HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Electrical and Communications Engineering

Jari Huttunen

Measurements on Differentiation of Internet Traffic

Master's thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology

Supervisor: Professor Raimo Kantola
Instructor: Marko Luoma, Lic.Sc.(Tech.)

Espoo, 13*th* January 2005

Author: Jari Huttunen

Title: Measurements on Differentiation of Internet Traffic

Date: 13*th* January 2005 **Number of pages:** 85

Department: Department of Electrical and Communications Engineering
Professorship: S-38 Networking Technology

Supervisor: Professor Raimo Kantola
Instructor: Marko Luoma, Lic.Sc.(Tech.)

The use of real-time applications in the packet networks is setting new requirements for the packet delivery. The current best-effort model in the Internet does not facilitate control over important characteristics such as delay, jitter and packet loss, which is needed for the proper operation of applications with real-time requirements.

The main focus during the last years on providing QoS in IP networks has been in the area of the Differentiated Services (DiffServ) architecture. DiffServ provides the necessary tools for implementing better service to the Internet in a scalable way. Scalability is an essential requirement for all Internet services.

DiffServ is a framework that describes the main components and mechanisms for realizing QoS. The framework gives quite a lot of freedom for how the implementation is done and what are the differentiation principles. This has created a lot of research and arguments about how the DiffServ network should be actually implemented.

This thesis studies the differentiation issues in the DiffServ network. Differentiation in this work is based on the idea of separating traffic with different characteristics (e.g. UDP and TCP) into distinct forwarding classes. The first part of this work presents the motivation for application based differentiation and describes the DiffServ architecture as well as some other ways of providing QoS in the Internet. Also two well known DiffServ implementations for general purpose PC hardware will be presented. The last part of this work presents measurements that we have conducted in an isolated DiffServ network using ALTQ as the QoS engine. ALTQ/CBQ is used to provide a class based isolation among different traffic types.

Keywords: Differentiated Services, Measurement, ALTQ

Tekijä: Jari Huttunen

Otsikko: Mittauksia Internet-liikenteen eriyttämisestä

Päiväys: 13. tammikuuta 2005 **Sivumäärä:** 85

Osasto: Sähkö- ja tietoliikennetekniikan osasto
Professuuri: S-38 Tietoverkkotekniikka

Valvoja: Professori Raimo Kantola
Ohjaaja: TkL Marko Luoma

Reaaliaikasovellusten käyttö pakettipohjaisessa tietoverkossa asettaa uusia vaatimuksia pakettien välitykselle. Nykyään käytössä oleva best-effort Internet-malli ei tarjoa mahdollisuutta kontrolloida viivettä, viiveen vaihtelua eikä pakettihukkaa, mikä tarvittaisiin reaaliaikapalvelujen kunnollisen toiminnan takaamiseksi.

Pääpaino viime vuosina palvelunlaadun tuomisella IP-verkkoihin on ollut Differentiated Services (DiffServ) arkkitehtuurin ympärillä. DiffServ tarjoaa tarvittavat työkalut, joiden avulla voidaan rakentaa parempaa palvelua Internetiin skaalautuvalla tavalla. Skaalautuvuus on välttämätön kriteeri toteutuksille, jotka liitetään osaksi Internet-arkkitehtuuria.

DiffServ on kehysrakenne, joka kuvailee pääkomponentit ja mekanismit palvelunlaadun toteuttamiseksi. DiffServ kuitenkin jättää varsin paljon vapautta ja tulkinnanvaraa kuinka itse toteutus tulisi tehdä ja mitkä ovat eriyttämisperiaatteet. Tämä on luonut paljon DiffServ-arkkitehtuuriin liittyvää tutkimusta ja sen myötä myös kiistelyä siitä, kuinka DiffServ-pohjainen tietoverkko tulisi itse asiassa toteuttaa.

Tämä opinnäytetyö tutkii eriyttämisperiaatteita DiffServ-verkossa. Eriyttäminen tässä työssä pohjautuu ajatukseen erottaa liikenne luonteensa perusteella (esim. UDP ja TCP) omiin liikenneluokkiinsa. Työn ensimmäinen osa esittää motivoinnin sovelluspohjaiseen eriyttämiseen sekä kuvailee DiffServ-arkkitehtuurin ja muita tapoja palvelunlaadun toteuttamiseen Internetissä. Myös kaksi yleisesti DiffServ-toteutusta PC-arkkitehtuurissa käydään läpi. Työn loppuosa esittää mittaukset, jotka on suoritettu eristetyssä ALTQ:n avulla toteutetussa DiffServ-verkossa. Mittauksissa käytettiin ALTQ:n CBQ-toteutusta saavuttamaan luokkapohjainen eriyttäminen eri liikennetyyppien välillä.

