then we describe the testbed architecture.

5.2.2 Routers

The testbed must provide similar conditions for all the three service models. Therefore, we use the same testbed-architecture for all the tests, but change the router software that realizes the different service models.

Changing only the software in a router to support another service model is very easy, and since the expenses are less than buying commercial routers, we choose to use software routers for the testbed. The software we have chosen in the routers is based on a FreeBSD operating system and a QoS forwarding mechanism named Alternate Queuing (ALTQ) [59]. ALTQ implements several queuing, scheduling and queue management mechanisms, and it supports both RSVP signaling and DiffServ code point marking. As small disfavor is that IntServ's Guaranteed Service is not supported by ALTQ. Help for configuring the routers are placed in appendix D.

5.2.3 Generating and measuring traffic

The testbed is not connected to the Internet, and will initially not have any traffic other than control traffic generated by the routers itself. To provide traffic, we must connect hosts that generate all the traffic we need to make the tests meaningful. For this purpose we need software that can generate traffic from the following criteria:

- The software must run on Windows 2000, since we will use our workstations as hosts.
- Traffic on the network must be able to be repeated in the exact same manner infinitely many times.
- The traffic generated shall be of such characteristics that it will emphasize the factors we want to study closely.

The goal in a QoS-enabled environment is to enable predictable service delivery to certain classes or types of traffic, regardless of other traffic flowing through the network at a given time. Software that measures traffic in the testbed must be able to recognize particular flows and report how the flows are treated. The treatment can be indicated by four measurable properties: End-to-end latency, packet loss, jitter and throughput experienced by a packet.

We have chosen to use an application at the hosts that both generate traffic and measure it. The software is recognized by many known test labs, and is very expensive. Luckily, we got an evaluation version for our test period. The name of the software is NetIQ Chariot [56], and it can drive traffic classified with Best Effort, IntServ and DiffServ techniques. A console controls what kind of traffic endpoints at the hosts shall generate, and the endpoints report back to the console about the traffic behavior with a view to QoS. Special traffic patterns can be made, executed and stored for later use.

A simple way to classify traffic is by its data rate, and a Chariot endpoint can control how fast data is sent. It can also provide realistic traffic emulation, e.g. there is an application script

that emulates the audio and video portion of NetMeeting 2.1. Chariot fulfills all the criteria listed above, and is a very user-friendly tool.

5.2.4 Other measuring tools

In addition of measuring the traffic behavior on the edges of the network, we need a tool to study traffic at the routers. We have chosen Tele Traffic Tapper (ttt) for this job, since it is capable of graphical traffic monitoring in real-time. The graphs are updated every second by default.

In some situations, we also might want to look into packets at a protocol level. Microsoft Net Monitor is used for this task.

5.2.5 Testbed architecture

QoS handling is imperceptible during light loads, and QoS effects can generally only be observed against heavy traffic (stress conditions). Therefore, we must focus on making the testbed in such a way that there will be a lack of bandwidth in the network. Before we start the testing, we perform a validation of IntServ and DiffServ support in the routers. The result of the validation is that the routers function as intended, and it is shown in appendix A.

We are going to focus on four factors that may be used as QoS measurements for real-time traffic. These factors are throughput, delay, jitter and packet loss. We are going to stress the testbed network to the limit of its available bandwidth and measure the different service models' QoS features.

The use of software routers may result in more delay than using ordinary routers. Specialized routers are manufactured with the purpose to serve only as routers, therefore the bandwidth on the memory busses, specialized processing and so on will result in more efficient routing. The increase of delay that may occur using software routers are not necessarily a negative feature for our testbed. Delay only makes the differences between the measurements more visible.

5.2.6 The bottleneck testbed

It is common in a network to have a certain link with a limited amount of bandwidth that is not possible to route around. This testbed has such a bottleneck in the network path. Today, bottlenecks in IP networks are very common. As an example, it will often occur in the transitions from the network backbone to the service providers outside the backbone. Backbone networks are built with high bandwidth throughout the network, but the networks outside have less bandwidth.

As a scenario, we can imagine that Microsoft releases a new OS that a lot of people request. Microsoft will probably have a high BW connection to the backbone network. If a lot of people in Norway with high-BW links all start to download this OS at the same time, the local service provider will set up FTP connections for all the transfers. The large number of connections will be a problem for the local service provider in Norway. Figure 27 can visualize this.

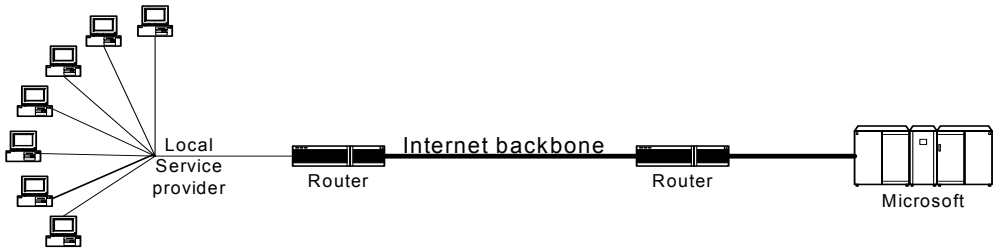

Figure 27. A possible bottleneck in an IP network.

As we can see from the illustration, there will be a problem getting all the file-transfers on the service provider's link at the same time. This will result in congestion, increased delay and packet drop in the routers along the path.

The testbed we implement simulates this scenario by setting a relatively low bandwidth on the link between the two routers, shown in Figure 28.

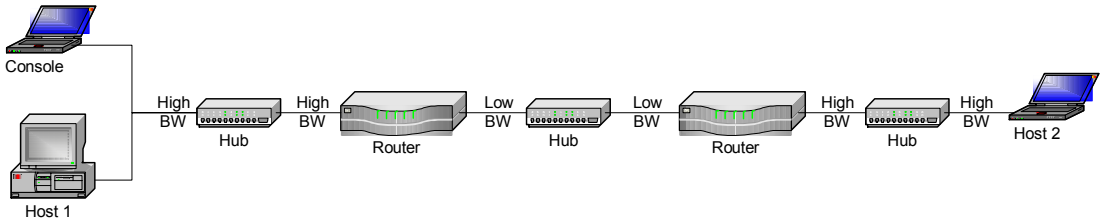

Figure 28. The testbed scenario.

5.3 Preliminary tests

5.3.1 Introduction

The majority of test results concerning QoS-behavior in a network can be observed only during extensive traffic load. To find the characteristics of a service model, we have to test and monitor what the end-users see since they are the ones that will experience the changes inside the network.

We start to perform some simple tests that we can learn from and improve our understanding of the testbed and the behavior of the service models. We look at response time and throughput in a silent network, then packet loss and jitter during streaming in an overloaded network. The overload conditions for the three last tests are described in appendix B, together with more tests. Among other things, the tests in appendix B show that bi-directional traffic not is supported in the testbed, and that differences in scalability not are possible to measure. The appendix also describes some assumptions we had to make before performing the preliminary tests.

5.3.2 The best possible response time in a silent network

Response time is a measurement that reflects the user's experience with a network. The objective is to see if IntServ or DiffServ adds noteworthy processing delay for the traffic in a silent network. The creditl script (Credit Check, Long Connection) can be used to determine response time. It sends a 1-byte file from Endpoint E1 to Endpoint E2 and asks for a confirmation. Using this script and TCP as the network protocol, we determine the best possible response time through a network consisting of two routers interconnected by a 10 Mbps link.

Table 2. The best possible response time.

	Best Effort	**IntServ BE flow**	**IntServ Cntl flow**	**DiffServ**
Mean Response time	0.83 ms	0.94 ms	5.10 ms	0.90 ms

As this is a test performed on an empty link, we expect to receive the best response times possible for the network. Table 2 shows that the flow in the Best Effort network has the lowest response time, which is as expected for a response test of such a network. Response times for a Best Effort network that forward traffic First In, First Out (FIFO) is lower than a network with sophisticated queuing. The increased delay is caused by more complexity in the routers such as the conditioner in DiffServ and Class Based Queuing (CBQ) in both IntServ and DiffServ.

From Table 2 we can see that IntServ Controlled Load has a relatively high response time compared to the other services. As we can see from the citation from RFC2211, we have measured delay values that are above expected.

"The transit delay experienced by a very high percentage of the delivered packets will not greatly exceed the minimum transmit delay experienced by any successfully delivered packet. (This minimum transit delay includes speed-of-light delay plus the fixed processing time in routers and other communications devices along the path.)" [29]

We have found that the high delay is introduced at the hosts, since we get the same delay in a network consisting of only two hosts with a hub in between. To control the Controlled Load

service from the endpoints, NetIQ's Chariot uses Generic QoS implemented in Microsoft Windows 2000. Peter Frame, an employee of NetIQ claims that the delay introduced, is owed by a slow TCP/IP protocol stack in Generic QoS.

The response time test is performed in a 10 Mbps network to avoid problems with increased response time due to limited bandwidth. In some of the following tasks, we will experience that the response time increases when the available bandwidth is reduced.

5.3.3 The maximum possible throughput in a silent network

Throughput numbers tell the rate at which traffic can flow through a network. A network that allows higher throughput allows data to be delivered in a shorter amount of time. Throughput is a measurement that reflects the capacity of a network and is usually measured in bits per second.

The objective is to see how throughput changes in a silent network with different classification of the traffic. We test with all the Service Models and all their possible classifications of flows. To properly measure throughput, we need to use a script that sends a large file to make sure there is sufficient data to fill the pipe. The best script for this is filesndl (File Send, Long Connection), where a 100 KB-file is transferred between Endpoint E1 and E2, followed by a confirmation. TCP is used as the network protocol, and the network is now limited between the routers with a 1 Mbps bottleneck link.

Table 3. The maximum possible throughput.

	Best Effort	**IntServ**		**DiffServ**					
	with FIFO	**BE**	**Cntl**	**BE**	**EF**	**AF1**	**AF2**	**AF3**	**AF4**
Allocated bandwidth [Kbps]	1000	600	400	460	200	70	140	50	80
Measured throughput [Kbps]	891	577	385	454	193	71	139	51	80

The throughput test summary in Table 3 shows the average measured throughput, and it provides a good indication of the achievable network throughput. Our traffic monitor (Chariot) measures only the payload of a packet as throughput, referred to as "Goodput". This means that Chariot does not consider packet overhead. It is therefore quite normal that the measured throughput is lower than the allocated bandwidth.

We use a 10 Mbps link end-to-end that is limited to 1 Mbps between the routers by using two different techniques; Dummynet [48] for the Best Effort service model and ALTQ for IntServ and DiffServ. Dummynet provides strict limiting of the bandwidth, which means that the traffic can never exceed the bandwidth specified. ALTQ on the other hand, has bandwidth limitations that are not so precise.

Best Effort throughput is affected by packet overhead and the fact that TCP pulls back every time packet loss is detected. Packet overhead alone limits the possible throughput with 2.7 %.

When we measure throughput in an IntServ network using Controlled Load, we have to reserve the bandwidth we intend to use before we measure it. Our experience is that Chariot reserves more bandwidth than we specify in the TSpec. In order to measure the maximum throughput, we had to tune the reservation in the Tspec to acquire the highest possible bandwidth. We also had to verify that the flow was treated as Controlled Load, to make sure that it was not refused and treated as Best Effort. By running both Best Effort and Controlled Load measurements at the same time, we could look after throughput degradations in the Best Effort flow to see if the Controlled Load flow was rejected.

The test result from the DiffServ network shows some interesting aspects with the service model. We can see that throughput of the EF class lies beneath the allocated bandwidth, while the AF classes lies just above the allocated bandwidth. Traffic in an EF class above a permitted bitrate is dropped, due to strict policing for the traffic in this class. Therefore, the TCP flow classified as EF will have an actual payload below the allocated bandwidth. However, AF traffic out of profile is marked with higher drop precedence, not immediately dropped when the flow exceeds the allocated bitrate. This allows the flow to push the bandwidth limit of its class.

Note that we are using TCP flows in our throughput measurements. TCP uses a payload of 1460 bytes, and the overhead is 40 bytes. This will have a marginal effect (2,7 %) on the throughput measured by Chariot. However, a NetMeeting audio RTP packet has 44 bytes of data and 40 bytes of overhead. This is an overhead of 47,6 % and we must be aware of that the throughput measured by Chariot does not reflect the actual load on the testbed network in this case.

5.3.4 Streaming performance of VoIP in a congested network

An important measurement for real-time applications is the amount of lost data. Lost data is important, because lost data in a VoIP flow means degraded voice quality. How much data that can be lost varies with the type of compression used before sending the data.

The goal of this task is to study how different quality levels of the service models affect the quality of voice in case of VoIP traffic. A 64 Kbps VoIP flow is transmitted through the 1 Mbps bottleneck network, and packet loss and jitter are measured. We run a VoIP script with RTP together with other traffic to measure the streaming performance of the different service models. The results are presented in Table 4 with the percentage of bytes that is lost during streaming.

Table 4. Streaming performance of a VoIP flow.

	Best Effort	IntServ BE flow	IntServ Cntl flow	DiffServ BE flow	DiffServ EF flow	DiffServ AF2 flow
Mean Lost Data	10.013 %	6.240 %	0.066 %	24.885 %	0.000 %	0.000 %

Data loss has several causes. For example, the data rate may be higher than the maximum throughput potential of a network, or packets can be dropped in the routers, due to bursts of traffic. Table 4 shows that Best Effort cannot avoid packet loss for real-time traffic. However, IntServ and DiffServ treat real-time traffic with high priority, and there are almost no bytes lost. The few bytes lost with IntServ are in the beginning of the transmission before the RSVP reservation is established.

We notice that the DiffServ Best Effort flow has a very high percentage of lost data compared to the other Best Effort flows. The reason is simply that DiffServ's Best Effort class in this case has less bandwidth available than pure Best Effort and IntServ's Best Effort class. See Table 3 for bandwidth allocations.

5.3.5 Jitter introduced in a congested network

The objective is to see if different service models yield different jitter patterns visible for the end-user. Below we see three graphs where a VoIP flow runs through a Best Effort, an IntServ and a DiffServ network. The graphs are taken from the measurements performed in subsection 5.3.4.

An interesting observation before evaluating the graphs is that mean jitter not is what we usually associate with the term jitter. Mean jitter is the way we can see it, defined by Chariot in the same way as mean delay. In this case, the interesting thing with a jitter graph is the variations from the mean value. A large difference between the mean value and the maximum tell us that a router treats streaming traffic very unreasonably. Since there are ways of avoiding jitter in routers, a good service model should have a smooth jitter graph.

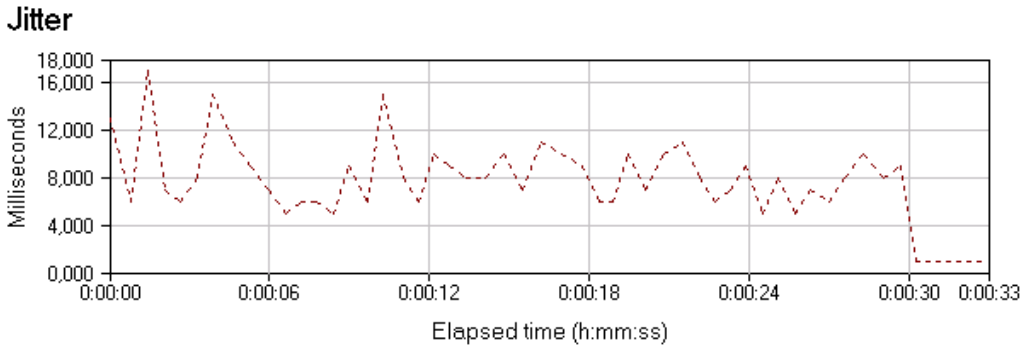

Figure 29. Measured Best Effort jitter performance (mean 8.208 ms).

Figure 29 shows variations in the delay introduced by the Best Effort network. We can see that the delay varies from 5 to 17 ms, with a mean of 8.2 ms. The maximum value is more than the double of the mean delay, and this may be a problem if the mean delay gets ten times higher. It is obvious that this situation is not acceptable for a real-time flow in a large network.

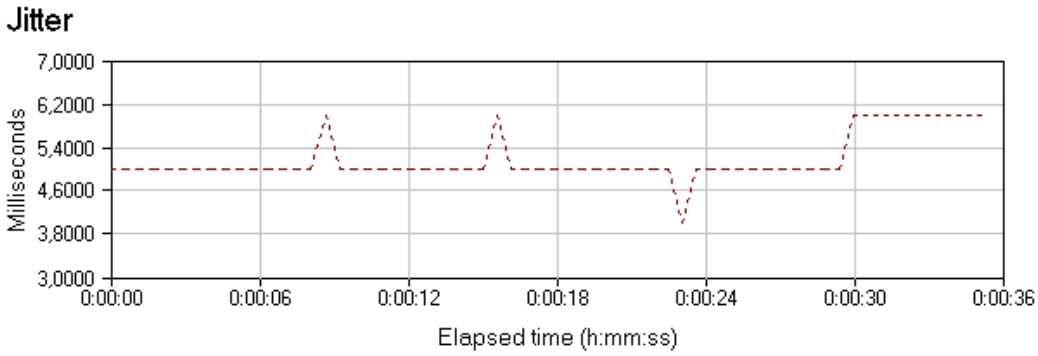

Figure 30. Measured IntServ Controlled Load jitter performance (mean 5.170 ms).

The jitter for a flow supported by IntServ's Controlled Load is shown in Figure 30, and the graph is not showing much variation in delay. The difference between mean delay and maximum delay is 16 %, and this is very good even when the mean delay gets much higher.

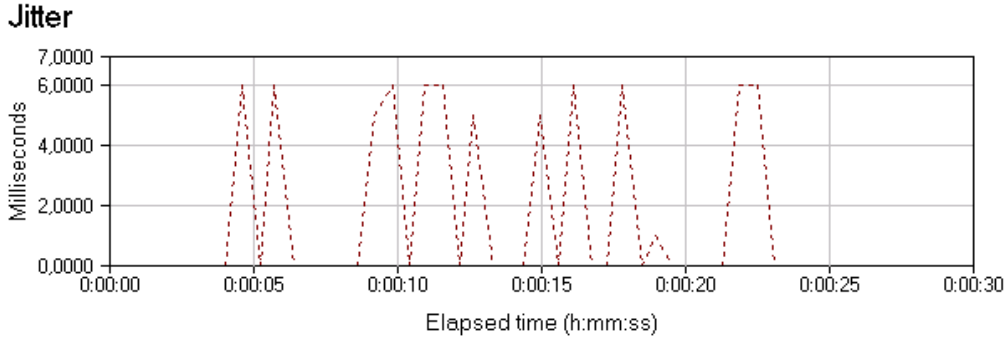

Figure 31. Measured DiffServ EF jitter performance (mean 1.321 ms).

The jitter graph from the EF class (Figure 31) shows high variations in delay. The difference between mean delay and maximum delay is as high as 354 %, and this is a very bad sign for what will happen if the mean delay gets higher.

The strange behavior of the EF class was explained to us by Kenjiro Cho, the maker of ALTQ. Apparently, the traffic in the EF class gets repeatedly "suspend and resume" by the CBQ scheduler. This is caused by a mismatch in the burst tolerance of the EF class and the EF conditioner. The EF conditioner seems to allow larger bursts than the EF CBQ class can accept. To avoid this problem, we can assign more bandwidth and larger "maxburst" to the EF class to give some headroom to absorb short bursts.

Another important thing to know is that just classifying traffic with EF treatment does not provide less jitter. In fact, CBQ does not have a mechanism to explicitly control delay or jitter. We should use HFSC instead for this purpose. The reason why we use CBQ is the wish of having the same queuing and scheduling mechanisms for both IntServ and DiffServ. It should probably also be mentioned that HFSC is a more complex queuing and scheduling mechanism than CBQ.

5.3.6 Real-time traffic during high internal strain

We will now see what happens with IntServ's Controlled Load service when the service class is put under high strain. The results will be compared with similar situations for DiffServ's AF and EF PHBs.

A situation with high strain on a premium service might occur if the network not has any kind of access control and users may access such services without control. We repeat the previous task with an additional filesndl script. The script sends a TCP flow classified for premium treatment at a rate similar to the premium services' available bandwidth to enforce high strain on the service.

Table 5. Real-time traffic during high internal strain.

	IntServ Cntl flow	**DiffServ EF flow**	**DiffServ AF2 flow**
Mean Lost Data	0.058 %	27.938 %	25.096 %

Even a service with explicit reservations will degrade if the resources are not wisely allocated. Table 5 shows that both the AF2 and EF flows now are degraded to a Best Effort service.

The IntServ Controlled Load flow keeps its promises during high strain. However, not all flows trying to get a premium service will get it, it is only the flows that ask for a reservation when there still is bandwidth available that will have their request granted. This feature of the Controlled Load service is exactly the same as with traditional telephony. When all the lines are occupied, we have to wait for a line to get available.