Avainsanat: Eriytetyt palvelut, Mittaus, ALTQ

Contents

Acronyms

ACK	Acknowledgement
AF	Assured Forwarding
ALTQ	Alternate Queueing
API	Application Program(ming) Interface
AQM	Active Queue Management
BA	Behavior Aggregate
BE	Best Effort
BSD	Berkeley Software Distribution
CBQ	Class Based Queueing
CBR	Constant Bit Rate
CPU	Central Processing Unit
CRUDE	Collector for Real-time UDP Data Emitter
DiffServ	Differentiated Services
DRR	Deficit Round Robin
DSCP	Differentiated Services Code Point
EF	Expedited Forwarding
EWMA	Exponentially Weighted Moving Average
FCFS	First Come First Served
FEC	Forwarding Equivalency Class
FIFO	First In First Out
FTP	File Transfer Protocol
GOP	Group of Pictures
HTTP	Hyper Text Transfer Protocol
IETF	Internet Engineering Task Force
IntServ	Integrated Services
IP	Internet Protocol
ITU-T	International Telecommunication Union-Telecommunication
LRS	Label Switched Router
LSP	Label Switched Path
MF	Multi Field

MPEG	Moving Picture Experts Group
MPLS	Multi Protocol Label Switching
NRT	Non Real Time
NTP	Network Time Protocol
P2P	Peer to Peer
PC	Personal Computer
PHB	Per Hop Behavior
PSQM	Perceptual Speech Quality Measure
QoS	Quality of Service
RAM	Random Access Memory
RED	Random Early Detection
RR	Round Robin
RSVP	Resource Reservation Protocol
RT	Real Time
RTT	Round Trip Time
RUDE	Real-time UDP Data Emitter
SSH	Secure Shell
TC	Traffic Control
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VBR	Variable Bit Rate
VoIP	Voice over IP
WRR	Weighted Round Robin
WWW	World Wide Web

Chapter 1

Introduction

1.1 Background

The traffic volume and the variety of applications in the Internet have evolved many changes since the early days. Originally, Internet was a rather small network for connecting academic institutions. The main applications used those days were file transfer and email. These applications do not have strict requirements on the delay or available bandwidth in the network and they can adapt quite well to changes in the network load. In addition, as the amount of users and traffic was low, the network was able to provide adequate quality of service (QoS) for the users by using a simple best effort scheme. Best effort means that the network does it best to deliver packets to the destination but it cannot give any guarantees on the service quality and the service is the same for all packets.

The situation today is different in many ways. Internet has become a communication media for all the people and the amount of traffic in the network has grown to a level where packet delays and drops are inevitable. Also many new applications have been born that require more than the original best effort Internet is capable of providing. The deployment of multimedia services such as voice over IP, video streaming and conferencing in the Internet will need control over packet loss and delay which is lacking in the best effort model. In addition to real-time applications also some interactive data applications exist that require better predictability in order to give good user experience. Such an application is www, the killer application from the 90's. Web browsing requires fast response times or otherwise the user gives up and aborts

the connection. The slowdown in web page download has already created the definition of "world wide wait".

To overcome these problems different QoS architectures have been developed. The most promising architecture appears to be differentiated services [BBC+98]. Differentiated services allows a scalable way of providing control on network resources in the Internet. Scalability is the key issue when adopting new services to the Internet.

1.2 Aims and scope of this work

The research challenge in this thesis was to create an isolated fully functioning DiffServ capable network with traffic sources and study differentiation principles and mechanisms through traffic measurements.

Many simulation and experimental studies have been done in the area of Internet traffic differentiation. However, most of these works include very simplified traffic models, topologies, and analyzing methods. To get a better understanding of the DiffServ mechanisms, following design principles were used when building the testbed:

1. Create a DiffServ capable network including several edge and core routers.
2. Emulate carefully different traffic types (Video, VoIP, WWW and FTP).
3. Create paths with dissimilar RTT.
4. Make analysis at user level and network level
5. Use dedicated measurement hardware for accurate one-way delay measurements.

The main goal of this thesis is to determine through measurements the appropriate principle on how traffic differentiation should be done, the level of differentiation that is needed and also study the performance and limitations in the implementation. This thesis does not handle the concepts of access policing or QoS routing.

1.3 The structure of the thesis

Chapter 2 begins with an introduction to the concept of Quality of Service and gives the motivation on why QoS should be realized. At the end of this chapter, different models for implementing QoS will be introduced.

Chapter 4 explains the concept of utility and the characteristics of different Internet applications. In addition, the interference between applications and different differentiation principles will be discussed.

Chapter 3 presents the key components in the DiffServ architecture.

Chapter 5 discusses the problems that arise when implementing DiffServ functionality into real world hardware. Also two most well known DiffServ implementations will be presented.