5.4 The power of a network

5.4.1 Introduction

The power of a network is defined by one of the inventors of the Internet, Leonard Kleinrock. It combines two basic QoS performance metrics to find the efficiency of a network and to compare it to other systems. We intend to utilize the power function to systematically visualize, and compare the performance of the service models. Power is expressed by the following formula:

$$Power = \frac{Throughput}{Delay} \tag{1}$$

If we calculate the power for different loads, we will have the ratio of throughput to delay as a function of load. The graph will give a good indication of a network's efficiency, and the optimal load a service provider should try to achieve for the network. Before we focus on assumptions and limitations for the testbed and the testresults, we will give an introduction to the power function based on an article by Christos Kolias and Leonard Kleinrock [49]. This will hopefully make the interpretation of the test results easier to understand.

5.4.2 Power as a performance and comparison measure

To introduce the power function we consider a communications system where packets arrive, receive service and depart. Let the input load applied to the system be λ ($0 \le \lambda \le 1$). The following fundamental system performance characteristics are defined:

- throughput, $\gamma = \gamma(\lambda)$. The rate at which packets arrive at the receiving host.
- mean delay (response time), $T = T(\lambda)$. The average time spent by a packet while it resides in the system.

It is often the case that a system cannot reach its highest capacity, thus not yielding a 100% throughput. This means that the system reaches its maximum throughput at input loads where $\lambda < 1$. Exceeding that point of input load drives the system to overloaded situations, i.e. delay and queue length grow to infinity. Below we show an example of a throughput-input load function

$$\gamma = \begin{cases} \gamma(\lambda), & \lambda < \lambda_s \\ \gamma_{\max}, & \lambda \ge \lambda_s \end{cases} \tag{2}$$

where λ_s represents the maximum input load where the system is stable, and γ_{max} the maximum throughput achieved. Thus, in order for a system to be stable we require that $\lambda < \lambda_s$. The above example shown graphically in Figure 32 refers to an ideal no-loss system.

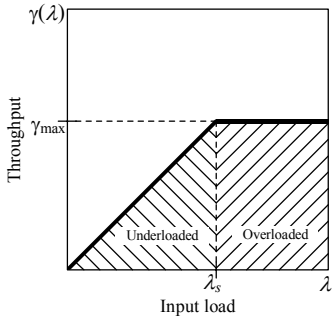

Figure 32. The ideal throughput characteristic of a system.

In a non-ideal system, throughput normally degrades for $\lambda > \lambda_s$, unless some type of flow control is implemented. As λ approaches λ_s, the system becomes saturated. It is then apparent that λ_s is a critical value, and being aware of it is a key point in a throughput analysis. Put in another way, being aware of when we have spent all the available capacity in a network is very important.

Figure 33 shows a mean delay profile of a communication system. The minimum mean delay T_{min}, measured in a system is found when $\lambda \to 0^+$. As $\lambda \to \lambda_{max}$ the delay goes to infinity.

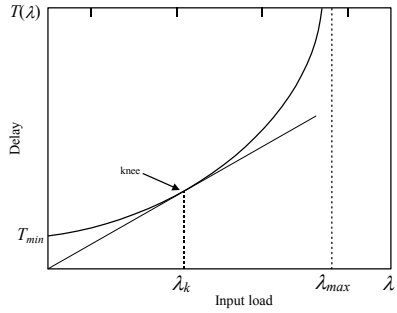

Figure 33. The delay characteristic of a system.

Power can be used to identify the operating point where a network delivers its best performance. Power does this by combining two "competing" performance measures, namely throughput and delay.

As we increase the applied input load, throughput increases, but delay becomes higher as the system tends to saturate. Thus, a tradeoff has to be made between high throughput and low delay. The notion of power proves to be useful in addressing this issue. It appears as a natural measure for characterizing real-time traffic such as video and speech, whose efficient transmission requires simultaneously high throughput and low delays.

The input load λ in a network is made unitless, with 0 meaning no input load, and 1 meaning the maximum bandwidth λ_s in a network. The power of a no-loss system is then defined as the ratio of the system's throughput γ over the system's mean delay T.

$$P(\lambda) = \frac{\gamma(\lambda)}{T(\lambda)} \tag{3}$$

Note that $0 \le P(\lambda) \le 1$. The throughput factor γ will never exceed 1, since the input load is unitless. The lowest measured delay T never gets below 1, since it is not measured a delay below 1 ms in this test, and we use milliseconds as the unit for delay-measurements. As Figure 33 shows, the load at which power is maximized, namely λ_k, can be found by taking

the tangent to $T(\lambda)$ from the origin. This tangency point is referred to as the "knee" of the curve.

From these conclusions, we can say that the larger the power, the higher the "performance" of the system is. As we have indicated, another useful feature of the power function is that it is very attractive as a comparison measure. We use the power function not only to observe how one service model performs, but also to compare the different service models with each other.

5.4.3 Assumptions and limitations

From the preliminary tests and during the test itself we have gained knowledge to how the testbed works and what measurements that are difficult to accomplish. In this subsection we will discuss how we perform delay-measurements and the composition of traffic that we put into the network. As the routers used in the testbed are for experimental use, we had to specify the configuration in the router ourselves. This makes the routers work more or less like commercial routers, but there might be differences concerning the implementations of these specifications.

Delay-measurements during this test are performed measuring roundtrip time for a TCP packet with 1 bit payload. By using TCP for the delay-measurements, we have as little amount of overhead as possible as well as we avoid too much retransmissions of packets since TCP backs off during congestion. However, with TCP we measure response time, which is twice the delay that real-time traffic perceives. This will in reality make the delay for real-time traffic better than measured in these tests. Numbers shown in the tests are not important, it is the relations between the numbers and the trends they emphasize that really matters.

Using TCP as the transport protocol might seem as a contradiction as we are focusing on real-time flows. However, we are interested in finding the response time of the available bandwidth in the network, and if we use UDP, this flow will force TCP flows to back off. By using TCP also for the delay-measurements we avoid this.

The delay-measurement performed by Chariot is repeated at least 2500 times during a test run, and the results measured can by this reason be used as reliable measurements. Nevertheless, there are variations in test results due to the order that flows are initiated and stochastic behavior in the network. E.g. a control packet generated by a router may cause a packet drop in another router, and force a TCP flow to decrease its window size. By studying the power graph, we can evaluate trends in the graph and decide if any of the test results have to be rejected and repeated.

Chariot measures only the rate at which user data is transferred, leaving the overhead data out of consideration. This is often referred to as "Goodput" and may affect a test's throughput readings. From monitoring the delay-measurements in a router we experience that this flow is occupying about 90 Kbps when the link is idle. This flow has 1 byte of payload and 40 bytes of overhead. As a result, the flow has a "Goodput" of 2.1 Kbps. As the delay measured on the network is strongly related to available bandwidth, increasing delay indicates that the throughput of the delay-measurement flow is decreasing.

To find the delay for different service classes, we assign one delay-measurement flow to each class. This flow will, with enough resources available, occupy about 90 Kbps. There will arise a misalignment in the total load of the different service classes, since DiffServ uses a total of six classes, IntServ two and Best Effort one class. A delay-measurement on an idle DiffServ network will occupy 12.7 % of the available bandwidth. We must remember that especially the measurements with DiffServ will suffer from this fact, but our experience is that we can still see a representative sample from the test.

With a bottleneck link at 1 Mbps, small classes in DiffServ get too small even to let the delay-measurement flow have its required bandwidth. We performed a test using 1 Mbps and learned that DiffServ achieved unfair results because of the lack of bandwidth. As Best Effort and IntServ have larger classes, we decided to increase the total bandwidth for this test to 4 Mbps to make DiffServ behave in a more sensible manner. Except from the increased bandwidth and different sizes of classes in DiffServ, the testbed are the same as in the preliminary tests.
To create the background load we want, we use one rate-limited TCP flow per class. By limiting the bandwidth on this flow, the delay-measurements will, as mentioned above, reflect the available bandwidth on the link as well as congestion in the routers. The load on the network are evenly distributed between the classes, which means that e.g. if we are running a test with 40 % of the total load (1.6 Mbps of 4.0 Mbps) on an IntServ network where Best Effort traffic gets 60 % of the bandwidth, 960 Kbps will be used by the Best Effort flow and 640 Kbps by Controlled Load.

The same approach is taken for DiffServ. However, this is not quite in accordance to the specifications of DiffServ. For instance, the EF class shall by definition never be overloaded. As this is a test to explore the limitations of the service models, we will however load all DiffServ classes equally and up to 100 %.

We have already mentioned that the delay-measurement flow occupies approximately 90 Kbps. With IntServ, this bandwidth has to be reserved prior to transmitting the flow. Combining this with the fact that Chariot does not measure flows with large overhead correctly made us come up with an alternative solution: To be able to measure Controlled Load correctly we increase the total bandwidth to 4.1 Mbps. This additional 100 Kbps is assigned to Controlled Load, and is, during all tests with IntServ, reserved for the delay-measurement flow. In this way, Controlled Load will never have more than the 40 % (1600 Kbps) available for other reservations, thus the results will reflect Controlled Load's qualities very well.

The sizes and bitrates of the classes are set according to what we assume reasonable. For IntServ we specify 60 % for Best Effort and 40 % for Controlled Load. In DiffServ, the Best Effort class gets 25 % while EF, AF1, AF2, AF3 and AF4 each get 15 % to avoid having too small classes.

To measure the power of a Best Effort network with FIFO queuing, we use the built-in routing and forwarding functions of FreeBSD together with Dummynet to limit the bandwidth from 10 Mbps to 4 Mbps. For IntServ and DiffServ we use CBQ for queuing and scheduling as well as for limiting the bandwidth to 4 Mbps. By using CBQ on both service models we avoid differences in how e.g. congestion is handled, which would influence the result of the measurements.

5.4.4 Test results

5.4.4.1 Introduction

The testing itself is straightforward. We set up the endpoints to generate traffic given by the predefined set of scripts. Then we run the Chariot console to monitor the traffic, and collect the measured throughput and delay. The results for Best Effort, Integrated Services and Differentiated Services are now presented.

5.4.4.2 Best Effort

Finding the power curve for Best Effort gives us a good reference before evaluating IntServ and DiffServ. Since Best Effort first of all is meant as a reference, we are not going to discuss it as thorough as IntServ and DiffServ.

We expect that the results will show that Best Effort are functioning very good at lower loads, but that the lack of traffic differentiation will result in fast degradation of the service delivered when congestion is starting to occur.

As we can see from the power curve in Figure 34, Best Effort is starting to show signs of congestion at 30 % input load. From this point the performance is slowly increasing until it reaches 50 % where the network is having a serious collapse. We can also see this by studying the mean delay in the network (Figure 35). We can see a clear resemblance between this curve and the curve in Figure 33. The delay measurements from 80 to 100 % are removed due to very high delays for these loads.

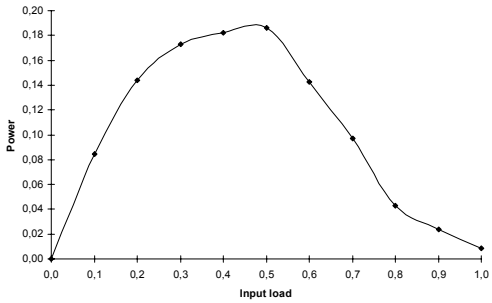

Figure 34. Computed power for Best Effort.

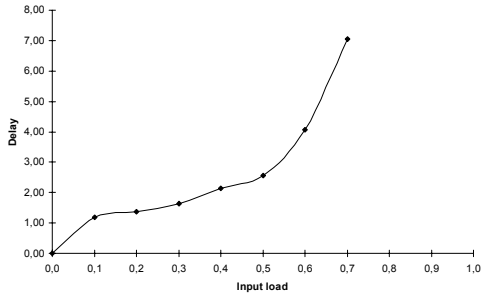

Figure 35. Measured delay for Best Effort.

The use of Dummynet to limit the bandwidth may introduce a delay, due to more processing in the routers. Dummynet probably makes the power of the Best Effort curve somewhat lower than what can be expected in real-world environments.

However, the trend from Figure 34 shows that Best Effort is very good for real-time traffic on lighter loads up to 30 %. The optimal load is around 50 %, but with higher loads the efficiency decreases rapidly and the delay for real-time traffic increases to a non-acceptable level. This is caused by lack of traffic differentiation, which results in large queues in the routers and therefore high delay.

5.4.4.3 Integrated Services

As mentioned earlier, we are using CBQ for both IntServ and DiffServ. CBQ offer lower delay for small packets, which results in better real-time performance and a stronger power curve given for Best Effort classified traffic. From Figure 36 we can see that Best Effort are contributing with a relatively high share of power compared to the Controlled Load class.

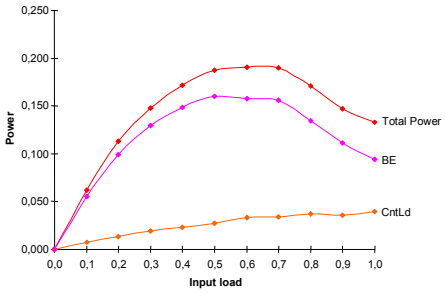

Figure 36. Computed power for IntServ.

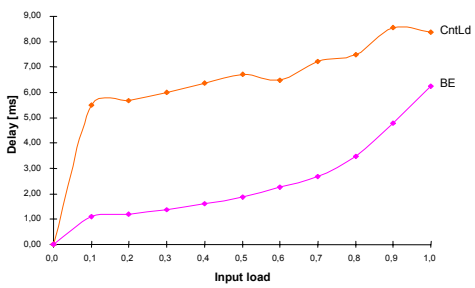

Figure 37. Measured delay for IntServ.

One important thing to notice before evaluating the results of IntServ is the high delay of Controlled Load shown in Figure 37. We have contacted several mailing lists and other resources to find an answer to this abnormal delay, and are told by Peter Frame, which is an international SE of the NetIQ Corporation, that this behavior probably is caused by Microsoft's implementation of Generic QoS and delay in the protocol stack.

As a consequence of the abnormal delay in Controlled Load, an increase of bandwidth for this class would result in a decrease of the total power of the IntServ graph.

By studying the graphs we can see that the total power of IntServ is increasing more slowly compared to the Best Effort network. This can be explained by the fact that the Best Effort class in this test holds 60 % of the total bandwidth and this is limiting the effect for the total power. When we combine this with the fact that Controlled Load has unexpected high delay, it is believed that the power would increase more rapidly at the beginning.

The Integrated Services power graph is steadily rising until 50 % load where the graph levels out. The optimal load occurs at 70 % load where the power is 0.190. From this point the power is decreasing, but not as rapidly as with the Best Effort network. Again, the mix of Best Effort and Controlled Load can explain this. Since we use the same flows for all the tests, the load on the Best Effort class itself is not as large as with the Best Effort network where no queuing and scheduling is enabled. Even when the Best Effort class is starting to get congested, the Controlled Load class' power graph continues to increase and is not significantly affected.

We have not discussed Guaranteed Service that IntServ offers. The software in the routers we have used does not support this service, and we refer to section 3.4 describing IntServ theoretically for more information about it.

Without thinking of Guaranteed Service, we see a lot of advantages using Controlled Load for real-time traffic. One interesting observation with the throughput graph of Controlled Load, shown in Figure 38, is that it is very similar to the ideal throughput graph shown in Figure 32. If the input load increases above the point where the system is stable (input load > 1.0) new flows requesting the Controlled Load service will be rejected, so the throughput will flatten out and the delay will still be kept low. The power graph for Controlled Load, shown in Figure 39, shows an approximately linear graph instead of the bell-shaped graph of a Best Effort power graph.

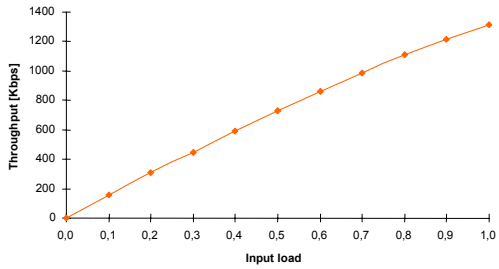

Figure 38. Measured throughput for Controlled Load.

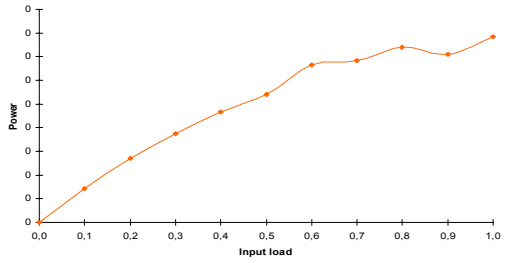

Figure 39. Computed power for Controlled Load.

The similarity with an ideal throughput graph is the result of IntServ's use of reservations prior to sending the data flow. Reservations eliminate the possibility that the link will allow more flows than it can serve properly and this result in less delay in the routers. For our testbed this means that the Controlled Load class is not able to allocate more bandwidth than the 1.6 Mbps it originally was granted.

5.4.4.4 Differentiated Services

DiffServ maintains QoS in a less strict manner than IntServ without any explicit reservations. Thus, DiffServ has the possibility of allowing more traffic in each class than IntServ. By this reason, it is reasonable to assume that we will experience more congestion in a DiffServ network than an IntServ network.

By studying the graphs in Figure 40 and Figure 41 we can see that, in the same way as with IntServ, the Best Effort class contributes with the largest share of power to the network. This results from the fact that Best Effort is granted 25 % of the bandwidth while the other classes only have 15 % and a higher drop precedence for the packets.

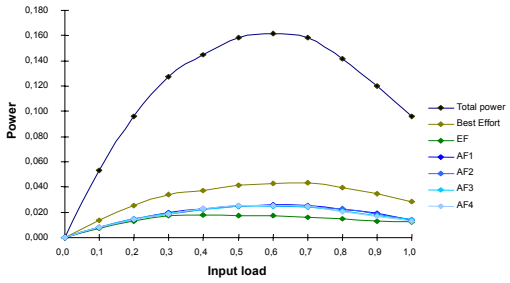

Figure 40. Computed power for DiffServ.

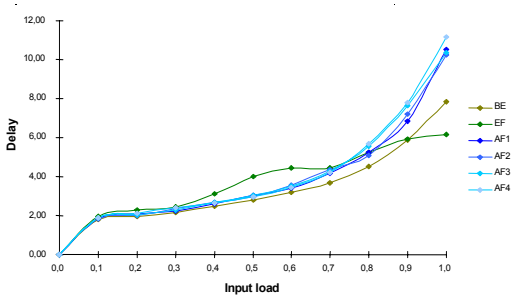

Figure 41. Measured delay for DiffServ.

The AF classes, which all have 15 % of the total bandwidth, are behaving much like expected in the way they are following each other in both delay and power. This indicates that all flows are getting their share of the network. Opposite to IntServ, the resources in DiffServ are better distributed in the way that each class contributes to the total power curve. This means that they are not receiving increased delay due to end-node static delay. The optimal load is located at 60 % input load and there is no indication of congestion collapse.

An interesting, yet disturbing observation is that the EF class, which ought to deliver a better service than the AF classes, delivers the poorest power graph. As we can see from the delay graph, the delay is starting to increase at 40 % load. After this “leap” the delay is normalizing until it during high load delivers the best delay of all the service classes. The reason is that the EF class does not allow a flow to have more than half the available bandwidth. As we run a simple testbed with relatively small bandwidth, we use only one EF flow to avoid stressing this class, which will result in improved delay at greater loads. This shows that the policy given by the administrators of a network have great influence on the power delivered from the network.

The reason for the “leaping” behavior of the EF class is a combination of using TCP for delay-measurements and a token bucket to rate-shape the flow. Unfortunately, while token buckets metering the EF flows have several relevant features, it has been shown that a token bucket must be extremely deep in order to perform well with TCP [50]. The problem is that TCP is ACK-triggered. That is, the sender usually does not send a packet unless it receives an acknowledgement from previously transmitted data. As TCP awaits the receipt of an acknowledgement to send more data, the token bucket being used to police the source can easily overflow and tokens can be lost. TCP is inherently bursty, sending packets in bunches with significant gaps between bursts.

One way to solve this problem is to use extremely deep token buckets. For large token buckets to work, the depth of the bucket must be large enough to accommodate the largest gap in the acknowledgement scheme. Unfortunately, large token buckets increase the burstiness of the EF flow. By allowing large bursts of EF-marked packets into the network, queue

management schemes at intermediate routers must allocate a significant amount of buffer space to accommodate bursts in priority packets. While this can be done, allocating such a large amount causes the buffer to be significantly underutilized.

Although the EF class has the least powerful graph, it offers low delay to real-time traffic, as long as the class is not congested. This conclusion agrees with the requirements set by IETF for this class. RFC 2598 say that the EF class never should be overloaded [34], so the service providers have to dimension this class to avoid this problem.

The DiffServ power curve suffers from our approach of filling all the classes with equally heavy load. It also suffers from having more traffic load during the test than the other service models caused by more response time tests occupying bandwidth.

5.4.4.5 The service models compared

By comparing all the service models' power graphs together, we see that IntServ and DiffServ traffic achieves a better overall performance than Best Effort. However, with the IntServ model, the optimal load is obtained at 10 % higher input load than with DiffServ. This may be preferable during conditions with heavy traffic. Best Effort has a lower optimal load, but a steeper power graph for lower input loads. Therefore, choosing a suitable design depends on the applied load at which we would like the network to operate.

Comparing the power functions in one chart makes it easy to point out differences, as well as positive and negative sides of each service model. As we can see from Figure 42 and Figure 43, the graphs indicate different behavior.

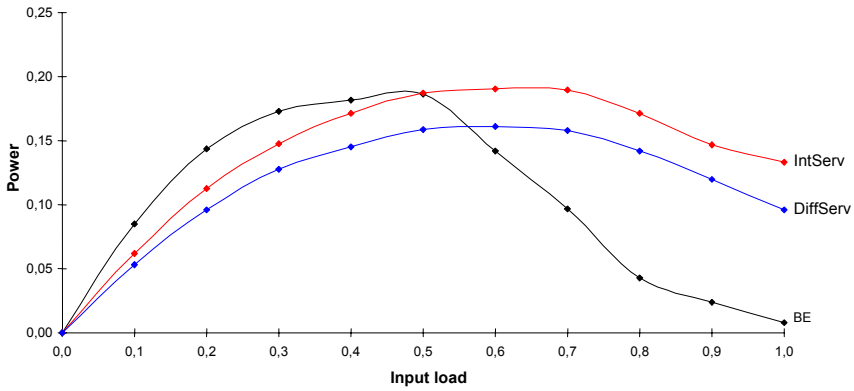

Figure 42. The power graphs compared.

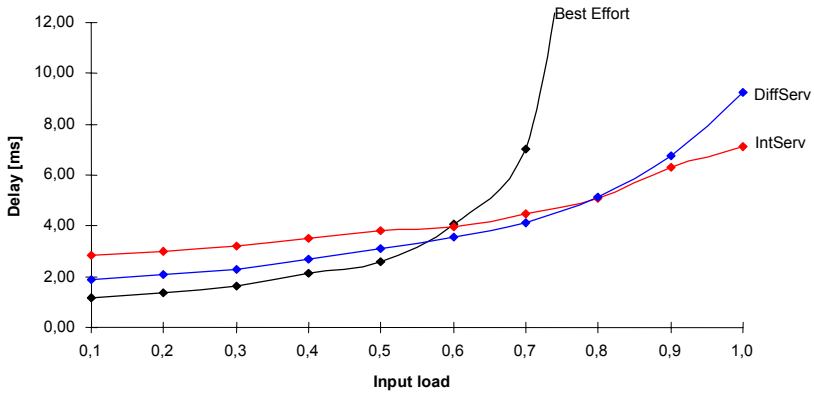

Figure 43. The delay graphs compared.

We can see that the Best Effort network is the best choice for networks that are low or moderate loaded, that is, below 50 % input load. It has a steeper increase in power and the delay is in average 0.63 ms lower than for DiffServ. This is a relatively small difference in delay, but as we are running the testbed with only two routers, the difference might be clearer with more routers. For loads over the optimal load of the Best Effort network the congestion in the network makes the power decrease dramatically due to the exponential increase in delay. With maximum input load the delay is as large as 110 ms, making Best Effort unsuitable for networks running real-time services.

With input loads from 50 % and up to 100 %, the IntServ network has the strongest power graph. From 50 to 80 % input load the delay is marginally poorer than for DiffServ, but from 80 % and up to maximum input load, IntServ is better. The reason for IntServ's strength is that it operates with only two traffic classes; IntServ Best Effort and Controlled Load. The advantage by having larger classes is that there is more available bandwidth in each class and this provides a lower measured delay. For comparison it can be mentioned that with 100 % input load on the network, DiffServ utilizes its 1 Mbps Best Effort class with 89 % "Goodput" while IntServ utilizes its 2.4 Mbps Best Effort class with 98 % "Goodput".

The advantages of the power function derive from its simplicity and ease of implementation and understanding, while powerful in its ability to capture the overall performance of a system. We have demonstrated that power is extremely useful, not only as a leading performance index, but also as a comparison tool.

5.5 Fair resource allocation

5.5.1 Introduction

Efficient utilization of network resources is not the only criterion for judging a resource allocation scheme. We must also consider the issue of fairness to find out if one of the service models act more fair than the others. In this section, we will see differences in how fair and efficient the service models treat each flow in a network.

Defining exactly what constitutes fair resource allocation is not very easy. There are always different kinds of traffic in a network with different needs, and what is fair to one kind of traffic may not be fair to another kind.

Raj Jain has proposed a metric that can be used to quantify fairness of different network mechanisms. Given a set of flow throughputs (x_1, x_2, …, x_n) the following function assigns a fairness index to the flows:

$$f(x_1, x_2, ..., x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n\sum_{i=1}^{n} x_i^2} \quad (4)$$

The fairness index always results in a number between 0 and 1, with 1 representing greatest fairness. The index is applicable to any resource sharing or allocation problem.

We start with an introduction to the fairness concept based on an article by Raj Jain [51]. Then we describe the testbed and what assumptions we make, before we look at the measured results analytically and show the efficiency and fairness of the three service models.

5.5.2 Fairness as a performance and comparison measure

5.5.2.1 Desired properties of the fairness index

The fairness index has several desirable properties. First of all, the index is independent of the population size. Therefore, it is applicable to any number of flows, as distinct from variance or standard deviation, which are statistical measures and require a large number of flows to be valid.

Scale independency makes it easy to understand the fairness index. The formula applies equally well to all metrics of raw throughput, response times, or their ratios (power). The fairness index has no units and regardless of the throughput unit used, the value always remains the same.

The fairness index is bounded between 0 and 1, and it can be expressed as a percentage between 0 and 100%. Most people find it very easy to understand a statement like “The scheme is 70% fair”. The fairness index has a very direct relationship to the fairness. As the fairness goes up, so does the index, and it does this in a continuously manner. Even a slight change in a flow’s allocation is reflected as a change in the index.

Using the fairness index as the fairness measure makes it easy to explain to non-statisticians. It results in intuitive values for several simple cases. For example, if the bandwidth is divided among contending flows such that it is divided equally among 80% of the flows while the remaining 20% of the flows are starved, the fairness index comes out 80%. This applies for any other percentage as well.

Based on the above arguments, we feel comfortable using the fairness index as the quantitative measure of fairness in comparing service models.

5.5.2.2 Selection of a key performance metric

Notice that the fairness index is denoted as $f(x)$. The parameter x indicates that the fairness based on another allocation metric would be different. The choice of the metric depends upon the application. Examples of metrics appropriate for different applications in networking are listed below:

- Response time is generally the performance indicator for interactive traffic.
- Throughput is the key performance parameter for file traffic.
- If the traffic consists of both file and interactive traffic, the fairness may be based on the ratio of throughput and response time, which is known as power.

The selection of a key performance metric for the testbed is based on the traffic we want to study. First of all, we want to measure how well real-time traffic like IP telephony is transferred. IP telephony is an example of interactive traffic, thus we choose response time as the performance indicator x.

However, it is also interesting to see what happens with non-real time applications when the different service models treat real-time traffic with priority. Therefore, we will also discuss fairness with throughput as the key performance parameter.

5.5.2.3 Definition of the fairness index

We already have introduced a quantitative measure called the fairness index. The index measures the "equality" of a user allocation x. If all users get the same amount, i.e. all x_i's are equal, then the fairness index is 1, and the system is 100 % fair. As the difference increases, fairness decreases and a scheme that favors only a selected few users has a fairness index near 0. Now we go a little further into the definition of the index and show how we can calculate fairness for the service models.

A user's perception of fairness can be found by rewriting the proposed fairness index to:

$$f(x) = \frac{1}{n \cdot x_f} \sum_{i=1}^{n} x_i \tag{5}$$

x_f is the "fair allocation mark" computed as follows:

$$x_f = \frac{\sum_{i=1}^{n} x_i^2}{\sum_{i=1}^{n} x_i} \tag{6}$$

Thus, each user compares the allocation x_i with the amount x_f and perceives the algorithm as fair or unfair depending upon whether the allocation x_i is more or less than x_f. For the i-th

user, the algorithm is x_i/x_f fair. The overall fairness is the average of the perceived fairness of all n users.

Consider an example where 2 Mbps are distributed among 10 flows where 4 flows receive 500 Kbps each and the others nothing. The 4 users that received 500 Kbps each consider this to be 100% fair while the 6 users who didn't receive anything perceive this to be 0% fair. The overall fairness is 40%. All users with $x_i>x_f$ are said to be "favored" users and those with $x_i<x_f$ are "discriminated" users.

We use equation 6 and 7 to find differences between how flows are treated by the service models. What we till now have called a user, we now call a flow instead, and we can see e.g. if an AF flow is treated more fair than an EF flow in DiffServ. But before we do that, we have to know a little more about the parameter x_i, which is the fairness indicator.

First we consider fairness as a function of response time. As the response time increases compared with other flows, the fairness experienced decreases. We have a given network configuration with n flows, and find the minimum response time T_{min_i}, which is the lowest measured response time for the i-th flow. We also have T_{min}, which is the lowest value of all T_{min_i}'s.

We want the largest possible value of $x_i(T_i)$ to be 1, i.e. when $T_i=T_{min}$, equation 8 must give $x_i(T_i)=1$. $x_i(T_i)$ shall decrease with increasing T_i, and finally reach 0 for $T_i\rightarrow\infty$. $x_i(T_i)$ decreases when the difference between T_i and T_{min_i} increases. The equation sees to that a flow with a high T_{min_i} not is favored compared to a flow with a low T_{min_i}, because it is the difference between T_i and T_{min_i} that matters, not the amount T_i has increased with. T_{min} is only a constant that scales x_i to be a value between 0 and 1.

Now if a scheme results in a response time T_i, we define that its fairness can be found taking a ratio of:

$$x_i(T_i) = \frac{T_{\min}}{T_{\min} + T_i - T_{\min_i}} \tag{7}$$

Equation 8 can also be explained with an example. We study a scenario with a Controlled Load flow and a Best Effort flow in IntServ. The initial delay of the Best Effort flow is 1 ms, with an exponential decrease to 6 ms at a given input load. The Controlled Load flow has an initial delay of 5 ms, and a linear increase to 10 ms at the same input load. Computing $x_i(T_i)$ gives 1/6 for both flows. However, if we take T_{min_i}/T_i, which is the same approach as in equation 9, $x_i(T_i)$ for Best Effort is 1/6, but for Controlled Load it is 1/2.

Assume we have the same given network configuration, and want to consider fairness as a function of throughput instead. The throughput of the i-th flow is γ_{s_i} under maximum optimality. Now if a scheme results in a throughput γ_i, instead, its fairness can be measured by first taking a ratio of:

$$x_i(\gamma_i) = \frac{\gamma_i}{\gamma_{s_i}} \tag{8}$$

An ideal scheme will give all flows their optimal allocation and all x_i's will be equal to 1. However, any real scheme will result in ratios slightly different from 1 and its fairness is given by the following fairness index:

$$f(x) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n\sum_{i=1}^{n} x_i^2} \tag{9}$$

Example: Suppose it is determined that the minimum response time T_{min_i} for three flows are 9 ms, 12 ms and 16 ms. A response time measurement for a congested network results in 12 ms, 15 ms and 25 ms respectively for the three sources. The scaled response times for the sources are 9/(9+12-9), 9/(9+15-12), 9/(9+25-16) or 0.75, 0.75 and 0.5. Substituting these in the fairness index, we get:

$$f(x) = \frac{(0.75 + 0.75 + 0.5)^2}{3 \cdot (0.75^2 + 0.75^2 + 0.5^2)} = 0.970$$

The scheme is 97.0 % fair.

5.5.3 Assumptions and limitations

In order to test fairness, we measure throughput and delay, which are the factors determining the fairness performance we wish to study. More tests concerning fairness are placed in appendix C, such as a study of the EF class versus the Controlled Load Service with an increasing number of real-time flows. The results from the test in appendix C show that all flows exceeding the capacity of Controlled Load do not receive this service. However, the EF class distributes its resources among all the flows, something that deteriorates the response time for high loads.

When we compare the fairness of the service models, we assume that the measurements are performed with exactly similar conditions. At the same time we exclude initial differences between classes by letting each class have its own minimum value as reference.

When evaluating the fairness index, there is one important factor to remember. For all graphs below, the total fairness will never exceed one. However, if one of many flows is experiencing more fairness than the total, the fairness of this flow will exceed one. It is also important to read the explanations for each index, as this will make it a lot easier to understand what happens.

To compute the graphs, we use the measurements and results from the power test. The procedure used to create the following graphs is identical for both delay and throughput. The graphs computed with response time as the performance metric concerns how real-time traffic perceives fairness, while the graphs computed by using throughput concerns non-real-time flows.

5.5.4 Test results

5.5.4.1 Introduction

First, we evaluate fairness by looking at each service model separately, and see how fair the internal classes are treated compared to each other. With this approach, Best Effort has a fairness of one, independent of the input load, since it only is compared with itself. Fairness for IntServ and DiffServ are presented in the next two subsections. Finally, we look at fairness when the service models are presented in proportion to each other.

5.5.4.2 Integrated Services

For Figure 44, the graph on the left hand indicates the fairness of IntServ as a function of delay and the one on the right as a function of throughput.

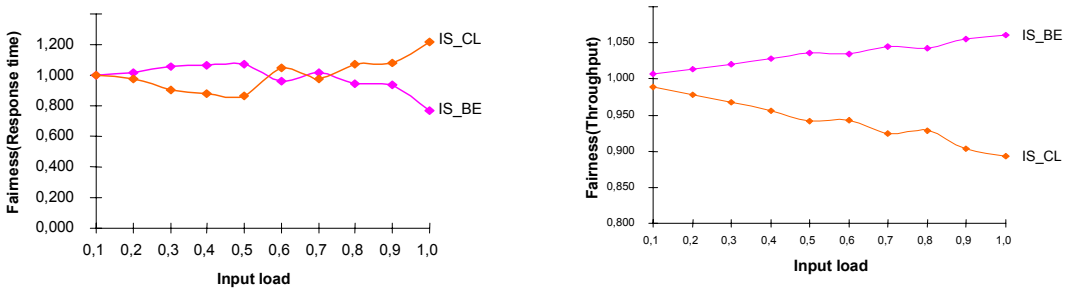

Figure 44. Computed fairness for IntServ.

For real-time traffic however, Controlled Load has less increase in delay for high input loads than the Best Effort class. Controlled Load has more increase in delay for light loads than the Best Effort class, and as a consequence of this experience less fairness.

The reason for the fluctuating values in the graph for input loads between 0.5 and 0.8 in Figure 44 is a variance in the delay measurements performed for Controlled Load. The variations in delay for Controlled Load influences x_f, which again results in both graphs becoming uneven.

To illustrate the relation between fairness for real-time traffic and the delay measured, we use Figure 45. In this graph, we have linearized the delay variations of the Controlled Load class and reduced its magnitude in order to see that the line can bee seen as a tangent to the Best Effort curve. As we can see from the illustration, the two graphs touch at an input load about 60-70 %, and at this point we expect the fairness of the Controlled Load class to exceed the fairness for Best Effort.

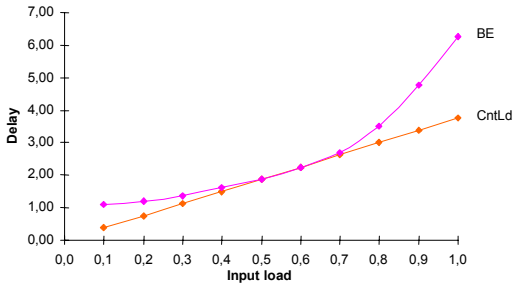

Figure 45. Difference in growth rate between delay of Controlled Load and Best Effort.

To conclude the results of fairness for real-time flows using Integrated Services, we can say that for loads less than 60 – 70 %, the Best Effort class will have the highest fairness. For loads higher than this, Controlled Load will be more fair.

For file transfers, Controlled Load is less fair compared to the Best Effort class. This can be explained by the reservations used in Controlled Load. By reserving bandwidth for each flow, the flows lose the opportunity to use bandwidth not used by other flows and the utilization of the class will decrease. As a result of TCP backing off to avoid congestion, the load of the

reservation will be kept below the reserved bandwidth. The result of TCP behavior in the Controlled Load and Best Effort class is illustrated in Figure 46.

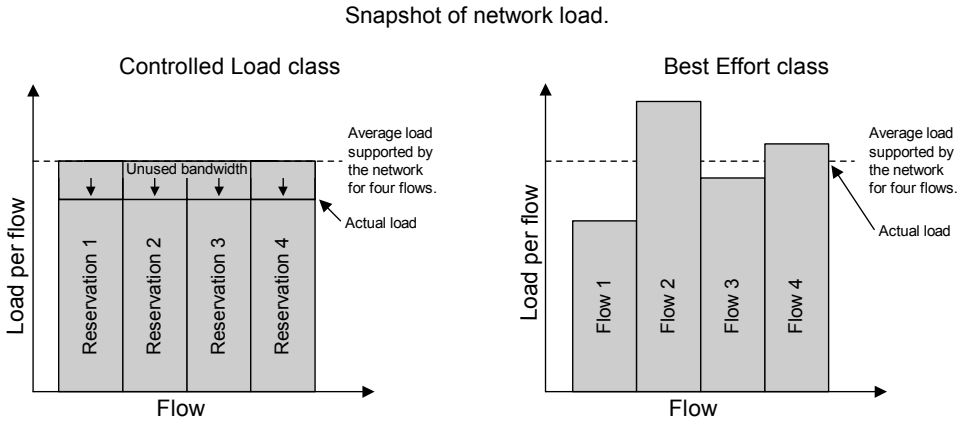

Figure 46. Illustration of TCP behavior in IntServ.

As we can see from the figure, TCP is not suited for use in the Controlled Load class because of the poor utilization of the reservations. This is not surprising as Controlled Load are mainly meant used for applications that are loss-, delay and jitter-dependent.

5.5.4.3 Differentiated Services

The graphs in Figure 47 show the fairness for Differentiated Services and are interpreted with the same approach as for IntServ.

Surprisingly, the Best Effort class provides better fairness for real-time traffic up to 90 % input load. One reason is that the Best Effort class allocates 25 % of the total bandwidth, while the other traffic classes only has 15 %. As the delay-measurements are dependent of the bandwidth, EF and AF will have more delay as long as Best Effort is not heavily congested.

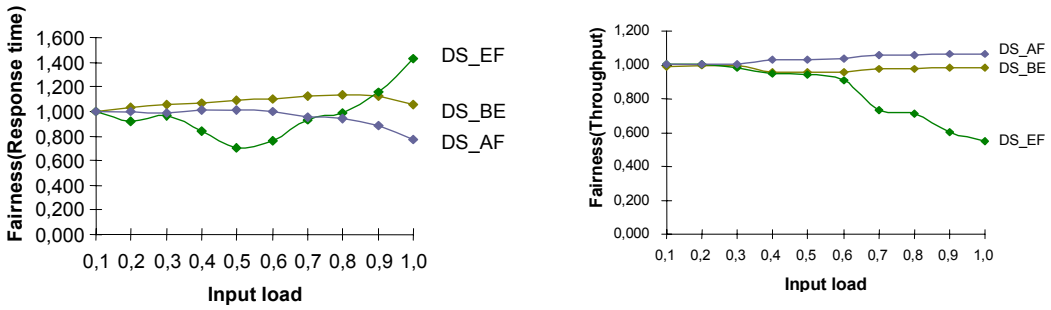

Figure 47. Computed fairness for DiffServ.

The behavior of the real-time fairness can be explained by examining on the delay-measurements performed. As mentioned earlier, the graphs are calculated by comparing the actual delay with the initial. From the delay-graph in the power test (Figure 43) we can see that the initial delays of the three classes are almost the same. The delay of EF intersects AF at 70 – 80 % while it crosses Best Effort at about 90 %. This is exactly the same points for intersection as shown in the real-time fairness graph.

For file transfers, Expedited Forwarding perceives a very poor fairness when the input load grows above 60 %. The measured throughput for EF traffic is located around 300 Kbps for 60-100 % load, even if the throughput is expected to be 600 Kbps for 100 % load. As explained earlier, this problem seems to have its origin in the fact that the policies of the EF

class are hard to configure. It must also be mentioned that EF is not intended used by file-transfer flows, which almost always use TCP.

5.5.4.4 The service models compared

Figure 48 and Figure 49 shows the fairness when we compare all the service models collectively. The graphs indicate how each service model perceives its fairness compared to the other service models. To create these graphs we use the "fair allocation mark" x_f, computed by using formula 7. x_f is the same for all the service models. The fairness index then is calculated for all the input loads dividing x_i with x_f.

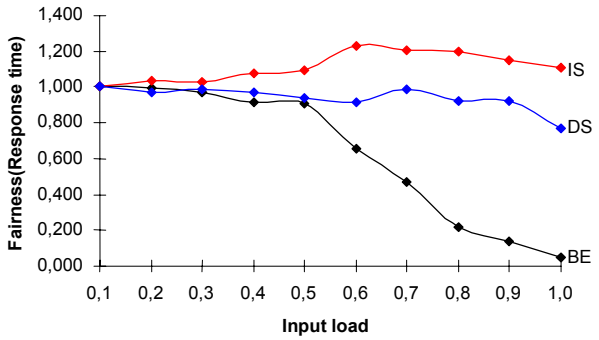

Figure 48. Computed fairness for real-time traffic.