Chapter 6 describes the tools and methods that were used in the measurements and analysis.

Chapter 7 presents the measurement environment and the emulated applications.

Chapter 8 contains the measurement results and the analysis.

Chapter 9 concludes this work and gives summary and conclusions from the measurement results.

Chapter 2

Quality of service

2.1 What is quality of service?

The concept of quality of service (QoS) is hard to define with only some sentences. The term QoS is rather ambiguous and a lot of debate can be brought out what it really means. ITU-T recommendation E.800 [IT94], formally defines QoS as: "*The collective effect of service performance which determines the degree of satisfaction of a user of the service*". This definition presents QoS as an attempt to satisfy the user demands with a service that fulfills his/her needs. Fulfilling this may not be always easy as different users may have different expectations on quality: some users are more demanding than others.

The concept of QoS can be also thought from the service provider's point of view. In that case, QoS means that the service the provider is giving to its customers fulfills certain quality measures. Typical measures in the packet networks are packet loss, delay, delay variation, and bandwidth. These parameters can be measured and used to give an overview of the current QoS in the network.

2.2 Motivation and benefits of implementing QoS

When the Internet was born in the beginning of 80's, the concept of QoS was not that important. The main objective was to create a robust network that could deliver successfully data packets. This was adequate while traffic

volumes were low and applications did not have strict real-time requirements. However, many applications have born after those times that require controlled delay or/and bandwidth behavior from the network. The current best effort network is very often not capable of satisfying the requirements from these applications and therefore also not capable of satisfying the customers using those applications.

The benefits that QoS can provide can be thought from two perspectives: 1) user and 2) service provider. For the user QoS means the possibility to get better service from the network and the ability to use applications that have different requirements. For instance, telephony in Internet does usually not perform well enough due to the need for strict delay control that is lacking in the single service network. From the operator's point of view QoS is a possibility to make more revenue as it can offer a wider range of services for the user and give certain guarantees regarding the service quality. If QoS is implemented well it will also help to make a better use of the network resources and therefore utilize the existing network more efficiently.

2.3 QoS models for IP networks

We focus here on four well-known architectural models for implementing QoS in the IP networks:

1. Over-provisioned Best Effort network
2. Integrated Services (IntServ)
3. Multiprotocol Label Switching (MPLS)
4. Differentiated Services (DiffServ)

Over-provisioning

Over-provisioning is a way to provision the link bandwidths in a way that traffic load never or very rarely exceeds the link capacity. The idea of over-provisioning is simple and therefore it is widely used as a way to increase the QoS level in the networks. Over-provisioning, however, results usually in a low network utilization as the Internet traffic is bursty in nature. In addition, by

over-provisioning too much, the QoS levels hardly improve anymore[1] and it becomes very cost-inefficient way of providing QoS.

Integrated Services

The Integrated Services architecture [BCS94] is based on an idea of making resource allocations in order to meet the user and application requirements. The reservations are made on per-flow basis so that assured bandwidth and delay can be guaranteed to applications.

The main components in the IntServ model are shown in Figure 2.1 [Wan01]. The model can be separated into two logical parts: the *control plane* and the *data plane*. The control plane is responsible of setting up the reservations while the data plane forwards data packets based on the reservations state acquired from the control plane.

Resource reservations are handled with a specific signaling protocol, the Resource Reservation Protocol (RSVP)[BZB+97]. The reservation process begins with a flow specification where an application characterizes its traffic flow and QoS requirements. The reservation setup is sent to the router where it interacts with a routing module and the admission control. Routing module is used to determine the next hop for the reservation forwarding. Admission control checks whether there are enough resources to satisfy the request. When the reservation setup is ready, the information for the reserved flow is stored into the resource reservation table. The information in the resource reservation table is used in the data plane to configure packet scheduling and the flow identification module. The flow identification module filters packets belonging to flows with reservation and passes them to appropriate queueing disciplines. The packet scheduler shares the resources to the flows based on the reservation information. [Wan01]

Despite that IntServ was developed already about ten years ago it has been deployed only in some Intra-networks. The main reason for the failure of IntServ is the lack of scalability. The setting of state in all routers along a path is non-scalable and non-workable administratively as most of the Internet data flows are short in lifetime. In addition, in IntServ, autonomous system or provider boundaries are considered to be essentially invisible. IntServ expects that reservation state can be delivered across administrative boundaries without any problems. This would require Internet wide peering agreements.

[1] See Chapter 4 for details how performance of an application depends on the network parameters

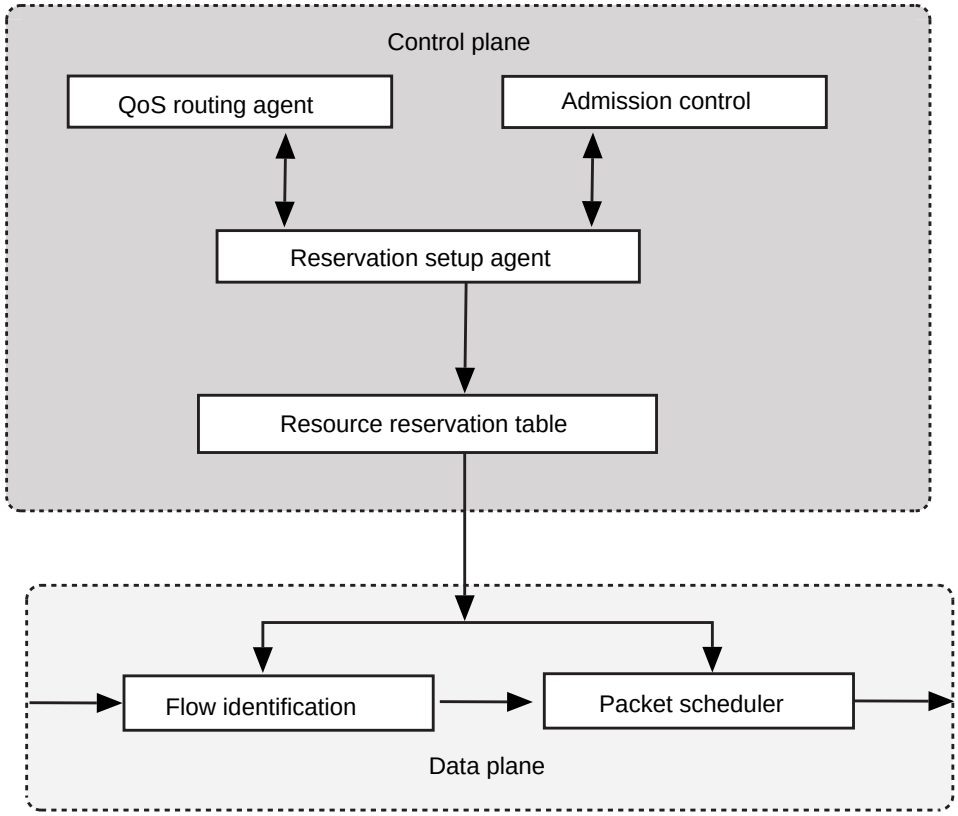


Figure 2.1: Main components in the IntServ architecture

Multiprotocol Label Switching

Multiprotocol Label Switching (MPLS) [RVC01] is an architecture developed by IETF for improving routing performance in IP networks. MPLS can be thought of as a way of combining the flexibility of network layer routing and the efficiency of link layer switching. MPLS also brings connection-oriented properties for IP networks in a scalable way.

Conventional routing in IP networks is based on an idea that every router in the path examines the packet header and makes an independent decision about the next hop. The main idea in MPLS is to simplify this process so that routers do not need to perform an address lookup for every packet. The next hop is determined in MPLS by using a fixed length label in the packet header. [RVC01]

As an IP packet arrives to an MPLS domain, the header is examined and it is assigned a small label. This label is used to set the Forwarding Equivalency Class (FEC) for the packet. The FEC is a group of IP packets, which are forwarded in the same manner. The MPLS capable label-switched routers (LSRs) use the label to determine the next hop and the corresponding new

label. When the existing label is changed to a new one the packet can be sent to the next hop. The path along which MPLS packet traverses is called a label-switched path (LSP). Since the mapping between labels is fixed at each LSR, the LSP is actually determined by the initial label value at the ingress LSR. When the packet arrives at the egress point of the MPLS domain, the MPLS header will be removed. The basic operation of an MPLS network is illustrated in Figure 2.2. [Wan01]