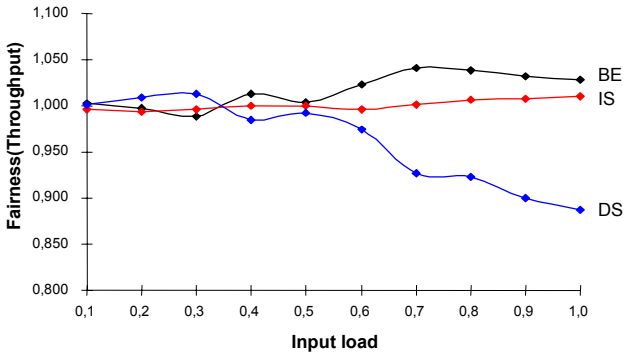

Figure 49. Computed fairness for file transfers.

From the graph on the left hand, we can see that IntServ has the best fairness concerning real-time traffic. Thus, IntServ has less increase in delay than the other service models as the input load grows.

The Best Effort network has an exponential increase in delay for real-time traffic. Therefore, the fairness for this service model is decreasing fast from 50 % load, which is the optimal load for this service model. A real-time flow transferred in an unloaded Best Effort network will perceive much better service than a flow transferred in a congested Best Effort network compared with the other service models.

The graph to the right, concerning fairness for file-transfers, shows what throughput a flow will experience compared to what is expected. The graph also indicates how a flow will observe its fairness compared to the other service models.

For file-transfers, Best Effort has better fairness than IntServ and DiffServ. All flows using Best Effort are prioritized equally and given a high throughput in proportion to the available bandwidth.

DiffServ has the least fairness of the service models. The reason for this is that DiffServ has several small classes (EF and AF is 15 %) that result in poorer utilization of the bandwidth compared to Best Effort and IntServ. The delay-measurements are also demanding a certain share of these 15 % and helps in degrading the results for the file-transfer flows. However, it is important to notice that the scale of the vertical axis to the right is much less than in the graph to the left, so the fairness is not as different as it might seem.

If we take both graphs into concern, IntServ has the best fairness of the service models. Compared with DiffServ, IntServ has only two service classes and is therefore able to utilize its bandwidth much better than DiffServ.

5.5.5 Summary

Fairness is an important consideration in most performance studies. Particularly in distributed systems, where a set of resources is shared by a number of users, fair allocation is important. Fairness implies equal allocation of resources, although there is little agreement among researchers as to what should be equalized.

If we take a small amount of resources from a user and give it to another user, the new allocation should be more fair if the receiver have less resources than the giver and vice versa. The difference in fairness between Best Effort and IntServ or DiffServ, is that there are certain rules for who is going to get the resources. We take resources from some users and give it to others. In total, we get a different resource allocation.

If we take resources from the most satisfied users, and give to the least happy users, we would expect individual perception of fairness to rise and the overall fairness to go up. On the other hand, if only real-time traffic is given additional resources, other applications and their users may become dissatisfied. Therefore, we have studied what happens with the “discriminated” file-transfers at the same time.

As a particular user’s allocation is increased from zero, the fairness first increases. It then reaches a maximum. Any further allocation to that user results in other users perceiving unfairness and the overall fairness decreases. If there are no limits on allocation of resources, the worst case of fairness can be near zero. By putting upper and lower bounds on allocation, no user can get less than a certain amount or more than a certain amount, and we can guarantee a minimum level of fairness.

6 QoS for different network topologies

6.1 Overview

In this chapter we make a proposal, based on theory and the testbed results, to what kind of QoS mechanisms that are appropriate for three different network topologies. First, section 6.2 describes a small-scale network, i.e. a company network, and makes a proposal to appropriate QoS mechanisms. Second, section 6.3 describes a mobile network with UMTS as an example and how QoS are achieved end-to-end between the Internet and the UMTS network. Section 6.4 is a description of a large backbone and a proposal to what QoS mechanisms that can handle the large amount of traffic in such a network.

6.2 A small-scale network

6.2.1 Network characteristics

In small-scale networks, the number of users is limited. As we can see from Figure 50, which is a small business network, the users will often be separated into segments with Ethernet switches. In this way, communication between two computers in the R&D division will not be noticed on other segments. Therefore, the load on the network is quite low with normal use. However, this does not solve problems associated with access to the Internet, like congestion in the gateway.

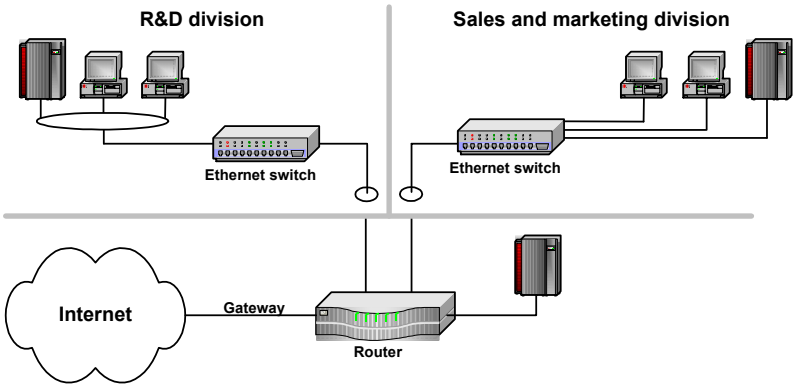

Figure 50. Small business network.

Examples of small networks may be companies, schools or similar institutions. When used in business, the networks are usually used to send mail, exchange documents and similar operations. Occasional downloads from servers using the network will occur, though most installation of new software and other bandwidth requiring operations will in most cases be avoided by installing from CD. If the users of the network are using it as they are supposed to, the load should be kept moderate. However, bursts of network traffic can arise as a consequence of coincidence or misuse.

As mentioned, a network may be segmented into smaller domains. This will help to keep down the overall load in the network, but there are still possibilities for bursts of data, within or between the segments, that can lead to congestion in the routers and switches.

It is not unexpected that some users will try to get better rights than others in a network. Malicious implementations of TCP may reduce an application's ability to back off during network congestion. Malicious users will force other honest users to back off and receive less throughput.

If each division in the company has its own server with most of the information needed for daily work, the need to communicate with other segments or the Internet will be minor. This can be improved even further by local caching of frequently used resources on the Internet.

A small-scale network will have a low number of routers in the network, thus transitions to new QoS mechanisms will be easier and the maintenance required is kept at a moderate level. Often there is a desire to differentiate different groups of users with concern to e.g. bandwidth. The average user will have other demands regarding throughput and delay than e.g. a user participating in a videoconference.

In most cases there are only one gateway to the Internet, and all the users connected to the network must share the resources of this connection. This may result in a bottleneck at the gateway when many users are accessing the Internet at the same time. However, this congestion does not influence the performance of each individual network segment.

6.2.2 A proposal to suited QoS mechanisms

Managing a network is very difficult, since there are many applications with different usage patterns of the available bandwidth. Many applications are virtually impossible to predict, yet network managers need to be able to support mission-critical applications even during peak periods.

At first glance it may seem that a Best Effort network would be an excellent choice in a small-scale network. If the total bandwidth is high enough to keep the link utilization below 50 %, all kinds of traffic will experience a fair and efficient service. However, to avoid the need for a constant increase in bandwidth, and at the same time be able to deliver service differentiation, the implementation of more sophisticated technologies are required.

We consider a simple improvement first, using the Best Effort model with Fair Queuing. This network will be able to schedule small real-time packets in front of large file-transfer packets, and in this way ensure that real-time flows will experience less delay than file-transfers. Another advantage with this solution is that, because of more fair queuing and scheduling, the optimal load is higher than in a pure Best Effort network.

For small networks used in locations where delay and jitter not are critical, simple solutions are suggested. A solution with Best Effort and e.g. CBQ would increase the utilization factor of the network and at the same time keep management costs low. Using this solution, flows with specific IP addresses or port numbers can be assigned different percentages of the total bandwidth and in this way stopping certain user groups from occupying the whole bandwidth.

Nevertheless, a Best Effort network would not be able to give any guarantees. Imagine that a salesman in an IT corporation runs a videoconference as part of a presentation for possible customers. Clearly, a flashy picture and poor sound will not give a good impression. To be able to individually reserve resources in the network, IntServ or DiffServ would be a better solution.

Since a business network is relatively small, scaling problems are of no concern when we focus on this network alone. IntServ would be able to deliver excellent QoS within the business network. The ability to guarantee a specific reservation and functionality in the network will be a welcome solution if the company is dependent on having guarantees for real-time traffic. By using IntServ in the network, we avoid that videoconferences are affected by random bursts in the network. The guarantees for QoS with IntServ are so good that the IntServ network may be used to handle internal phone calls in the company.

With QoS guarantees for real-time applications, the input load of the network may be much higher than 50 % without any significant decrease in service quality. IntServ also have the advantage of giving explicit feedback to the applications. In this way the host can be certain of the reservation being executed or not.

To have a small network operating optimally, it is important to configure the routers optimally. We figure that the need for QoS may be of limited extent, and it is therefore smart to keep the amount of bandwidth that is reserved for QoS moderate. IntServ can be configured with a given bandwidth for QoS and then let the Best Effort class borrow bandwidth when it is not used by the QoS classes. If a larger portion is reserved for QoS, without the possibility of lending free bandwidth to Best Effort, it will degrade the performance of the Best Effort class and non-real-time traffic will suffer under an unfair resource allocation.

6.3 A mobile network

6.3.1 Introduction

There have been several years of uncertainty to what type of QoS technologies that are appropriate for networks in the future. While providers of fixed networks have problems deciding what new technologies to use, mobile networks already have identified some Quality of Service aspects. The reason is the evolution of the third generation (3G) mobile standard, UMTS, which in the last couple of years has pushed the achievements in this area.

The proposal of QoS mechanisms in a mobile network is based on two specifications from 3GPP [52] [53]. They describe how a mobile network will support QoS for IP traffic and how it interacts with the Internet. The experiences from the alternatives chosen for mobile networks will probably influence the choices of QoS technologies for IP networks in the future. Thus, it will give us interesting information before summing up what we think will be the best QoS solution for real-time IP traffic.

6.3.2 The mobile network architecture

For the time being, mobile networks are getting ready to offer packet switched data services, in addition to already existing circuit switched data solutions. The mobile network extension, called General Packet Radio Service (GPRS) make use of IP in the core network. From Figure 51, we can see the packet-switched mobile core network in yellow.

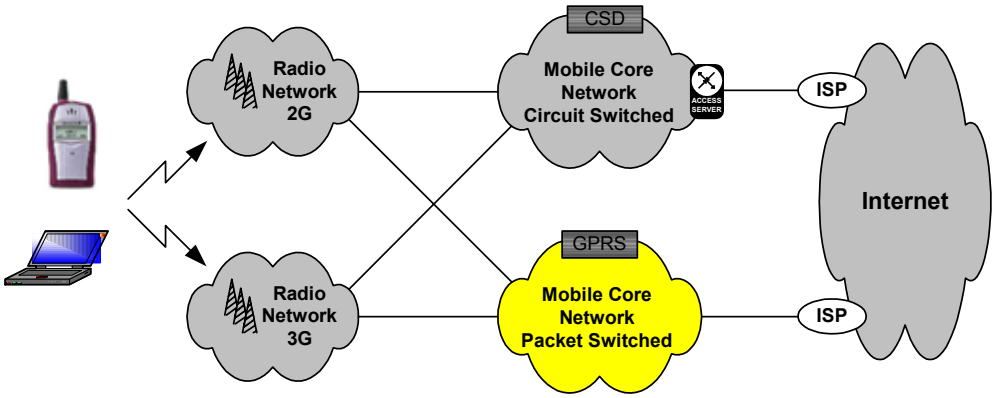

Figure 51. A third generation mobile network.

The 2G radio network and the circuit-switched mobile core network, both on top of the figure, is what we associate with a second-generation mobile network, known as GSM. GPRS is an extension, carried out as a step between 2G and 3G mobile networks. In 2002, a new radio interface that supports higher bandwidth for data transmissions will be added, transforming the mobile network into a third generation mobile network, a.k.a. a Universal Mobile Telecommunications System (UMTS).

We will focus on the network from the mobile host via the 3G radio network and the GPRS network to the Internet. The network is similar to many other data networks because it uses IP as the network protocol, and since the network is entirely new, new network technologies can be introduced to provide QoS for traffic.

A part of the mobile network is a radio interface, and the restrictions and limitations of it have to be taken into account. The delay, jitter and amount of packet loss in such a network is very

high, compared to fixed networks. This challenges the rest of the network to be faultless, so the retransmission of traffic over the already vulnerable radio interface is kept to a minimum.

The data traffic over a mobile network is different than in a fixed network. This has to do with lower bandwidth, something that makes the users reduce the use of throughput-demanding services like file-transfers. Interactive services such as IP telephony and videoconferences will not likely be used in great extent for the next few years, since a mobile network has other ways of handling telephony, and videoconferences will demand too high throughput in a combination of low delay. Interactive services with less critical demands for delay such as chat, games and different types of database lookups, will probably have an enormous growth with UMTS.

A mobile network will typically have a lot of users at the same time during particularly busy hours of the day. Thus, scalability is an important issue for a mobile network.

6.3.3 A proposal to suited QoS mechanisms

6.3.3.1 Introduction

It is not reasonable to define complex mechanisms in a mobile network, due to different error characteristics of the radio interface. The QoS mechanisms provided, have to be robust and capable of providing reasonable QoS resolution, and only the QoS perceived by the end-user matter. This proposal is based on a graduate thesis performed by a former student at Agder University College, Jan Flemming Henriksen [54].

6.3.3.2 UMTS QoS classes

There are four different UMTS QoS classes: Conversational class, Streaming class, Interactive class and Background class. The main distinguishing factor between these classes is how delay sensitive the traffic is: Conversational class is meant for traffic which is very delay sensitive, while the Background class is the most delay insensitive traffic class.

The Conversational and Streaming classes will mainly be used to carry real-time traffic flows. The main difference between them is how delay sensitive the traffic is. Conversational real-time services, like telephony speech (e.g. GSM), voice over IP and videoconferences, are the most delay sensitive applications and required characteristics are strictly given by human perception. Streaming services is one-way transport of real-time data flows always aiming at giving a live audio or video experience at the receiver. The service uses an adaptive playback buffer to limit variations in delay.

The Interactive and Background classes will be used by traditional Internet applications like www, email, telnet, FTP and news. Due to looser delay requirements, compared to the Conversational and Streaming classes, both provide better error rates by means of channel coding and retransmission. The main difference between Interactive and Background class is that Interactive class is mainly used by interactive applications, like chat, games or interactive web browsing, while Background class is meant for background traffic, e.g. background download of email, files or SMS. Separating interactive and background applications ensure better services for the interactive applications. Traffic in the Interactive class is scheduled with higher priority than Background class traffic, so background applications use transmission resources only when interactive applications do not need them. This is very important in a wireless environment where the bandwidth is low compared to fixed networks.

IP-based service models are supported by Packet Data Protocol (PDP) contexts in UMTS, and both IntServ and DiffServ are supported. Both service models are controlled by applications

residing in the Terminal Equipment (TE), allowing different application-specific QoS levels for the same PDP context. Internet applications' QoS reservations shall be mapped to UMTS QoS attributes at the border between the mobile and fixed network. RSVP support within the UMTS packet core network requires flow establishment and possibly aggregation of flows, while DiffServ requires that there is either one QoS profile for each traffic type or alternatively the priority and traffic type information is included in the data packets. PDP context guarantees that IP packets are given the right priority and treatment in the cases where UMTS nodes support neither IntServ nor DiffServ.

6.3.3.3 Mapping from UMTS QoS classes to DiffServ PHBs

UMTS must handle a lot of users at the same time, which means the network must be scalable. Therefore, 3GPP recommends DiffServ as the end-to-end QoS model. The UMTS QoS classes are well suited for mapping to DiffServ's PHBs (EF, AF and Best Effort). The traffic from the UMTS Conversational class is mapped to DiffServ's EF class, due to the strong real-time demands. However, the EF class should not be over provisioned, so it is important with a good traffic control mechanism at the ingress node.

UMTS Streaming tolerates some delay, as well as it can handle a low percentage of packet loss. DiffServ's AF PHB is suited for this kind of traffic, where delay still is low, but increases with the input load. As we remember, AF uses different drop precedences for traffic out of profile, based on the RIO mechanism. Therefore, an AF class introduces some packet drop for traffic out of profile to keep network congestion and delay low. The order of the packets is not tampered with, avoiding the introduction of unnecessary jitter to the traffic.

The AF class may also well handle the UMTS Interactive class. However, an AF class appropriate for interactive applications with more bursty bitrates should have a token bucket meter with lower rate and more depth than streaming to handle bursts and avoid applications constantly sending high bitrates. UMTS Background traffic is not prioritized and should use the remaining available capacity from DiffServ's Best Effort class. This class should be able to borrow free capacity from light-loaded AF classes.

6.3.3.4 Mapping from UMTS QoS classes to IntServ services

Even though 3GPP has DiffServ as their first choice of service model, they want to offer IntServ and RSVP as an optional choice. IntServ's Guaranteed Service is capable of handling intolerant real-time applications like telephony, and should be the first choice for UMTS Conversational class, and perhaps UMTS Streaming if this traffic can be prevented from deteriorating the Conversational.

IntServ's Controlled Load is most appropriate for adaptive traffic, since some changes in delay and throughput will exist in this class. UMTS Streaming is adaptive, but not to the same extent as UMTS Interactive that will get a very good service from Controlled Load. UMTS Background is as with DiffServ, placed in the Best Effort class.

6.3.3.5 End-to-end signaling

A mobile network is complicated with intensive signaling, therefore it is important that service differentiation is efficient, and not unnecessary complex. The QoS mechanisms on the Internet must interact with internal control mechanisms in UMTS, thus UMTS' internal QoS mechanisms must be converted to IETF standardized mechanisms at the egress UMTS node. 3GPP has described the superior control architecture for the QoS mechanisms at the IP layer. Figure 52 is a simplified QoS model showing a mobile host connected to a remote host on the Internet via a UMTS network.

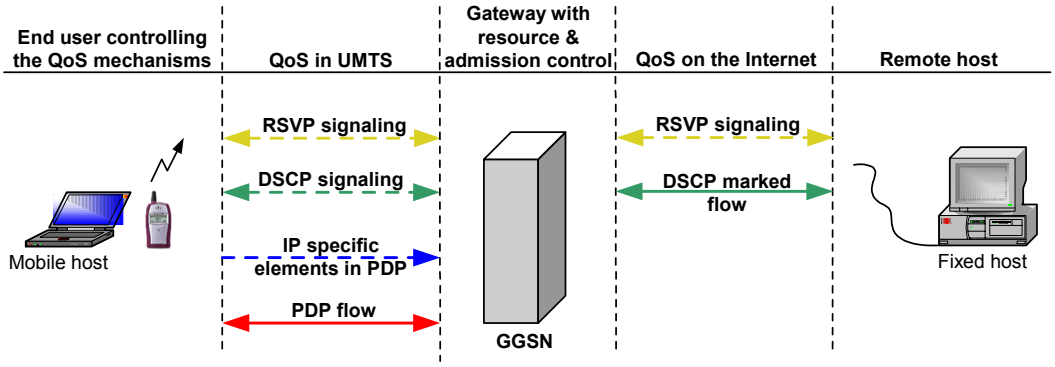

Figure 52. QoS model of a mobile network.

The mobile host is signaling control attributes in a PDP context. The attributes are negotiated with GGSN and depend on the service profile of the mobile subscriber. GGSN is the gateway to the Internet and must convert the QoS attributes in the PDP context to be understood by IntServ or DiffServ. However, this is not necessary if the application is IntServ or DiffServ capable, then such signaling is carried transparent (dotted lines) to GGSN. The mobile host may also provide IP specific information to GGSN without supporting any service model. Using IP specific elements of the PDP context, the host may enhance the interworking options to the DiffServ edge function of the GGSN.

The resource control in GGSN makes decisions based on available capacity in the UMTS network and available capacity estimated from other resource controls on the Internet. The resource controlling nodes may have functionality similar to a Bandwidth Broker.

It is not expected that low-priced and simple UMTS terminals will support IntServ. However, laptops operating with recent versions of Microsoft Windows are able to support IntServ. IntServ offers the possibility to continuously signal variations in radio coverage and possible handovers to areas with changed capacity. Sudden link variations are valuable to know about both for the corresponding host and for the administrating nodes in the network. Thus, it is a strength that DiffServ do not have.

On the other hand, DiffServ is scalable, but does not give any end-to-end guarantees. To be able to give that guarantee with DiffServ, we need to use the IntServ over DiffServ architecture or a Bandwidth Broker that can signal available resources throughout the network. However, both solutions lead to increased signaling and complexity, which again might decrease the network's scaling capabilities.

Real-time applications demanding low delay and having high throughput, like videoconferences, should use IntServ. The end-users will be able to signal each other to optimize bitrates under fluctuating conditions of transmission. In addition, GGSN will be continuously updated, which makes it able to adapt the resource allocations through the network. Videoconferencing is not very common to use compared to telephony, thus scalability is not a problem. However, the users of telephony services are many, and the service model supporting telephony must scale well. All telephone calls have about the same demands for delay and throughput, thus everyone can be treated equally. DiffServ's EF class will according to these facts take care of IP telephony very well.