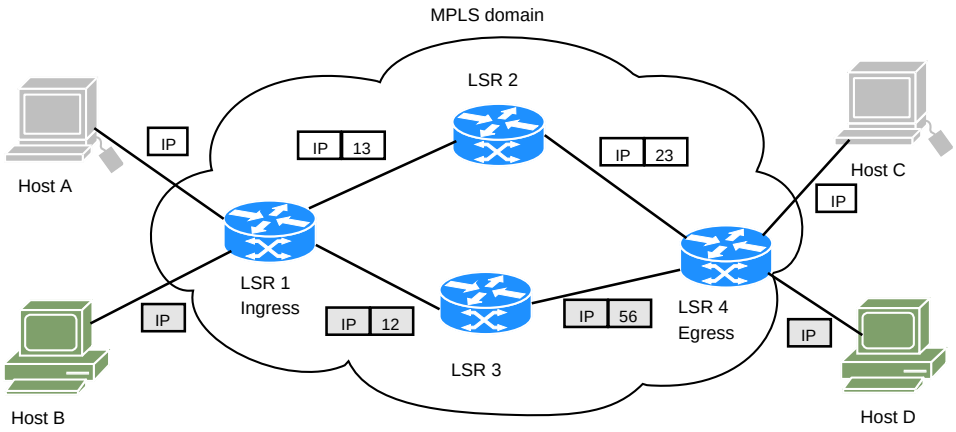


Figure 2.2: Basic operation of an MPLS network

Differentiated Services

As IntServ was found to be a non-scalable solution for implementing QoS in the IP networks, IETF designed a new more scalable QoS service model called Differentiated Services (DiffServ) [BBC+98]. DiffServ gives up the idea of providing end-to-end guaranteed service and focuses on providing better than best effort service. The DiffServ architectural model is explained in more depth in Chapter 3

Chapter 3

Differentiated Services

Differentiated Services (DiffServ) [BBC+98] is an architecture based on the idea of grouping traffic flows into a finite number of traffic classes. As DiffServ is based on aggregates, it offers a scalable way of providing QoS, which was lacking in the IntServ architecture that uses state-based reservations of resources for individual traffic flows.

DiffServ network has two types of routers: edge routers and core routers (Figure 3.1).

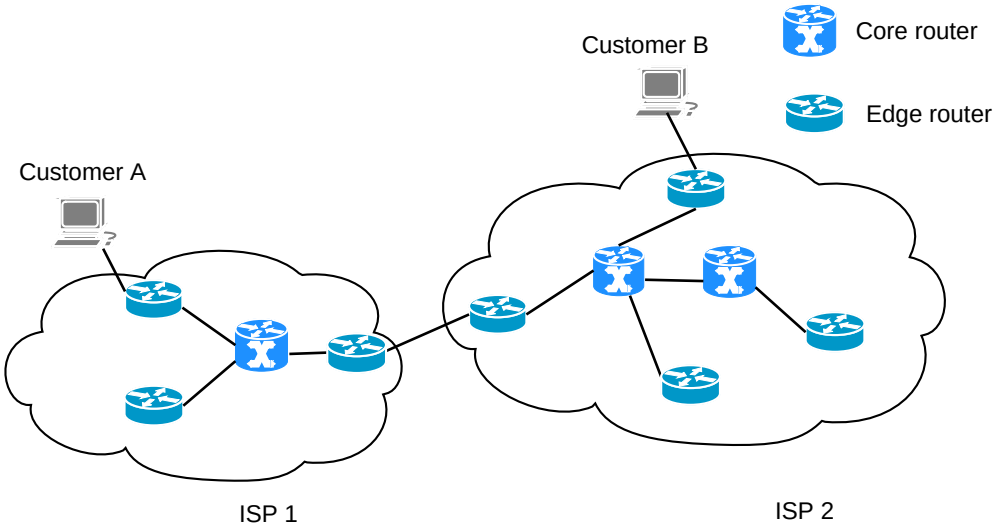


Figure 3.1: Differentiated service network consisting of edge and core routers

The complexity in the DiffServ network is located in the edge routers where traffic volume is relatively low. The low data volume makes it possibly to make packet processing with rather low overhead. The packet processing at the edge routers includes classification, policing, and conditioning actions. For

every packet, a DSCP [1] value is set that defines the forwarding behavior for the packet in the core network.

Core routers are simple compared to edge routers and their job is to do forwarding based on the DSCP coding. In other words, the queueing behavior is determined by the value in the DSCP field. This queueing behavior is called per hop behavior (PHB).

3.1 Per Hop Behavior

A Per Hop Behavior (PHB) [BCF00] describes the treatment for traffic belonging to a certain behavior aggregate at an individual network node. The differentiation between different PHB's is obtained by looking at the DSCP field in the IP packet's header. Many PHB's have been proposed and some of them have been also standardized. We focus here on two commonly used standardized PHB's: Assured Forwarding and Expedited Forwarding.

3.1.1 Assured forwarding

Assured Forwarding (AF) PHB [HBWW99] defines four independent forwarding classes for IP packet delivery. Within each AF class one of the three different drop precedences can be assigned to IP packets (Figure 3.2). The drop precedence of a packet can be used to determine the importance of a packet within the AF class. In case of congestion, packets with low drop precedence are favored and packets with high drop precedence are more likely to be discarded.

The forwarding assurance of a packet depends on three factors: [HBWW99]

1. Provisioning i.e. the resources allocated to the AF class where the packets belong to.
2. Current load of the AF class.
3. Drop precedence of a packet (in a case of congestion).

[1] A Differentiated Service Code Point (DSCP) is encoded in the most significant 6 bits of the ToS byte contained in the IP header

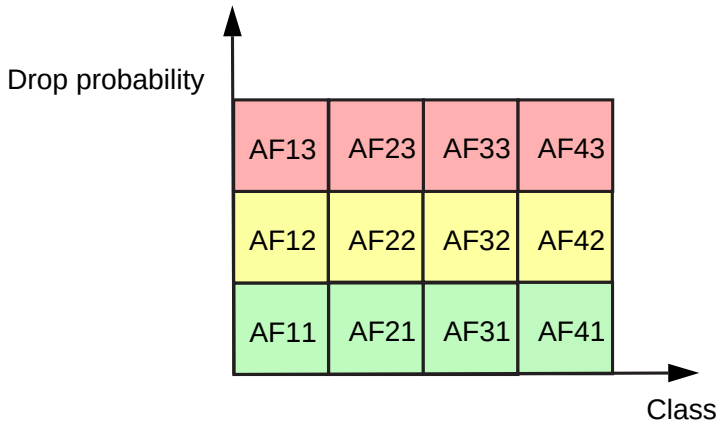


Figure 3.2: AF classes with different drop precedences [Luo03]

3.1.2 Expedited Forwarding

Expedited Forwarding (EF) PHF [JNP99] aims to provide a low delay, low jitter, low loss and assured bandwidth service and is therefore intended to be used with applications like Voice over IP and video conferencing. To be able to provide low delay and low jitter service the packets belonging to the EF group need to encounter empty or very short queues. Ensuring short queues means that the arrival rate of EF packets must not exceed the service rate at the interface.

3.2 DiffServ building blocks

DiffServ model includes two conceptual elements in the ingress point of the network: classification and conditioning. Conditioning includes numerous functional elements that are used to implement conditioning actions. (Figure 3.3)

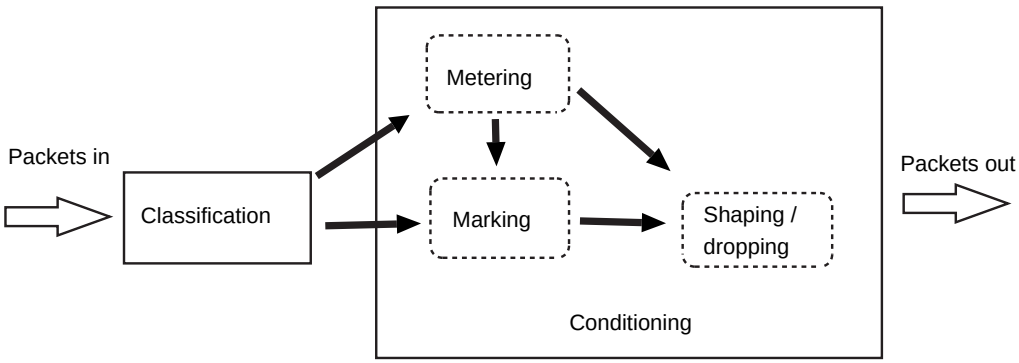


Figure 3.3: Traffic conditioning components

3.2.1 Classification

Packet classification is a process where packets are identified and separated for further processing based on the information in the packet header. There are two kinds of classifiers used for traffic classification: Behavior Aggregate (BA) classifier and Multi-Field classifier (MF). BA uses only the DSCP field for classification whereas MF uses a combination of fields of the IP header (e.g. source address and source port). MF is usually used at the edges of the network for packet classification and BA in the core of the network due to its simplicity.

3.2.2 Conditioning

Conditioning is an important element in the DiffServ model that may contain the following functional components [BBC+98]: metering, marking, shaping and dropping. Conditioning is used to ensure that on average each behavior aggregate will get the agreed service.

Metering is a process to determine whether the behavior of a packet stream is within the specified profile. The output result of metering is used to trigger events in other conditioning blocks. There are many estimators that can be used for implementing metering. The most known and widely used estimator in the packet networks is the Token Bucket estimator.

Token Bucket is a relatively simple algorithm that can be described by two parameters: Token generation rate (R) and size of the token bucket (S) (Figure 3.4). The tokens arrive at the bucket with rate R. The size of the bucket describes the maximum burst size that can be sent to the link. Overflowed tokes can not be stored, they are simply discarded. To send a packet size of B the corresponding amount of tokens are reduced from the bucket. Each token represents some number of bytes and the packet can be sent if enough tokens exist in the bucket. If there are not enough tokens in the bucket, the packet is either shaped or simply dropped.

Marking is a process where packets are marked to belong to a certain service class. Marking is usually done at the edges of the network where some predefined DSCP value is set to the packet header.