Streaming and interactive applications with low demands for delay will get QoS from specially adjusted DiffServ AF classes with policy control to avoid flooding of data and overload conditions. There is no need to reserve resources for such traffic, because either the flow is totally predictable like streaming, or else it is very unpredictable like bursty interactive traffic. Signaling with RSVP and IntServ is therefore unnecessary.

A solution for guaranteeing QoS for IP telephony is to establish dedicated circuits on the Internet. The circuits must have very low delay, but do not necessarily need high bandwidth. Using MPLS and traffic engineering, it is possible to have dedicated circuits. Traffic from UMTS' Conversational class may be reasonable to route over an MPLS path. In practice, such traffic must have a dedicated DSCP to indicate which MPLS path the traffic will follow.

6.4 A large backbone

6.4.1 Introduction

A backbone is a high-bandwidth "spine", often based on fiber technology, delivering packets to a large number of service providers. The location of the backbones have been chosen to distribute data traffic between areas with high demands, and the local service providers connected to the backbone have to deal with the final distribution out to the customers. Figure 53 shows the Telia backbone in Northern Europe.

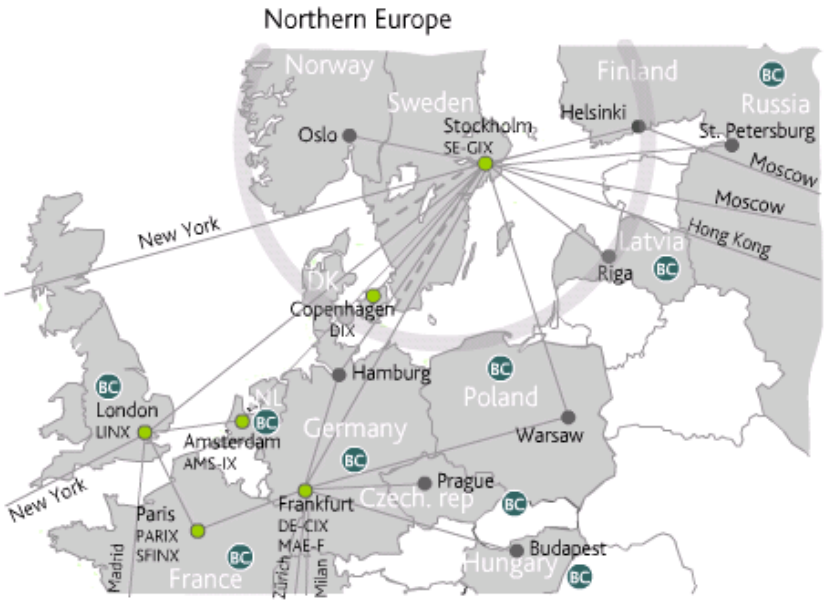

Figure 53. Backbone topology for Northern Europe [55]

6.4.2 Network characteristics

We can see from Figure 53 that the node in Stockholm probably handles a high amount of traffic. For real-time traffic, efficient handling in these nodes is essential in order to reduce delay and jitter. To satisfy these demands, the routers require efficient routing and forwarding algorithms.

To use Telia as an example, they operate with a strict policy of "no overbooking". When traffic on a link reaches 50% of maximum capacity, they start to increase the capacity. By the time traffic levels reach 70%, they bring the new capacity into service. As a result, the bandwidth in the backbone is dimensioned according to the traffic experienced in the network.

Redundancy is another important issue for backbones. There will always be a danger of only having one link to a backbone destination, which means that if the link fails, the connection will be lost until the link is repaired. Having more than one link to each node opens the possibility to route around links that are down. Another advantage with redundancy is the possibility to reduce traffic on heavily loaded links. The two links from Europe to New York are an example of this. To make redundancy work optimal, it is important to have frequent routing updates.

6.4.3 A proposal to QoS mechanisms in a large backbone

Global IP backbones typically handle a great number of flows at the same time, thus IntServ will not be considered due to scaling problems. There are a large variety of users, and the backbone should be able to deliver the best possible QoS to satisfy even the most demanding user.

Services adapted for real-time flows such as EF and Controlled Load are not made for handling TCP flows, and these must be denied access with policing. This is important to avoid unnecessary strain on a large IP backbone.

There have been suggested different solutions how to be able delivering QoS efficiently without having the scaling problems of IntServ. As mentioned in chapter 4, both IntServ over DiffServ, MPLS and the Bandwidth Broker shows positive trends to accomplish this.

IntServ over DiffServ, with focus on backbones, means that there will be used IntServ internally in companies or other small domains, while DiffServ should be implemented in the backbone.

A solution using a Bandwidth Broker would increase the overall control of the resources available in the backbone. This would lead to a more controlled utilization of the backbone, as well as it can give explicit feedback if there are available resources to handle the desired QoS throughout the backbone. However, as mentioned in the section describing the Bandwidth Broker, a BB can experience scaling problems in a network as large as an Internet backbone.

MPLS delivers numerous qualities ideal to be used in a global IP backbone. As we can see from Figure 53, the topology of the backbone is of such type that there will be a lot of packets traveling the same route. E.g. if all kinds of real-time traffic is classified in one FEC, then we could use one label switched path (LSP) for all such flows going from Sweden to France. From the illustration of the TeliaNet backbone we can see that this would at least require the traffic to be routed through the node in Frankfurt. Then, the router in Frankfurt can route all real-time traffic from Sweden to France only by looking at one single label. The fact that MPLS uses indexed routing tables makes the lookup of the label much faster than ordinary IP routing.

Traffic engineering in MPLS opens the possibility to route around links that are heavily loaded. As most backbones uses redundancy, this could be used to decrease the load on overloaded links and in this way distribute the traffic over more links in order to avoid congestion. E.g. looking further at the LSP from Sweden to France, we could classify file-transfers from Sweden to France as another FEC, and set up an LSP for this traffic via London. Having file-transfers and real-time traffic flowing on two different paths, we know that they will not influence each other.

The use of optical networks in the backbone would also make it possible to utilize the qualities of MPLambdaS. This technology avoids transitions from optical to electrical signaling in intermediate nodes, and will in this way be able to decrease delay in the backbone.

One of the most essential demands for a backbone is that it has to handle as much traffic as possible. This sets demands for the delay experienced in each router, as well as the opportunity to route traffic around routers where the load is already high. Fulfilling the demands will result in reduced delay and increased QoS.

As discussed above, MPLS delivers many useful functions to increase the efficiency of routing in a global IP backbone. The possibility to route traffic around congested nodes in the network and have minimal delay, shows that MPLS is very suited for this task.

7 Discussion

7.1 Overview

Throughout this report, we have discussed service models and different QoS technologies individually. In this chapter, we combine the theory from chapter 3 and 4 with the results we have achieved by testing the service models in order to look at a total solution for an IP network. We use the Internet as an example, and an illustration of the architecture can be seen in Figure 54.

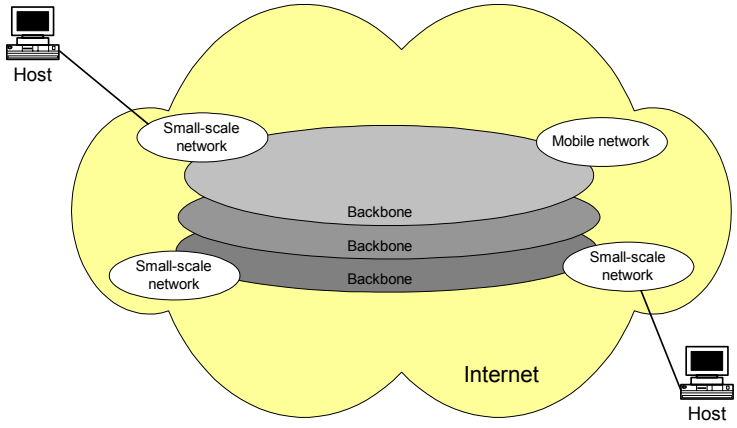

Figure 54. Architecture of the Internet.

Section 7.2 is a preliminary discussion where we look at limitations and trade-offs in the thesis. Further, this chapter discusses the three service models individually from section 7.3 to section 7.5. We point at strengths and weaknesses and suggest improvements to the service models. Improvements may be different QoS forwarding mechanisms or other QoS technologies. Section 7.6 is a discussion of future work.

7.2 Preliminary discussion

This thesis could have been done with different approaches. We would like to focus on limitations we have done and the approach to the testing phase.

First, it should be mentioned that we have neither been able to test MPLS, Bandwidth Broker nor IntServ over DiffServ. This means that the discussion and conclusion concerning these technologies is based on theory only. However, we find these technologies very interesting and promising. Therefore, they are a natural part of a thesis concerning QoS in IP networks.

In the tests, the delay measurements are performed using an extra flow, since Chariot is not able to measure delay directly from the real-time flows. This means that this flow will introduce additional strain on the routers and therefore probably give a poorer result than if the delay was measured directly from the real-time flows. Delay should be measured by using a dedicated delay-measurement tool that records the delay from a packet is sent from host A until it arrives at host B. This would give us a more accurate delay measurement for the real-time flows.

It must also be mentioned that Best Effort is tested with different conditions than IntServ and DiffServ. The reason is that we have chosen to test the service models with another bandwidth than what is actually available on the link. Since we are not using ALTQ for Best Effort, we had to use Dummynet to limit the bandwidth. Instead, the behavior of ALTQ and Dummynet might be different, therefore the measurements may be influenced.

We have measured very high initial delay for IntServ's Controlled Load class. This could be related to Microsoft's implementation of the protocol stack and generic QoS. This is however confirmed from neither Microsoft nor NetIQ, and should be investigated further.

After the testing phase of the thesis, we have discovered that some of the tests could have been performed differently. The strange behavior of DiffServ's EF class, especially in the power test, is probably related to the use of TCP flows for the delay measurements. As EF is using a token bucker meter, it requires a very large bucket depth in order to handle the bursty nature of TCP, and we were not able to set the bucket depth deep enough.

We have not discussed admission control in this thesis, as it is not directly in relation to the behavior of the service models. In the absence of admission control, everybody will probably ask for the best QoS and in this way degrade a real-time service to a Best Effort service. Thus, admission control is a vital part to make QoS work as desired.

7.3 Best Effort

From the tests in chapter 5, we have found that Best Effort can have an input load of 50 % before it starts to decrease rapidly in both efficiency and fairness for real-time traffic. This means that if Best Effort still is to be used on the Internet, the links have to be over-dimensioned in order to deliver QoS that is satisfactory to real-time flows.

TCP improves the behavior of Best Effort with the ability to decrease the data rate when it senses congestion in the traffic path. If all hosts in a Best Effort network use TCP, the network will offer a good service for non-real-time flows. However, users with enough knowledge and the urge to speed up their data rates can manipulate their TCP implementation in order to avoid a back-off during congestion. Other hosts that use the original TCP implementation will experience a degraded service with less throughput.

UDP flows do not have the ability to adapt the data rate when congestion occurs in the same way TCP does. Real-time applications using UDP then causes unfairness since they will behave like a manipulated TCP flow, stealing other flows' resources.

Best Effort cannot give any QoS guarantees for real-time flows. In this way, VoIP will be treated unfair compared with file-transfers, as file-transfers only demand throughput with less concern to delay in contrast to real-time flows that desire minimal delay.

In order to increase the QoS delivered by an IP network using Best Effort, there are several improvements that can be done without changing the service model entirely. An increase in service quality can be obtained by introducing QoS forwarding and queue management to differentiate between QoS dependent flows and those who are not.

Weighted Fair Queuing (WFQ) will aspire to let low-volume flows, like VoIP, fast through the routers, with better delay and jitter characteristics, than file-transfer flows with larger data packets. Priority Queuing (PQ) can let packets with high priority through the queue before other traffic. This will provide a good service for VoIP, but demands some kind of marking of the packets. The ToS field can be used for this.

Queue management such as Random Early Detection (RED) is designed to drop packets before the routers are congested. As mentioned above, TCP would interpret lost packets as a sign of congestion and therefore reduce its data rate. In this way real-time flows will be able to get their packets faster through the routers since the queues are shorter.

7.4 Integrated Services

Integrated Services (IntServ) uses signaling in order to make reservations. The reservations are maintained active in the router during the entire conversation. As the Internet is of a vast size, the number of reservations made in key traffic nodes can become very high. Every router in a reservation path has to keep track of all flows going through it, which results in a large number of reservation lookups per time unit. This results in many time-consuming operations that make IntServ scale poorly in larger networks.

The strength of IntServ is the ability to give explicit feedback whether a reservation is successful or not. By using reservations throughout the IntServ-compliant network, the load on each node can be predicted by using the reservation requests at hand and use this to decide if more reservations can be made without degrading the performance of those who have already been reserved. Once the reservation is established reservations still can be changed if needed, since the signaling continues during the entire conversation.

During a reservation, IntServ provides certain guarantees for bandwidth as well as delay and jitter. This can be seen as unfair for applications that are not allowed to reserve resources, but are very efficient in order to deliver QoS in smaller networks.

As mentioned, IntServ does not scale well. There have been suggested solutions to make IntServ function better on the Internet, and one of these is IntServ over MPLS. MPLS offers the ability to aggregate flows with similar behavior heading in the same direction, and mark these flows with the same label. The routers in the MPLS network will be able to understand the RSVP message sent by IntServ and utilize it to map flows with similar behavior into one Forwarding Equivalence Class (FEC). MPLS uses indexed routing tables, and will be able to route flows much faster than IntServ alone. This solution is very useful if we consider a multicast where a lot of the destinations are concentrated within a small geographical area.

Another solution to poor scaling is the combination of IntServ and DiffServ. While IntServ's strengths is in smaller networks, DiffServ functions best in larger networks. With IntServ over DiffServ, IntServ is only used in the edge networks, where the number of users is moderate. In the core network, e.g. a backbone, the DiffServ model is responsible for the flows' QoS behavior.

7.5 Differentiated Services

Differentiated Services (DiffServ) was originally designed to avoid signaling in the networks, thus increasing scalability. The lack of signaling means that there cannot be made complete guarantees throughout the network, which is not so good for IP telephony and other real-time applications. No signaling offer little dynamics for a bursty input load, since a DiffServ domain only can give guarantees for itself, not other domains the flow might pass through. Consequently, the flow might experience different service quality throughout the DiffServ network. To avoid a degraded service quality, the core network can be over-dimensioned and strict access control at the edges must be performed.

The increased scalability of DiffServ is a result of the way it groups flows in behavior aggregates. The aggregation of these flows will cause faster handling in the core network. However, it is difficult to configure per-hop behaviors that will satisfy the needs of various applications.

DiffServ is dependent on marking all its packets, and uses the ToS field in the IP header for this. The field is called a DiffServ Code Point (DSCP) and the redefinition of this field may cause old routers to misinterpret the DSCP as a ToS and therefore handle traffic with the wrong treatment

Improvements of DiffServ can be done with the use of MPLS, which is a mapping of the DiffServ behavior-aggregates onto forwarding equivalence classes. The mapping enables the ability to use indexed forwarding provided by MPLS, which causes faster forwarding. Traffic engineering gives the opportunity to use specific paths in the core network, and avoid nodes where the load is high.

The use of Bandwidth Brokers together with DiffServ will give DiffServ the possibility of signaling available resources throughout the network. In this way, the network can be managed more efficiently and there can be given more reliable guarantees for IP telephony. However, the signaling will, if widely used, introduce a scaling problem. If all flows are to be signaled, the load on the Bandwidth Brokers will be extremely high. By reducing the signaling to only concern real-time flows, the scaling problem can be reduced and the advantages of signaling are still present.

7.6 Future work

The suggestions presented in this thesis are based on technologies that we expect to result in standards. The solution with Bandwidth Brokers is still early in its development and we believe that there will be further evolvement of this model before it could be implemented in a large network.

MPLS is also still under development. However, the initial goals of the MPLS working group have been largely completed. In particular, the group has produced a number of RFCs that define the Label Distribution Protocol (LDP), the basic MPLS architecture and encapsulations, and definitions for how MPLS runs over ATM and Frame Relay links. From the MPLS working group at IETF we can see that, for the time being, goals and milestones are set until December 2001.

The IntServ working group at IETF has been removed from the website, and we can assume minimal further evolvement of this model. The DiffServ working group [57] however, has standardized a small number of specific per-hop behaviors (PHBs), and recommended a particular bit pattern or 'code-point' of the DS field for each PHB. No more PHBs will be standardized until all the current milestones of the working group have been satisfied and the existing standard PHBs have been promoted at least to draft standard status.

IPv4 contains some weaknesses that are addressed by IPv6. The IPv6 packet format, among other things, contains a new 20-bit traffic-flow identification field that can be used to distinguish traffic flows for optimized routing. The flow label can be used to identify flows even when the payload is encrypted [58]. QoS for IPv6 is still in the planning stage, but it is a foundation to make QoS functions more available in an open and interoperable manner.

The new release of the UMTS standard (release -5) is going to be finished this year, and should be followed closely. It defines a way to transfer all traffic in a mobile network based on IP, and will have a solution on how to handle real-time traffic based on the QoS mechanisms discussed in this report.

Routers from different manufacturers tend to interpret implementations of QoS mechanisms differently, and an important step to achieve openness between the implementations is the agreement of a common QoS architecture on the Internet. This cannot happen without an agreement between the large actors within IETF, such as Cisco, Ericsson, Microsoft and several others.

8 Conclusion

This thesis presents a performance evaluation of three different service models and their ability of providing QoS for real-time traffic in an IP network. The evaluation is performed through theoretical studies and practical tests. Even though there have been some remarkable behaviors in some of the service classes, we find that there is a consistency between theory and testing. The results obtained show that IntServ and DiffServ are capable of offering QoS guarantees for real-time traffic within certain limits.

Best Effort is suited for "friendly" applications that can adjust the data rate by sensing congestion in the network. The Best Effort network should be over-dimensioned, with at least twice the input load, in order to avoid large queues in the routers and high delay for real-time applications. Then, requirements to QoS will be fulfilled for real-time applications during steady traffic, even though there are no guarantees.

Differentiated Services and MPLS should primarily be used in the core network because of fast handling that results in more traffic being handled per time unit. By using MPLS, flows that have the same demands for QoS and are heading in the same direction can be mapped into the same aggregate flow.

Integrated Services are well suited for small networks, since it provides explicit feedback and strict resource reservation. The problem with IntServ is scaling when the network becomes very large. To be able to take advantage of IntServ's good properties in a large network, the core network should use DiffServ with some sort of signaling. A good solution that offers scalable signaling end-to-end is "IntServ over DiffServ" with Bandwidth Brokers between the DiffServ domains. Then, the core network will know the needs of the end-users and the bandwidth can be better utilized by letting the Bandwidth Brokers have the opportunity to configure DiffServ's behavior-aggregates dynamically.

Another solution is to have the core routers enabled for both IntServ and DiffServ at the same time. Then, real-time applications can reserve a Guaranteed Service end-to-end with RSVP just like traditional telephony does. This can only be done in a small scale with strict admission control, or else it will give poor scaling. Less delay-critical applications should be classified in a behavior-aggregate and be treated by DiffServ functions in the router. This gives weaker guarantees, but ensures network scalability.

An edge network may also deploy its own QoS model, as UMTS does. However, a network with its own QoS model must have the ability to map service requests into a standard IP network QoS mechanism. The mapping must be done at the ingress of the standard IP network and in such a way that the entire network can understand the QoS demands end-to-end.

The perspectives of this work are the evaluation of QoS mechanisms in a network that can realize real-time demands for interactive IP traffic. Future work should be to follow the evolution of DiffServ, MPLS and Bandwidth Broker. The standardization process for these is still at the draft phase, so changes and improvements most likely will occur. Also, standardizations made by 3GPP for handling real-time traffic in UMTS should be followed closely.

Abbreviations

3GPP	The 3rd Generation Partnership Project
ACK	Acknowledgement
Adspec	Advertising specification
AF	Assured Forwarding
ATM	Asynchronous transfer mode
BA	Behavior Aggregate
BB	Bandwidth Broker
BE	Best Effort
BGP	Border Gateway Protocol
CAR	Committed Access Rate
CBQ	Class Based Queuing
CBR	Constant Bitrate
DiffServ	Differentiated Services
DP	Drop Precedence
DS	Differentiated Services
DSCP	Differentiated Services Code Point
EF	Expedited Forwarding
Exp	Experimental
FEC	Forwarding Equivalency Class
FIFO	First In First Out queuing
Flowspec	Flow specification
FQ	Fair Queuing
FTP	File transfer protocol
GGSN	Gateway GPRS Serving Node
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
HFSC	Hierarchical Fair Share Curve
IEEE	The Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IntServ	Integrated Services
IP	Internet Protocol
IS	Integrated Services
ISDN	Integrated Services Digital Network
ITU	International Telecommunication Union
ITU-T	Telecommunication Standardization Sector of ITU
Kbps	Kilo(1024) bits per second
LDP	Label Distribution Protocol
LIB	Label Information Base
LSP	Label Switched Path
LSR	Label Switch Router
MAC	Media Access Control
MF	Multifield
MPLamdaS	Multiprotocol Lambda Switching
MPLS	Multiprotocol label switching
NBR	Nominal BitRate
OSI	Open System Interconnection
OSPF	Open Shortest Path First
OXC	Optical Cross Connections
PDP	Packet Data Protocol
PHB	Per-Hop Behavior
PQ	Priority Queuing
PVC	Permanent Virtual Circuit
QoS	Quality of Service
RED	Random Early Detection
RFC	Request For Comment
RIO	RED with In and Out

Rspec	Reservation Specification
RSVP	Resource Reservation Protocol
RTP	Real-time Transfer Protocol
SLA	Service Level Agreement
TCA	Traffic Conditioning Agreement
TCP	Transfer Control Protocol
TE	Terminal Equipment
ToS	Type of Service
trTCM	two rate three color marker
Tspec	Traffic specification
TTL	Time-to-live
UDP	Universal datagram protocol
UMTS	Universal Mobile Telecommunications System
VBR	Variable Bitrate
VC	Virtual Circuit
VLAN	Virtual Local Area Network
VPN	Virtual Private Networks
WFQ	Weighted Fair Queuing
WRED	Weighted RED
WTP	Weighted Time Priority

References

All the hyperlinks pointing to references are accessed during May 2001.