Shaping is a process where packets are delayed in order to get the traffic stream fit to a predefined profile. Dropping has similar objectives as shaping but it

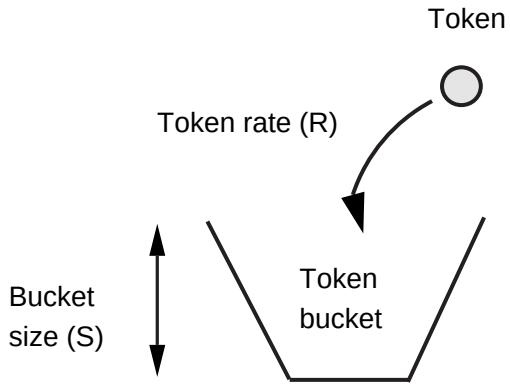


Figure 3.4: Token bucket estimator

drops packets in order to get the traffic stream into compliance with the profile.

3.3 Active queue management

Queues are essential in the routers as they smooth bursty traffic in order to avoid packet loss. Queue management defines the policy which packets are dropped in the case of congestion. The simplest, and still widely used, dropping policy is tail-drop (a.k.a drop-tail). Tail-drop simply drops incoming packets when the buffer fills up. Unfortunately, when dealing with persistent congestion tail-drop performs badly and may lead to higher delays, bursty packet drops, bandwidth unfairness and to a global oscillation of traffic sources. To face these problems a number of active queue management (AQM) algorithms have been proposed. Active queue management is a pro-active approach of informing the sender about the congestion before the buffers overflow.

3.3.1 Random early detection

Random Early Detection (RED) [BCC+98] is the most studied active queue management algorithm in the Internet. RED was developed to provide better fairness, maximize the link utilization and to avoid global synchronization. [FJ93]

RED uses the average queue size as the indication of emerging congestion. The average queue size is calculated using the exponentially weighted moving average (EWMA) algorithm. The use of EWMA makes it possibly to distinguish between short time bursts and long time congestion. The RED mechanism

begins to drop packets with increasing probability *p_drop* when the minimum threshold value *th_min* is exceeded. The dropping probability will increase with a linear behavior from zero to a maximum value *p_max* until the upper threshold value *th_max* is reached. After this point RED falls to a simple tail-drop algorithm by dropping all the packets (Figure 3.5).

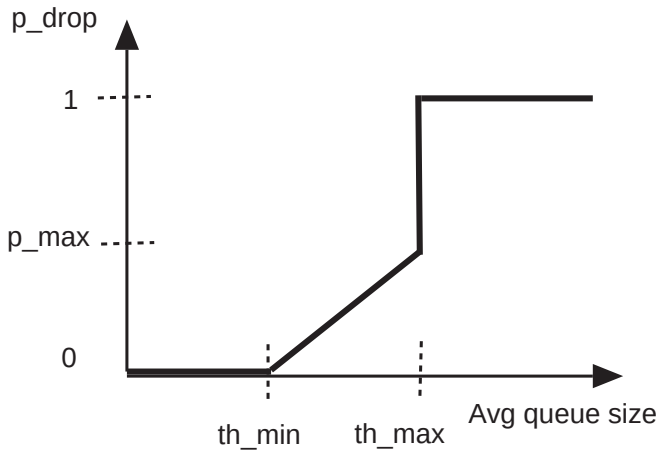


Figure 3.5: RED dropping function behavior

3.4 Scheduling

Scheduling is an event that decides the order of packets to be served from different queues. Scheduling algorithms can be categorized in many ways. A typical way is to divide scheduling schemes to work conserving and non-work conserving.

A work conserving scheduler is never idle if there are packets to be served. This means that a work conservative scheduler can effectively utilize the network resources. A non-work conserving scheduler can be idle even though there are packets to be served. Therefore, a non-work conserving scheduler usually utilizes network resources worse than a work conserving scheduler [LY99]. Non-work conserving schedulers may also yield to higher end-to-end delays. However, non-work conserving schedulers have some properties that make them suitable for QoS networks. The most important advantages are 1) control over delay jitter[2] and 2) rate control. Controlled delay jitter is important for certain applications with hard real-time requirements (e.g. Voice over IP). Rate control enforces a traffic stream to conform to its profile before forwarding it

[2]Delay jitter is defined as difference between the largest and the smallest delay

to the scheduler. Thus, the use of rate control usually makes the scheduler non-work conserving.

3.4.1 First Come First Served

First Come First Served (FCFS) is the dominant scheduling algorithm used in the Internet serving best effort service. In FCFS packets are served based on the order they arrive; first packet in is first served out. The advantage of FCFS is its simplicity and therefore it can be easily implemented. The drawback of FCFS scheduling is that it can not really provide any delay or throughput guarantees and is therefore not well suited for networks providing QoS.