[1] IETF Network Working Group
Integrated Services in the Internet Architecture: an Overview, RFC1633, June 1994
http://www.ietf.org/rfc/rfc1633.txt?number=1633

[2] IETF Network Working Group
An Architecture for Differentiated Services, RFC2475, December 1998
http://www.ietf.org/rfc/rfc2475.txt?number=2475

[3] IETF MPLS Working Group
http://www.ietf.org/html.charters/mpls-charter.html

[4] Internet2 Qbone BB Advisory Council
A Discussion of Bandwidth Broker Requirements for Internet2 Qbone Deployment, August 1999
http://www.merit.edu/working.groups/i2-qbone-bb/doc/BB_Req7.pdf

[5] IETF Network Working Group
A framework for Integrated Services operation over DiffServ networks, RFC2998, November 2000
http://www.ietf.org/rfc/rfc2998.txt?number=2998

[6] Internet Engineering Task Force
http://www.ietf.org

[7] MPLS Resource Center
http://www.mplsrc.com/about.shtml

[8] The Information Technology Professional's Resource Center
http://www.itprc.com

[9] Internet2
http://www.internet2.edu/

[10] Internet2 QoS Working Group
http://www.internet2.edu/qos/wg/

[11] Qbone
http://qbone.internet2.edu/

[12] H. Zimmerman
OSI reference model-The ISO model of architecture for open systems interconnection, IEEE transactions on communications COM-28, pages 425-432, April 1980.

[13] Grenville Armitage
Quality of Service in IP networks, Macmillan Technical Publishing, pages 183-195, April 2000

[14] Cisco
Frame Relay
http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/frame.htm

[15] Geoff Huston
Internet Performance Survival Guide, QoS Strategies for Multiservice Networks, John Wiley & Sons, Inc, pages 280-282, February 2000

[16] Telecommunication Standardization Sector of ITU
ITU-T Recommendation G.114 - One-way Transmission Time, February 1996

[17] Grenville Armitage
Quality of Service in IP networks, Macmillan Technical Publishing, page 66, April 2000

[18] J. Heinanen and R. Guerin
A Single Rate Three Color Marker, RFC2697, September 1999
http://www.ietf.org/rfc/rfc2697.txt?number=2697

[19] J. Heinanen and R. Guerin
A Two Rate Three Color Marker, RFC2698, September 1999
http://www.ietf.org/rfc/rfc2698.txt?number=2698

[20] Cisco White Paper
Committed Access Rate
http://www.cisco.com/warp/public/cc/pd/iosw/tech/carat_wp.htm

[21] Ion Stoica, Hui Zhang, T.S. Eugene Ng
A Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Service, proceedings of SIGCOMM'97
http://redriver.cmcl.cs.cmu.edu/~hzhang-ftp/SIGCOM97.pdf

[22] Constantinos Dovrolis, Dimitrios Stiliadis and Parameswaran Ramanathan
Proportional Differentiated Services: Delay Differentiation and Packet Scheduling.
http://www.cis.udel.edu/~dovrolis/Papers/sigcomm99.ps

[23] S. Floyd, V. Jacobson
Random Early Detection Gateways for Congestion Avoidance, August 1993.
http://www-nrg.ee.lbl.gov/floyd/red.html

[24] Larry L. Peterson & Bruce S. Davie
Computer Networks, A Systems approach, Morgan Kaufmann Publishers, pages 476-482, October 1999

[25] Cisco
Random Early Detection (RED)
http://www.cisco.com/warp/public/732/Tech/red/

[26] W. Feng, D. Kandlur, D. Saha, K. Shin
Blue: A New Class of Active Queue Management Algorithms, April 1999.
http://www.thefengs.com/wuchang/blue/CSE-TR-387-99.pdf

[27] Larry L. Peterson & Bruce S. Davie
Computer Networks, A Systems approach, Morgan Kaufmann Publishers, pages 506-509, October 1999

[28] IETF Network Working Group
Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification, RFC2205, September 1997.
http://www.ietf.org/rfc/rfc2205.txt?number=2205

[29] IETF Network Working Group
Specification of the Controlled-Load Network Element Service, RFC2211, September 1997
http://www.ietf.org/rfc/rfc2211.txt?number=2211

[30] IETF Network Working Group
Specification of Guaranteed Quality of Service, RFC2212, September 1997
http://www.ietf.org/rfc/rfc2212.txt?number=2212

[31] IETF Network Working Group
Resource ReSerVation Protocol (RSVP) Version 1 Applicability Statement, Some Guidelines on Deployment, RFC2208, September 1997
http://www.ietf.org/rfc/rfc2208.txt?number=2208

[32] IETF Network Working Group
RSVP Cryptographic Authentication, RFC2747, January 2000
http://www.ietf.org/rfc/rfc2747.txt?number=2747

[33] IETF Network Working Group:
Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, RFC2474, December 1998
http://www.ietf.org/rfc/rfc2474.txt?number=2474

[34] IETF Network Working Group
An Expedited Forwarding PHB, RFC2598, June 1999
http://www.ietf.org/rfc/rfc2598.txt?number=2598

[35] IETF Network Working Group
Assured Forwarding PHB Group, RFC2597, June 1999.
http://www.ietf.org/rfc/rfc2597.txt?number=2597

[36] IETF Network Working Group
Requirements for IP Version 4 Routers, RFC1812, June 1995
http://www.ietf.org/rfc/rfc1812.txt?number=1812

[37] Information Sciences Institute University of Southern California
Transmission Control Protocol, RFC 793, September 1981
http://www.ietf.org/rfc/rfc0793.txt?number=793

[38] IETF Network Working Group
Multiprotocol Label Switching Architecture, RFC 3031, January 2001
http://www.ietf.org/rfc/rfc3031.txt?number=3031

[39] Uyless D. Black
MPLS and Label Switching Networks, Prentice Hall, pages 1-18, January 2001

[40] Uyless D. Black
MPLS and Label Switching Networks, Prentice Hall, pages 60-85, January 2001

[41] IETF Network Working Group
LDP Specification, RFC 3036, January 2001
http://www.ietf.org/rfc/rfc3036.txt?number=3036

[42] IETF Network Working Group
Requirements for Traffic Engineering Over MPLS, RFC 2702, September 1999
http://www.ietf.org/rfc/rfc2702.txt?number=2702

[43] Geoff Huston
Internet Performance Survival Guide, QoS Strategies for Multiservice Networks, John Wiley & Sons, Inc, pages 432-433, February 2000

[44] Dr. Bruce Davie
Service Providers to Benefit from New Functionality, Packet Magazine, April 1999
http://www.cisco.com/warp/public/784/packet/apr99/6.html

[45] Uyless D. Black
MPLS and Label Switching Networks, Prentice Hall, pages 157-173, January 2001

[46] ITTC
Differentiated Services - Implementations
http://demasie.aditel.org/mirrors/qos.ittc.ukans.edu/%257Ekdrao/BB/

[47] Eurescom P1008F
Inter-operator interfaces for ensuring end to end QoS, State-of-the-art of IP Inter-domain management and supporting measurements, July 2000

[48] Luigi Rizzo
IP Dummynet
http://www.iet.unipi.it/~luigi/ip_dummynet/

[49] Christos Kolias and Leonard Kleinrock
The Power Function as a Performance and Comparison Measure for ATM Switches
http://www.lk.cs.ucla.edu/~ck/res/glob298.ps

[50] W. Feng, D. Kandlur, D. Saha and K. Shin
Understanding TCP Dynamics in an Integrated Services Internet, NOSSDAV'97, May 1997
http://www.eecs.umich.edu/~wuchang/work/nossdav97.ps.Z

[51] Raj Jain, Eastern Research Lab
A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer System, September 1984

[52] Technical Specification Group Services and System Aspects
3GPP TS 23.107 V5.0.0 (2001-04) "QoS Concept and Architecture (Release 5)", April 2001
ftp://ftp.3gpp.org/Specs/2001-03/Rel-5/23_series/23107-500.zip

[53] Technical Specification Group Services and System Aspects
3G TR 23.821 V1.0.1 (2000-07) "Architecture Principles for Release 2000", July 2000
ftp://ftp.3gpp.org/Specs/2000-06/R2000/23_series/23821-101.zip

[54] Jan Flemming Henriksen,
En vurdering av aktuelle teknikker for tjenestekvalitet i et IP-basert stamnett for UMTS, pages 54-61 December 2000
http://www.siving.hia.no/ikt00/ikt6400/jfhenriks/Hovedoppgave_JFH_v21.pdf

[55] TeliaNet,
http://www.telia.net

[56] NetIQ Corporation
http://www.netiq.com

[57] IETF DiffServ Working Group
http://www.ietf.org/html.charters/diffserv-charter.html

[58] Larry L. Peterson & Bruce S. Davie
Computer Networks, A Systems approach, Morgan Kaufmann Publishers, pages 328-340, October 1999

[59] ALTQ: Alternate Queueing for BSD UNIX
http://www.csl.sony.co.jp/person/kjc/programs.html#ALTQ

Appendixes

Appendix A - Validation of IntServ and DiffServ support in the testbed

Appendix B - Preliminary tests

Appendix C - Fair resource allocation

Appendix D - Installation of software routers

Appendix E - Software

Appendix A - Validation of IntServ and DiffServ support in the testbed

A.1 Integrated Services

The functionality of the IntServ testbed has to be validated in order to confirm that the network is delivering the services it is supposed to. Because IntServ is transparent to non-RSVP clouds, we will not receive any direct indication of error if the network does not support it. Therefore, this will complicate the verification.

"*Resv messages carry reservation requests hop-by-hop from receivers to senders, along the reverse paths of the data flows for the session. The IP destination address of a Resv message is the unicast address of a previous-hop node, obtained from the path state. The IP source address is an address of the node that sent the message.*" [RFC2205]

This means that if the routers support RSVP, they intercept Resv messages traversing through the network and try to apply the reservations described in that message. When the reservation is done, the router changes the IP source address in the Resv message to the address of the output interface on the router.

By using Microsoft Network Monitor, we were able to monitor the messages on the network. As the citation from RFC2205 says, the source address of the Resv message should be the IP address of the router closest to the receiver if the network support IntServ.

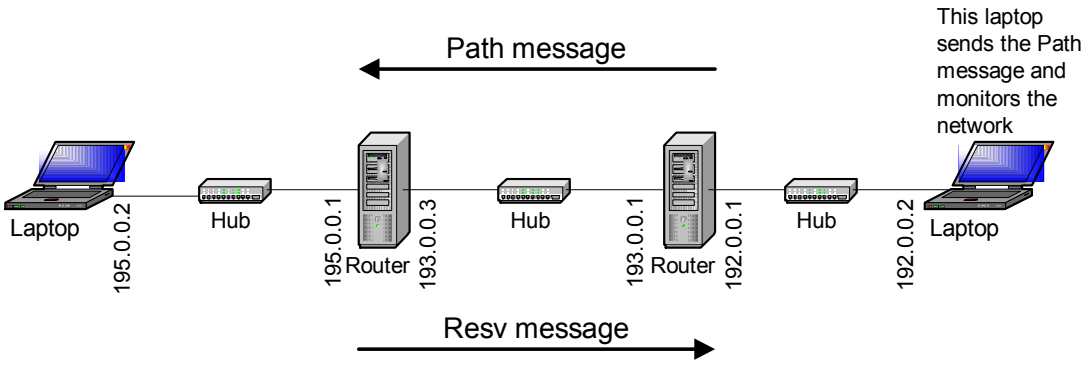

Figure A-1. Testbed for verifying the IntServ functionality.

As we can see from the testbed in Figure A-1, the source address of the Resv message received by laptop 192.0.0.2 should be 192.0.0.1. By studying the Resv message on the Network monitor in Figure A-2, we can see that our assumptions are correct. The router closest to the network monitor has intercepted the Resv message and changed the source address to the one of its output interface.

```
+ETHERNET: ETYPE = 0x0800 : Protocol = IP:  DOD Internet Protocol
-IP: ID = 0x223B; Proto = 0x2E; Len: 116
   IP: Version = 4 (0x4)
   IP: Header Length = 20 (0x14)
   IP: Precedence = Routine
   IP: Type of Service = Normal Service
   IP: Total Length = 116 (0x74)
   IP: Identification = 8763 (0x223B)
  +IP: Flags Summary = 0 (0x0)
   IP: Fragment Offset = 0 (0x0) bytes
   IP: Time to Live = 255 (0xFF)
   IP: Protocol = 0x2E
   IP: Checksum = 0x191D
   IP: Source Address = 192.0.0.1
   IP: Destination Address = 192.0.0.2
   IP: Data: Number of data bytes remaining = 96 (0x0060)
+RSVP: RSVP Reservation message
```

Figure A-2. Printout from Network Monitor showing the source IP address of an RSVP message.

By studying the body of the Resv message in Figure A-3, we can see that no non-RSVP router is detected. The IP destination address is 195.0.0.2 because this address is kept from the originating Path message, and is therefore the original source address of this message. The service requested from the application is Controlled Load, which is the only IntServ service supported by the ALTQ routers.

```
+ETHERNET: ETYPE = 0x0800 : Protocol = IP:  DOD Internet Protocol
+IP: ID = 0x223B; Proto = 0x2E; Len: 116
-RSVP: RSVP Reservation message
  +RSVP: RSVP Header
  -RSVP: Object Class = SESSION
    +RSVP: Object Header
     RSVP: IP Dest Address = 195.0.0.2
     RSVP: Protocol Id = 17 (0x11)
    -RSVP: Flags = 1 (0x1)
       RSVP: .......1 = E_Police flag set
       RSVP: ...0.... = No non-RSVP router detected
       RSVP: ..0..... = No non-RSVP router detected
     RSVP: Dest Port = 16430 (0x402E)
  +RSVP: Object Class = RSVP_HOP
  +RSVP: Object Class = TIME_VALUES
  +RSVP: Object Class = STYLE
  -RSVP: Object Class = FLOWSPEC
    +RSVP: Object Header
    +RSVP: Main Header
    -RSVP: Service Element Header
       RSVP: Service number = CONTROLLED LOAD SERVICE
       RSVP: Length in 32-bit words = 6 (0x6)
```

Figure A-3. The body of the Resv message.

The last validation to show that the routers support IntServ is done with Chariot. The first graph shows the lost data for a 64 kbps VoIP flow using IntServ Best Effort. Simultaneous with this flow we are running 6 TCP flows over a 1Mbps bottleneck link to provoke lost packets for the VoIP flow.

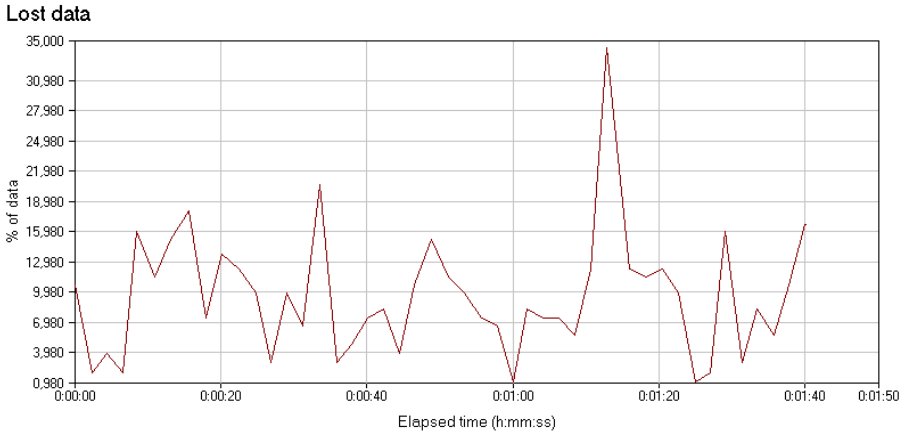

Figure A-4. Lost data for a 64 kbps VoIP flow using IntServ Best Effort.

Then we changed the QoS for the VoIP flow to IntServ Controlled Load, and ran the test again.

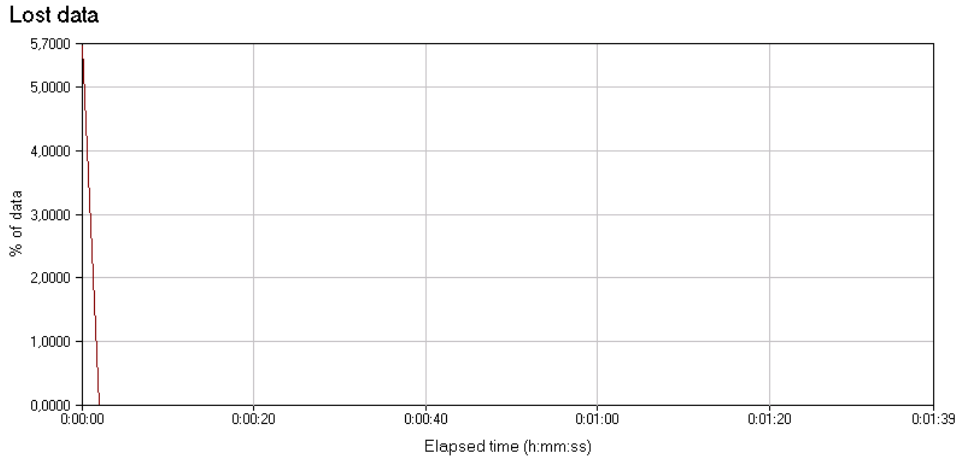

Figure A-5. Lost data for a 64 kbps VoIP flow using IntServ Controlled Load.

As we can see from Figure A-4 and Figure A-5, using Controlled Load dramatically changed the QoS delivered to the VoIP flow. The spike of lost packets (5.7 %) in the beginning is probably a result of the TCP flows initiating a slow start to get the maximal bandwidth before the reservation of the VoIP flow is completed. After this, the Controlled Load flow runs with no loss throughout the test.

A.2 Differentiated Services

To make sure that the testbed really provides the services of DiffServ, we perform a validation. The ingress router in a DiffServ network always marks all incoming packets with a per-hop behavior (PHB). Each PHB has a unique bit pattern in the DiffServ Code Point (DSCP), which is the six most significant bits of the ToS-field in the IPv4 header. When the packet arrives at its destination, we look at the DSCP to verify that a PHB has been set.

An application sends TCP-packets from 192.0.0.2 to 195.0.0.2, shown in Figure A-6. All the packets' DSCP bits are initially 0. The ingress router's conditioner is configured to mark TCP traffic from 192.0.0.2 to 195.0.0.2 with the DSCP pattern 001010 (AF11). Figure A-6 below shows a transcript from the Network Monitor that verifies our assumptions. The packet on top is sent from the source and the one below is received at the destination.

The hexcode marked in black is the IP header, and we can see that the second octet in the header, which is the ToS-field, has been changed from 00_{16} to 28_{16} by the conditioner. We can also see that the Time-to-Live octet has been decremented from 80_{16} to $7E_{16}$ caused by two router hops. The checksum has changed from $C0B6_{16}$ to $C28E_{16}$. The rest of the header is the same, and so is the data succeeding the IP header.

```
00 A0 24 AE B7 5D 00 50 DA B0 A0 36 08 00 45 00
00 28 B7 14 40 00 80 06 C0 B6 C0 00 00 02 C3 00
00 02 09 A3 05 BD CB D1 02 44 8A 78 08 12 50 10
44 0C 78 C3 00 00

00 10 5A FC 85 75 00 08 C7 BC 9F A8 08 00 45 28
00 28 B7 14 40 00 7E 06 C2 8E C0 00 00 02 C3 00
00 02 09 A3 05 BD CB D1 02 44 8A 78 08 12 50 10
44 0C 78 C3 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure A-6. A TCP-packet before and after it has been marked by the conditioner.

So far, we know that the edge routers perform conditioning of packets before they traverse the network. We also need to validate that the packets get the treatment they have been promised by the network. We start with a 1 MB bottleneck link congested with file-transfers, HTTP traffic and real-time traffic, all classified with default priority (Best Effort). Figure A-7 shows the three UDP flows that consist of two 50 Kbps VoIP flows (gray line and black dash line) and one 128 Kbps real-audio flow (cyan line). As we can see, they all suffer from severe congestion in the bottleneck.

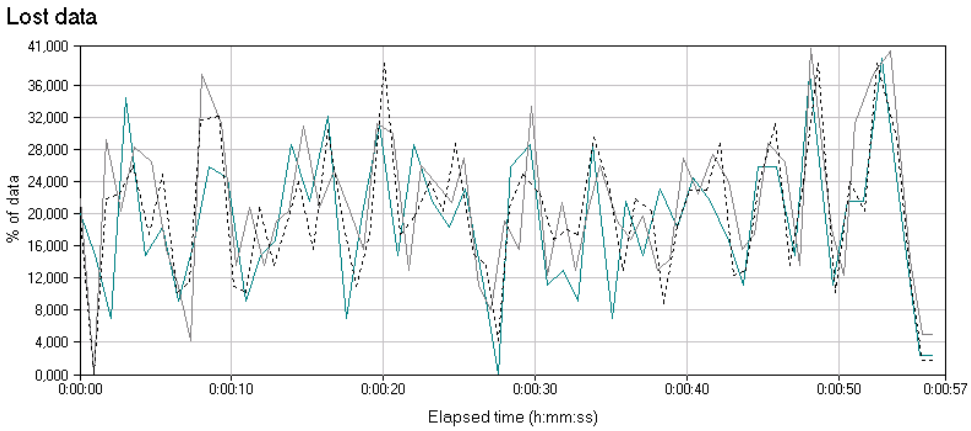

Figure A-7. Three UDP flows that suffer extensive packet loss.

In Figure A-8, the situation has been changed considerable. Now, only the 128 Kbps real-audio flow has packet loss. The two 50 Kbps VoIP flows are now classified as AF1x, and are inside a required profile of 200 Kbps or less, which mean they get high priority and no packet loss. The 128 Kbps real-audio flow is classified as AF4x, a profile that treat all packets with a bitrate above 70 Kbps with high drop precedence. This class is meant for 12 Kbps VoIP flows, so the 128 Kbps real-audio flow is way out of profile. The result is low priority of the packets and extensive data loss.

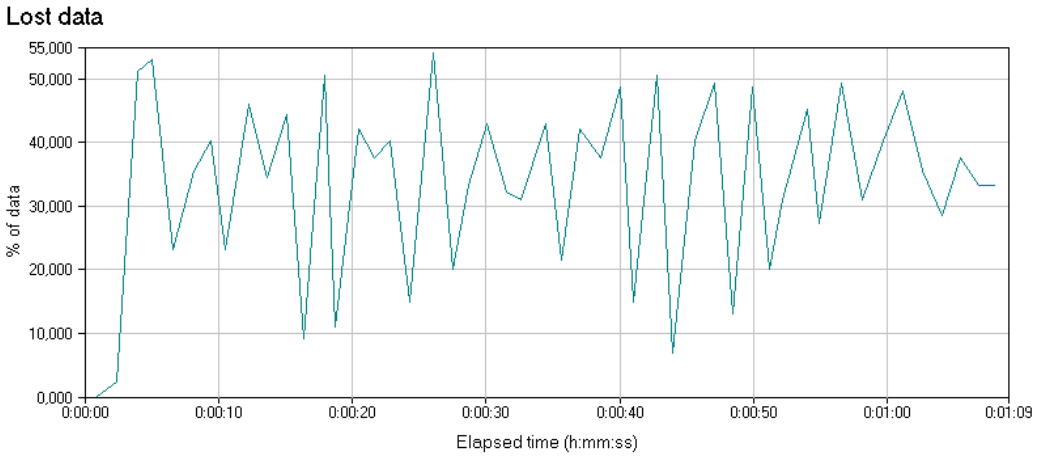

Figure A-8. Three UDP flows, one with packet loss, and two without packet loss.

The validation proves that the testbed network supports DiffServ.

Appendix B - Preliminary tests

This appendix describes the testbed equipment, some rules to follow for successful testing, basic tests and at last more extensive tests.

B.1 Rules to follow for successful testing

We start with very simple tests and add the complexity as things develop the way we expect them to (or learn what is not functioning the way we thought it would). Starting off with a complex test suite only makes the interpretation of the results difficult and may result in inaccurate conclusions.

As all non-test related processes on the testbed computers will degrade the performance, screen savers and other unnecessary processes should be disabled. To know when bandwidth is limiting the connection, we can look at the number of transactions being processed at an endpoint. Normally, when we add a pair to the endpoint, the total number of transactions should be increasing, but if the transactions are just being spread out across the total number of pairs, we know that bandwidth is limiting the connection.

The application scripts can generally be divided into two basic categories or types: response time centric scripts and throughput centric scripts. Using the proper script will ensure that we obtain the desired results. A script that emphasizes response time may not be able to flood the bandwidth of a pipe, and a script that emphasizes throughput may not show us the best possible response time the pipe is capable of achieving.

The Console computer can be used as an endpoint, but normally it has enough work to do without having to serve as an endpoint. In most scripts, Endpoint 1 (E1) does the timing and data collection for the test and passes this information back to the console, while Endpoint 2 (E2) is just responding to the data flows as defined in the application script. To avoid interfering the traffic under test, we should wait with the collection of data showing the results until after the test.

The MTU size of all the links in the network is 1500 bytes. For low bandwidth links, it may take a while to put 1500 bytes on the wire and the latency goal might be impossible to achieve. However, 10 Mbps links like the one we are using only use 1.2 ms to put 1500 bytes on the link, and even if we limit the capacity that is allowed to use on the link to 1 Mbps, the speed on the link and delay would still be the same. Therefore, we do not change the MTU size from 1500 bytes.

B.2 Forwarding mechanisms & policies

The routing table is built with RIPv2. The IntServ and DiffServ routers use Class Based Queuing (CBQ) for queuing and scheduling.

The Integrated Services network in the testbed has very simple policing, and there are no Guaranteed Services, due to missing support in the forwarding mechanisms of the routers. We have decided that only real-time traffic can be classified as Controlled Load, and all other traffic is treated with no more privilege than ordinary Best Effort. The Controlled Load class is 40% of the bottleneck bandwidth, and the Best Effort class uses the remaining 60 %.

Defining the policies in the Differentiated Services network is a bit harder than for IntServ. The Expedited Forwarding (EF) class should be reserved for urgent, low-bandwidth traffic,

like VoIP. There are four Assured Forwarding (AF) classes, each with three precedence levels.

Real-time traffic and other more or less interactive traffic are being classified with EF or AF behavior. All other traffic will be classified as Best Effort. To avoid treating IntServ and DiffServ too differently, the EF and AF classes in total uses 40 % of the bandwidth just like the Controlled Load service in IntServ.

B.3 Measurements in a silent network

B.3.1 Evaluate the Effects of Bi-directional Traffic

The objective is to see if bi-directional traffic changes the testbed's behavior. We use the first two tests described in section 5.2, as a starting point, and add flows to have traffic in both directions. To create bi-directional traffic, we simply add another pair using filesndl or creditl and reverse E1 and E2.

This test is intended as a validation of how well the network handles bi-directional traffic. The behavior experienced in this test was used to make us aware of weaknesses in how the bandwidth is limited by Dummynet and ALTQ.

The result from the Best Effort network model shows the same results as found in the two tests described in the thesis. When we use Dummynet to limit the bandwidth, traffic in both directions is considered when limitations of the bandwidth are being enforced.

For IntServ and DiffServ, the results show us that ALTQ perform bandwidth limitations independently in each direction. Since the bandwidth limitations only are being enforced at the incoming interface at the router, the outgoing interface only limits flows in the opposite direction. Then, the router will allow 1 Mbps in each direction, and since the response time is affected by the bandwidth, the response time of this test will stay the same as in the first test described in the thesis. However, it is important to notice that as the number of flows increase, the processing delay in the routers will raise and result in higher response times.

B.3.2 Differences in scalability

Differences in scalability between the service models are an interesting issue. A network should be able to scale well as the number of users increase.

To test the scalability of the service models, we measure response times for all of the flows listed in Table B-1. We use 10, 25 and 50 TCP flows sent by a creditl script all demanding the same type of QoS. This approach will hopefully put strain on the routers and enforce different behavior for the service models.

Limited bandwidth will cause a higher response time. To avoid this effect, we set the bandwidth for the testbed to 10 Mbps to avoid influencing the flows' response times.

Table B-1. Differences in scalability.

	Best Effort	**IntServ BE flow**	**IntServ Cntl flow**	**DiffServ BE flow**	**DiffServ EF flow**	**DiffServ AF flow**
Mean RT (10 flows) [ms]	3.68	3.05	5.02	2.95	4.10	3.02
Mean RT (25 flows) [ms]	8.16	8.81	8.75	8.64	10.62	8.64
Mean RT (50 flows) [ms]	15.31	17.91	17.29	18.07	21.44	17.36

From Table B-1, we can see that there is a linear increase in the response time as more flows are added. The Controlled Load flow differs from this tendency with higher response times at a low amount of flows. The phenomenon is the same as the one we commented in section 5.3.2, but as we can see, the response time in the Controlled Load flow approximates the response time of the other flows when the number of flows increases.

IntServ is known to scale poorly, but in small-scale networks like the testbed we use, this does not seem to be provable.

Best Effort has the lowest overall response time, and this is not a surprise; it has less processing in the routers than the other service models. However, what is surprising is that the EF flow has the highest response time. If we take a closer look at the EF flows, we can see that the response time between the individual flows varies. The response time of other flows does not vary with such an extent, as we can see in Figure B-1.

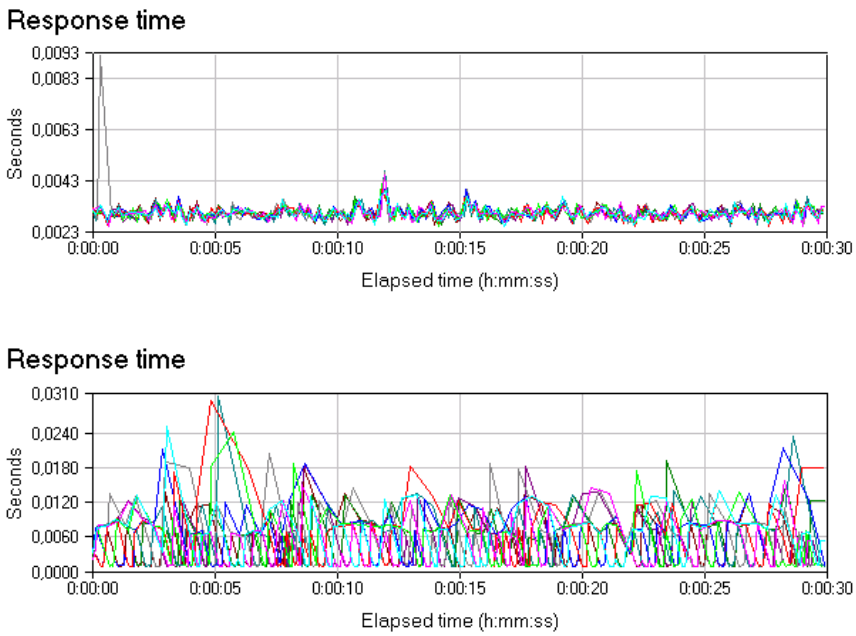

Figure B-1. Response time graphs of 10 AF flows (top) compared with 10 EF flows (bottom).

We can see from Figure B-1 that EF gets much more fluctuating response times than the AF class does. As we know, EF is the premium service and therefore we are quite surprised regarding this behavior. The explanation of this is given in subsection 5.3.5.

B.3.3 Bandwidth sharing - a file transfer versus VoIP

The objective is to see if file transfer traffic would seriously reduce the Quality of Service of a voice conversation. We send a 128 Kbps voice conversation and one persistent TCP file transfer through a 1 Mbps bottleneck network. The test is run for all the service models with premium service levels for the voice conversation when provided.

The script used to simulate a file transfer is filesndl. It uses TCP to send 1500 byte packets. A script called VoIP, which sends RTP packets with a data load of 40 bytes at a rate of 128 Kbps, simulates the voice conversation. The size of the EF and Controlled Load classes are changed to 50 % of the available bandwidth to make sure we get no packet loss. (A 128 Kbps stream uses twice the bandwidth due to packet overhead.)

Table B-2. Bandwidth sharing - a file tranfer versus a voice conversation.

	Best Effort	IntServ	DiffServ
Mean Throughput (file transfer)	678 Kbps	503 Kbps	503 Kbps
Mean Throughput (voice)	102 Kbps	126 Kbps	130 Kbps
Bytes lost (voice)	6.115 %	0.105 %	0.000 %
Mean Delay (voice)	3.143 ms	3.580 ms	0.542 ms

From Table B-2, we can see that the Best Effort network does not provide a satisfactory voice conversation. There is packet loss and the mean delay is higher than with DiffServ. The file transfer throughput degrades with the service models supporting QoS, since more of the bandwidth is reserved for real-time traffic and other traffic with special demands.

IntServ and DiffServ treat real-time traffic with high priority, and there are almost no bytes lost. The few bytes lost with IntServ in the beginning of the transmission are before the RSVP reservation is established. Using IntServ as the Service Model, it was expected to have a delay just above 5 ms, due to the reason explained in section 5.3.2. However, the mean value is measured for 38 seconds while the flow only lasted for 30 seconds, giving the last 8 seconds 0 delay, and an average of the 38 seconds is 3.580 ms. The reason why the DiffServ flow also has very low delay is related to the same explanation.

B.4 Measurements during overload conditions

To provide overload conditions, we use background flows as shown in Table B-3 below. The flows are chosen to represent what we think could be a possible situation in a real network, but are not based on any particular measurements of real network situations. All the flows are treated with no more than Best Effort service quality in the network independent of the service model used.

Table B-3. The background flows providing overload conditions.

Name	Characteristics
filesndl	Long held adaptive reliable traffic flow, two TCP flows
HTTPtext	Short duration reliable transaction, six TCP flows
VoIPG729	Externally controlled streaming, two 8 Kbps UDP streams.

B.4.1 Response time during overload conditions

The objective with this task is to see if the service models can provide low delay even under overload situations. We study differences between the treatments of real-time traffic when there is a background load with a total bandwidth just below the available bandwidth. We measure delay with a creditl script, sending a TCP-flow.

Table B-4. Response time during overload conditions.

	Best Effort	IntServ BE flow	IntServ Cntl flow	DiffServ BE flow	DiffServ EF flow	DiffServ AF2 flow
Mean Delay	363.19 ms	166.48 ms	5.61 ms	147.73 ms	3.56 ms	3.12 ms

The results in Table B-4 show that traffic in the IntServ Controlled Load flow and the DiffServ AF and EF classes, to a certain amount of traffic, provide low response time. The results can be seen in relation to section 5.3.4, where the same flows have good streaming characteristics.

B.5 Isolated performance measurements of IntServ

B.5.1 RSVP message overhead

The object is to see if the RSVP message overhead is significant compared to the amount of data transmitted using the Controlled Load service.

This test is performed using Microsoft Network monitor to analyze the traffic of a NetMeeting VoIP flow. We filter the traffic to get all the RSVP messages that is traversing the network at the time when the test is performed. The length of the test was 62.99 seconds (according to Microsoft Network monitor) and the RSVP packets captured are shown in Figure B-2:

Frame	Time	Src MAC Addr	Dst MAC Addr	Protocol	Description	Src Other Addr	Dst Other Addr	Type Other Addr
49	11.486517	sgrml51	LOCAL	RSVP	RSVP Path Message	IKT01-FT	IKT01-FTMOB	IP
50	11.486517	LOCAL	sgrml51	RSVP	RSVP Reservation message	IKT01-FTMOB	IKT01-FT	IP
51	11.486517	LOCAL	sgrml51	RSVP	RSVP Reservation message	IKT01-FTMOB	IKT01-FT	IP
52	11.496531	LOCAL	sgrml51	RSVP	RSVP Path Message	IKT01-FTMOB	IKT01-FT	IP
53	11.496531	LOCAL	sgrml51	RSVP	RSVP Path Message	IKT01-FTMOB	IKT01-FT	IP
54	11.496531	sgrml51	LOCAL	RSVP	RSVP Reservation message	IKT01-FT	IKT01-FTMOB	IP
254	13.479383	sgrml51	LOCAL	RSVP	RSVP Reservation message	IKT01-FT	IKT01-FTMOB	IP
255	13.479383	sgrml51	LOCAL	RSVP	RSVP Path Message	IKT01-FT	IKT01-FTMOB	IP
256	13.489397	LOCAL	sgrml51	RSVP	RSVP Path Message	IKT01-FTMOB	IKT01-FT	IP
257	13.489397	LOCAL	sgrml51	RSVP	RSVP Path Message	IKT01-FTMOB	IKT01-FT	IP
258	13.489397	LOCAL	sgrml51	RSVP	RSVP Reservation message	IKT01-FTMOB	IKT01-FT	IP
259	13.489397	LOCAL	sgrml51	RSVP	RSVP Reservation message	IKT01-FTMOB	IKT01-FT	IP
439	21.881464	sgrml51	LOCAL	RSVP	RSVP Reservation message	IKT01-FT	IKT01-FTMOB	IP
440	22.702645	LOCAL	sgrml51	RSVP	RSVP Reservation message	IKT01-FTMOB	IKT01-FT	IP
441	22.702645	LOCAL	sgrml51	RSVP	RSVP Reservation message	IKT01-FTMOB	IKT01-FT	IP
446	29.722739	sgrml51	LOCAL	RSVP	RSVP Reservation message	IKT01-FT	IKT01-FTMOB	IP
450	34.429507	LOCAL	sgrml51	RSVP	RSVP Path Message	IKT01-FTMOB	IKT01-FT	IP
451	34.429507	LOCAL	sgrml51	RSVP	RSVP Path Message	IKT01-FTMOB	IKT01-FT	IP
455	39.997514	LOCAL	sgrml51	RSVP	RSVP Reservation message	IKT01-FTMOB	IKT01-FT	IP
456	39.997514	LOCAL	sgrml51	RSVP	RSVP Reservation message	IKT01-FTMOB	IKT01-FT	IP
457	40.207816	sgrml51	LOCAL	RSVP	RSVP Path Message	IKT01-FT	IKT01-FTMOB	IP
568	60.547063	sgrml51	LOCAL	RSVP	RSVP Path Message	IKT01-FT	IKT01-FTMOB	IP
569	60.547063	sgrml51	LOCAL	RSVP	RSVP Reservation message	IKT01-FT	IKT01-FTMOB	IP
675	71.823277	LOCAL	sgrml51	RSVP	RSVP Reservation message	IKT01-FTMOB	IKT01-FT	IP
676	71.823277	LOCAL	sgrml51	RSVP	RSVP Reservation message	IKT01-FTMOB	IKT01-FT	IP
677	71.823277	LOCAL	sgrml51	RSVP	RSVP Path Message	IKT01-FTMOB	IKT01-FT	IP
678	71.823277	LOCAL	sgrml51	RSVP	RSVP Path Message	IKT01-FTMOB	IKT01-FT	IP
719	74.437035	LOCAL	sgrml51	RSVP	RSVP Reservation teardown message	IKT01-FTMOB	IKT01-FT	IP
720	74.437035	LOCAL	sgrml51	RSVP	RSVP Reservation teardown message	IKT01-FTMOB	IKT01-FT	IP
721	74.437035	LOCAL	sgrml51	RSVP	RSVP Path teardown message	IKT01-FTMOB	IKT01-FT	IP
722	74.437035	LOCAL	sgrml51	RSVP	RSVP Path teardown message	IKT01-FTMOB	IKT01-FT	IP
729	74.477093	sgrml51	LOCAL	RSVP	RSVP Path teardown message	IKT01-FT	IKT01-FTMOB	IP

Figure B-2. RSVP packets captured by Microsoft Network monitor.

Table B-5. Total number of RSVP packets captured by the Microsoft Network monitor.

Type of packet	Total number captured	Size [bytes]
RSVP Path message	12	192
RSVP Reservation message	15	116
RSVP Reservation teardown message	2	60
RSVP Path teardown message	3	104

A total of 4476 bytes of RSVP messages were transferred during the test period of 62.99 seconds. By dividing the total number of bytes transferred in the time period, we get that there is on average transferred approximately 568 bps per flow.

To draw parallels with a larger network in the future, where VoIP is more widespread, a total of 100 000 VoIP flows on an Internet backbone are not unexpected. This gives a total of about 54 Mbps RSVP messages on the link at all times.

With concern to VoIP that with maximum compression has a bitrate of 8 Kbps, this makes RSVP add an additional 6.9 % data rate (average) to the VoIP flow. For a 64 Kbps VoIP flow the additional overhead will be 0.87 %.

The conclusion must be that the data overhead introduced by RSVP not is a problem. The overhead introduced by small real-time packets alone is usually about 50 %, thus a couple of percent caused by RSVP messages do not really matter.

B.6 Isolated performance measurements of DiffServ

B.6.1 Packet loss with different drop precedences

The objective is to see how the drop precedence changes when varying the bitrate of a flow with a particular Per-Hop Behavior (PHB). The test network is overloaded all the time and all other parameters except the nominal bitrate (NBR) is kept unchanged during the measurements. The NBR values are selected so that all three Drop Precedence (DP) levels of the DiffServ network are examined.

The test is performed with a 32 Kbps VoIP flow and the same background traffic as used in previous tasks. The amount of packet loss as a function of the nominal bitrate (NBR) is presented in Figure B-3.

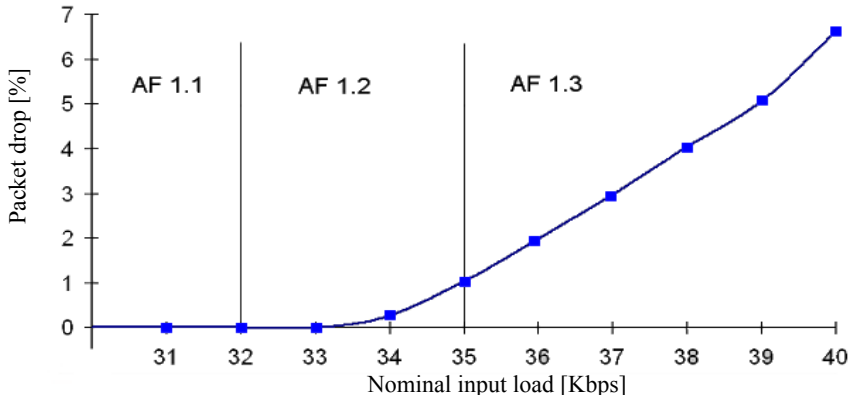

Figure B-3. Packet loss with different drop precedences.

The drop precedences given by AF1 are 64 Kbps and 70 Kbps, but since the VoIP flow sends 40 byte packets with 40 bytes of overhead the limits between the drop precedences actually is 32 Kbps and 35 Kbps the way we see it through Chariot (Chariot measures “Goodput”).

Figure B-3 shows that the drop precedences actually work as expected. AF11 has no drops, AF12 starts to drop packets, and AF13 drop packets with a linear increase.

Appendix C - Fair resource allocation

C.1 The service models compared

In the thesis, we evaluated fairness between service classes internally in each service model. We are now going to evaluate the internal differences in fairness from each service model with each other.

As Best Effort has only one service class, it has no other internal classes to be compared with. Therefore, the result is that the fairness for Best Effort always will be 100 %.

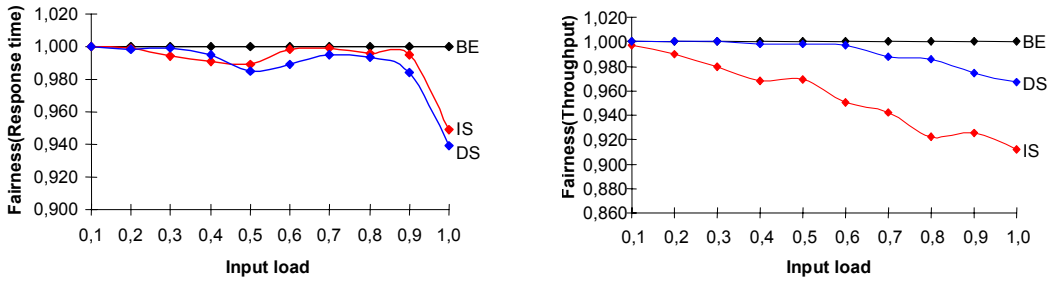

Figure C-1. Comparing internal differences in fairness against other service models.

For real-time flows (Figure C-1, left hand side), we can see that IntServ and DiffServ have fairness curves that are quite similar. Both have two points at the graphs where the fairness is lower than the Best Effort graph. Thus, the difference in QoS delivered internally in IntServ and DiffServ is high at these points.

For networks that are expected to have loads lower than 50 %, Best Effort will be a suitable choice, even for real-time flows. On the other hand, it is important to notice that the delay for lower loads nevertheless is quite small, so EF and Controlled Load also may be a suitable choice.

For loads higher than 90 %, EF or Controlled Load will be an excellent choice. These service classes will for extreme input loads show favor for real-time flows and provide much better QoS, in relation to delay, than flows classified as Best Effort. The drop for EF and Controlled Load at 100 % visualizes this, but this also means that other traffic will be disfavored in this situation.

For file-transfers, IntServ is the definite loser in concern to fairness. This can be explained by Figure 46, which visualizes the fact that TCP flows utilize reserved bandwidth worse than if the flows were running Best Effort.

Differentiated Services degrades in fairness because the bandwidth of the EF class is limited at 300 Kbps (50 % of the available bandwidth in this class). It can be read more about this in 5.5.4.3.

C.2 Fairness with increasing number of real-time flows

In the following graphs we wish to study the Expedited Forwarding class in DiffServ and the Controlled Load class in IntServ. To create equal conditions for both IntServ and DiffServ, we set both classes to hold 40 % of the total bandwidth. To generate traffic for this test, we load the 1.6 Mbps Controlled Load and EF classes with one 400 Kbps flow and measure throughput and delay for each scenario. This is repeated with an increasing number of flows until we are running eight flows, or twice the class' capacity. In addition, we are using 2 TCP file transfer flows to occupy the capacity of the Best Effort classes.

It is important to remember the nature of Controlled Load, where all flows have to reserve resources and that when the capacity of the class is full, no more flows are able to do this. When we compute the fairness for Controlled Load, flows that do not receive increased service per definition experience a fairness equal to zero.

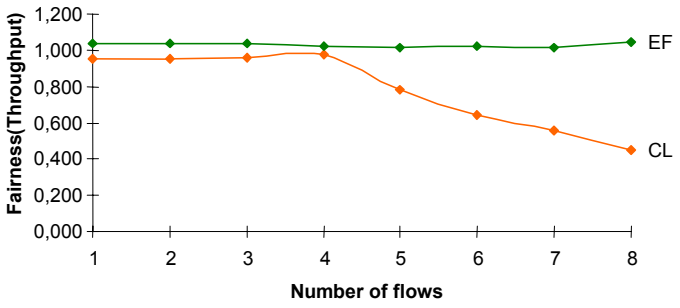

Figure C-2. Fairness as a function of throughput showing Controlled Load vs. the EF class.

Figure C-2 clearly shows that all flows exceeding the four flows that are allowed to make reservations will not receive Controlled Load. As Expedited Forwarding has no reservations, all flows demanding this class will share the bandwidth, resulting in a stable fairness of 100 %.

Appendix D - Installation of software routers

This appendix is meant as an aid to install the programs used in our testbed. For detailed descriptions on how to use them, use the manual pages on the computer.

D.1 Installation and configuration of FreeBSD 3.3

1. Start the computer with the installation CD in the CDROM. If the computer is not configured to try to boot from the CDROM, you have to change this in the BIOS.
2. Chose Start Kernel configuration in full-screen visual mode and delete the drivers that causes conflicts. Type Q to exit and Y to save configuration.
   (you then have to wait a while)
3. When the "/stand/sysinstall Main Menu" appears, chose novice(2) and then press Enter (OK).
4. Then you will enter the "FDISK Partition Editor" where you can partition your disk. For out use, we chose to use the whole disk for FreeBSD. This is done by pressing A and then confirm by pressing Enter (Yes). You can then press Q to Finish.
5. "Install Boot Manager for drive wd0?"
   Most likely the BootMgr will be default, and it is this we want to use, so you can press Enter and then confirm by pressing Enter once more.
6. "FreeBSD Disklabel Editor"
   Press A to use "Auto Defaults for all!" and finish by pressing Q.
7. "Choose Distributions"
   Here you chose X-Kern-Developer by using the arrow keys and pressing Space to activate.
8. "User Confirmation Requested"
   We do not want to install DES cryptographic software, so here you press No.
9. The next window will ask you if you want to install the FreeBSD ports collection. We choose to do so.
10. "XFree 3.3.5 Distribution"
    This will be configured right on the basis that you chose as the distribution in point 7. Choose Exit and press Enter twice.
11. "Choose Installation Media"
    As we are installing from CD-ROM we choose 1 and presses Enter twice. The machine will now format the disk and start installing FreeBSD. This takes a while, so it's now time for a coffee break.
12. Press Enter to continue.
13. Choose No to configure any Ethernet or SLIP/PPP network devices.
14. Choose Yes for "Will the machine be an IP gateway (e.g. will it forward packets between interfaces?" We do not want to allow anonymous FTP connections so we press No for both this, NFS server and NFS client.
15. "Would you like to customize your system console setting?"
    Here we press Yes and enters a menu. Chose Keymap to specify that we are using a Norwegian keyboard. Then press E and Enter.
16. As time zone is of no importance to us, we will not set the machine's time zone. Press No.
17. Press Yes to enable Linux binary compability.

18. If your machine has a mouse attached to it, choose Yes.
    Use these settings: → Type → MouseSystem
    → port → PS/2
    → enable (if not working, try with other mouse configuration).
    → exit
19. "Would you like to configure your X server at this time?"
    Choose Yes and select XF86Setup. Press OK and Enter twice to enter the configuration menu. Use the following configuration:
    → Mouse → Protocol → SysMouse
    → MouseDev → /dev/sysmouse
    → Emulate 3D buttons → Apply
    → Keyboard →Norwegian→Apply
    → Card →Chose the graphics card you have
    → Monitor →Chose what resolutions the monitor can support.
    → Modeselection → chose the resolutions and color depth you want to use.
    → Done
    If starting the server goes well, you can save the configuration and exit.
20. [Do you want to create an "X" link to the SVGA server?] → Yes
21. Chose KDE.
22. Packages → Languages → tcl-8.0.5
    → shells → bash
    → tcl80 → chose all
    → tk80 → chose expect-5.30 and p5-Tcl-Tk-b2
    Choose Install and OK.
23. If you want to create logins for other users you can do that, but for our testbed this was not considered necessary.
24. [Root password] Type the password you want to use. This login gives you administrator rights, which in FreeBSD language means that you have extended rights. This means that you can delete for example the whole hard disk and you will never be asked to confirm anything, so you have to be careful.
25. "Visit the general configuration menu for a chance to set any last options?"
    Choose No if everything has succeeded.

[The machine reboots]

Configuring X-windows

When your machine have booted, you will be prompted for your login. Type root as login and the password you typed in point 11 as your password.
Type *startx* to start the X-windows.

Resolution and color depth.

If you are not satisfied with the resolution and color depth of the desktop, you can open a terminal window [må vise icon her] and type the following:

```
#cd /
#stand/sysinstall
```

The window you used for your installation of FreeBSD is now opened.
Chose *XFree86*, which is located on the bottom of the menu and select *XF86Setup*.

You will now enter the same menu you were in at point 6 in the installation. Do the changes in *Monitor* and *Modeselection* and press done to test. If this goes well, you can save the setup and exit the *XF86Setup*.

Configuration of network interfaces.
Since you are going to use this machine as a software router, it will be necessary to configure the network interfaces. This is done by opening a terminal window [må vise icon her] and type the following:

#cd /
#stand/sysinstall

The window you used for your installation of FreeBSD is now opened. Select *Networking* and chose *interface*.
You will now be asked if the network is already configured. Here you will answer no and chose the interface you want to configure. It is important that you do not select DHCP, because this will cause a lot of unnecessary error messages when you are using your testbed. Enter the hostname, IP address and IP netmask that you want to use for this network interface. When you are done, you will be asked if you want to bring the interface up; select yes. If you have more network adaptors, you can select the next you want to configure and repeat the configuration explained.

D.2 Installation of ALTQ 2.2

This release supports FreeBSD-3.4-RELEASE and 4.0-RELEASE.

Making ALTQ-kernel.
Put a fresh kernel source in /usr/src/sys-altq (recommended directory name), which is done by typing:

cd /usr/src
mkdir sys-altq
cd sys
tar cvf - . | (cd ../sys-altq; tar xf -)

Copy and extract the ALTQ 2.2 distribution from the CDROM.
#cd /usr
#mkdir install
#cd /cdrom (if the CDROM is not mounted, this can be done by typing #mount /cdrom)
#cp altq-2.2.tar /usr/install/
#tar xf altq-2.2.tar

Apply the altq kernel patch to the kernel source
cd /usr/src/sys-altq
patch -p < /usr/install/altq2.2/sys-altq/sys-altq-freebsd-X.X.patch

“X.X” corresponds to your FreeBSD version (3.4 in our installation).

Copy the altq files to “sys-altq/altq” directory.

mkdir altq
cp /usr/install/altq2.2/sys-altq/altq/* altq/

If you are not familiar with how to make a kernel, consult the FreeBSD handbook, configuring the FreeBSD Kernel section http://www.freebsd.org/handbook/handbook.html)

Make and install the new kernel.

```
# cd i386/conf
# config ALTQ
# cd ../../compile/ALTQ
# make depend
# make clean
# make
# make install

# shutdown -r now
```

For FreeBSD-3.x or later, the default configuration includes no queuing discipline but disciplines can be loaded at run time using KLD.
Disciplines can also be built-in by specifying them in the kernel configuration file.

Making (or re-making) altq devices

```
# cd /usr/install/altq2.2/
# sh MAKEDEV.altq all
```

The altq major device numbers for FreeBSD is 96. This is the official number from FreeBSD (the device major number is defined in sys-altq/altq/altq_conf.c).

Loading altq queueing desciplines (FreeBSD-3.x or later)

To make the altq KLD modules:

```
# cd /usr/src/sys-altq/modules/altq
# make
# make install
```

The modules are installed in the "/modules" directory.
The altq modules have names starting with "altq_" (e.g., altq_cbq.ko).

altqd tries to load required modules automatically so that you do not need to "kldload" modules explicitly.

In case you want to manipulate the modules by hand,

To load a module:

```
# kldload altq_cbq
```

To check the loaded modules:

```
# kldstat -v
```

To unload a module:

```
# kldunload altq_cbq
```

Installing userland tools

```
# cd /usr/install/altq2.2/
# make
# make install
```

By default, the altq tools are installed to /usr/local/{bin,sbin}.
You can change it by make install PREFIX=your_install_prefix

D.3 Installation of additional software

Tele traffic tapper (ttt)
Install libpcap-0.6.2
cd /cdrom
cp libpcap-0.6.2.tar /usr/install/
cd /usr/install/
tar xf libpcap-0.6.2.tar
cd libpcap-0.6.2
./configure
make
make install

Install blt2.4u
cd /cdrom
cp blt2.4u.tar /usr/install/
cd /usr/install/
tar xf blt2.4u.tar
cd blt2.4u

To make the installation work, you have to alter the configuration script which is named *configure*.

Line 2806 and 2807 must be altered:
Originally reads:
"${blt_with_tcl_libraries}/lib${TCL_LIB_NAME}.${SHLIB_SUFFIX}" \
"${blt_with_tcl_libraries}/lib${TCL_LIB_NAME}.a"
Must be changed to read:
"${blt_with_tcl_libraries}/libtcl82.${SHLIB_SUFFIX}" \
"${blt_with_tcl_libraries}/libtcl82.a"

Line 2836 and 2837 must be altered:
Originally reads:
"${blt_with_tk_libraries}/lib${TK_LIB_NAME}.${SHLIB_SUFFIX}" \
"${blt_with_tk_libraries}/lib${TK_LIB_NAME}.a"
Must be changed to read:
"${blt_with_tk_libraries}/libtk82.${SHLIB_SUFFIX}" \
"${blt_with_tk_libraries}/libtk82.a"

When you have done this, you can continue the installation.

./configure –with-tcl=/usr/local/lib/tcl8.2/ --with-tk=/usr/local/lib/tk8.2/ --with-tcllibs=/usr/local/lib/ --with-tklibs=/usr/local/lib/

Again, there seems to be some problems with the installation scripts. The following files must be altered to make the installation work:

/usr/install/blt2.4u/src/Makefile
Line 41 and 42 originally reads:
LIBS = -L/usr/local/lib/ -ltk8.2 –ltcl8.2 –L/usr/X11R6/lib –lX11 –lm
TCL_ONLY_LIBS = -L/usr/local/lib/ -ltcl8.2 –lm

Must be altered to read:
LIBS = -L/usr/local/lib/ -ltk82 –ltcl82 –L/usr/X11R6/lib –lX11 –lm
TCL_ONLY_LIBS = -L/usr/local/lib/ -ltcl82 –lm

/usr/install/blt2.4u/src/shared/Makefile
Line 19, 33 and 34 originally reads:
SHLIB_LD_LIBS = -L/usr/local/lib/ -ltk8.2 –ltcl8.2 –L/usr/X11R6/lib –lX11 –lm
LIBS = -L/usr/local/lib/ -ltk8.2 –ltcl8.2 –L/usr/X11R6/lib –lX11 –lm
TCL_ONLY_LIBS = -L/usr/local/lib/ -ltcl8.2 –lm

Must be altered to read:
SHLIB_LD_LIBS = -L/usr/local/lib/ -ltk82 –ltcl82 –L/usr/X11R6/lib –lX11 –lm
LIBS = -L/usr/local/lib/ -ltk82 –ltcl82 –L/usr/X11R6/lib –lX11 –lm
TCL_ONLY_LIBS = -L/usr/local/lib/ -ltcl82 –lm

After these changes has been done, the installation can proceed.
#make

To test that blt2.4u is installed correctly, you can run a demo.
cd demos
./graph1.tcl

If this displays a graph, blt is installed correctly. Proceed with installation.
#make install

Install ttt
cd /cdrom
cp ttt-1.6.tar /usr/install/
cd /usr/install/
tar xf ttt-1.6.tar
cd ttt-1.6
./configure –with-blt=/usr/install/blt2.4u/src/
make
cp /usr/install/blt2.4u/library/bltGraph.pro /usr/local/lib/blt2.4/

D.4 Installation of NIST switch

This version of the NIST Switch software works with FreeBSD 3.3 and ALTQ 2.0 along with RSVPD 4.2a4. In order to install, follow these steps:

Install FreeBSD 3.3 and ALTQ 2.0 on the machine.

FreeBSD 3.3 is in fact quite difficult to get, because it is revoked as a result of security problems with this version. We downloaded an ISO image of it from ftp://ftp.fu-berlin.de in the directory /unix/FreeBSD/releases/i386/ISO-IMAGES/. Here you will find the file named 3.3-install.cd0 that is a 675 Mb file.

ALTQ 2.0 is available from the Sony site at ftp://ftp.csl.sony.co.jp/pub/kjc/. Before you go any further, make sure you can build and boot new kernels!

Note: You must use the versions of FreeBSD and ALTQ indicated.
Extract the NIST switch installation into the directory /usr/install/
cd /cdrom
cp nistswitch-0.2.tar /usr/install/

```
# cd /usr/install/
# tar xf nistswitch-0.2.tar

#cd /usr/src

copy sys-altq to sys-mpls
#cp -r sys-altq sys-mpls

#cd sys-mpls
#patch -p < /usr/install/nistswitch-0.2/patch-mpls
```

Modify the ALTQ kernel configuration under i386/conf if you wish.
Then in any case, configure and make the kernel. If you have forgotten how to do this, it is described under the installation of ALTQ.

Copy the .h files from /usr/src/sys-mpls/net to /usr/include/net
and from /usr/src/sys-mpls/netinet to /usr/include /netinet

```
#cp /usr/src/sys-mpls/net/*.h /usr/include/net/
#cp /usr/src/sys-mpls/netinet/*.h /usr/include/netinet/
```

Make the label module

```
#cd /usr/src/sys-mpls/modules/label
#make clean
#make depend
#make
#make install
```

Copy the altq.conf and label.conf in /etc

```
#cp /usr/install/nistswitch-0.2/*.conf /etc/
```

Make libaltq under rsvpd-kit-991103

```
#cd /usr/install/nistswitch-0.2/
#gunzip rsvpd_kit.tar.gz
#tar xf rsvpd_kit.tar
#cd rsvpd-kit-991103
#cd libaltq
#Make
```

Make rsvpd under rel4.2a4l/rsvpd and all utilities under rel4.2a4l/labeltest

```
#cd /usr/install/nistswitch-0.2/
#gunzip rel4.2a4l.tar.gz
#tar xf rel4.2a4l.tar
#cd rel4.2a4l/rsvpd
#Make
#cd ..
#cd labeltest
#Make
```

Reboot the kernel, load the label module using

```
#kldload label
```

See HOW_TO (in the directory /usr/install/nistswitch-0.2/)for an example on how to use the software.

Appendix E - Software

CD1

Thesis documentation:

- report.pdf (Report in PDF format)
- report.doc (Report in Word format)
- poster.doc (Poster in Word format)

Router software:

- altq-2.2.tar (QoS support for the FreeBSD router)
- tele traffic tapper 1.6 (ttt) (Traffic monitor for FreeBSD)
- libpcap-0.6.2.tar (Interface for user-level packet capture in ttt)
- BLT2.4u.tar (A library extension to ttt)
- nistswitch-0.2.tar (MPLS extension for ALTQ)
- rsvpd.rel4.2a4-1 (RSVP extension for ALTQ)
- altq.conf.intserv (Router configuration file for IntServ in ALTQ)
- altq.conf.diffserv (Router configuration file for DiffServ in ALTQ)

Host software:

- Chariot_Inst.exe (Evaluation version of Chariot)
- gsendw32.exe (Windows endpoint for Chariot)
- endfbsdr.tar (FreeBSD endpoint for Chariot)

CD2

FreeBSD 3.4 bootable installation CD