3.4.2 Round Robin scheduling

Round Robin (RR) is an old, rather simple, and widely used scheduling algorithm designed especially for time-sharing systems. In RR, a small time slice is allocated for each process. These processes are kept in a circular queue and served until the task is accomplished. RR can be used in routers to give each class a service time of one time slice during each round. The best solution in terms of fairness would be a bit-by-bit Round Robin scheduling where one bit of each flow is served per round. This is, however, not practical and can not be implemented efficiently. A simple RR implementation is packet-by-packet Round Robin that schedules one packet from each class per round. If the time slice is not equal between the classes, the scheduling algorithm is called Weighted Round Robin (WRR). WRR works well if the packet size is fixed or the average packet size for a flow is known. Otherwise WRR is not able to allocate bandwidth fairly [Wan01]. Deficit Round Robin (DRR) [SV95] is similar to WRR but it takes into account variable packet sizes by using a deficit counter. The deficit counter is used to keep track of the unused time slices for the class from the previous rounds. The unused time slices can be added at the next round to the current time slice. If the class is idle, the unused time slices will be discarded as the class has wasted its opportunity to send packets.

3.4.3 Class Based Queueing

Class Based Queueing is a method of hierarchical link-sharing proposed by Sally Floyd [FJ95]. CBQ can be used to share the bandwidth among dif-

ferent agencies, protocols, and traffic types on a link in a controlled fashion. The network resources are divided into a tree-like class structure. *Root class* contains all the resources on the link that can be shared among intermediate classes. *Intermediate classes* form logical groupings that specify how resources are divided to the leaf classes. *Leaf classes* are the actual traffic classes and therefore all packets sent out of the link must first be queued in one of the leaf classes (Figure 3.6).

The CBQ scheduling mechanism can be divided into two operational parts: the general scheduler and the link sharing scheduler. The general scheduler is used when none of the leaf classes have exceeded their assigned resources. The link sharing scheduler is activated when some of the leaf classes use more than the assigned amount of resources. When a class is sending more than its share of the bandwidth, it is said to be *overlimit* and it must be *regulated*[3]. If a class sends less than the assigned bandwidth it is said to be *underlimit*. CBQ includes a borrowing mechanism that allows traffic classes to utilize the unused bandwidth by other classes. This means that in some cases a class can send even in an overlimit situation.

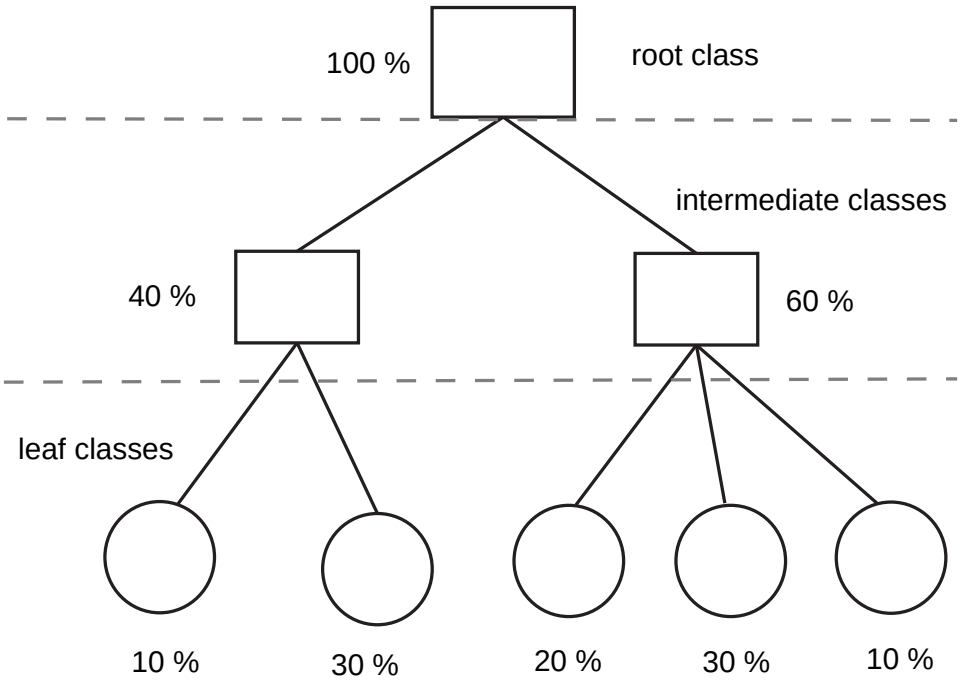


Figure 3.6: A CBQ resource sharing structure example

[3] A class is said to be *regulated* if packets are scheduled using link sharing scheduler

3.4.4 CBQ implementation in ALTQ

ALTQ [Cho99] includes an implementation of CBQ that has all the mechanisms introduced by Sally Floyd [FJ95]. ALTQ/CBQ is a testbed that has been widely used in the networking science for prototyping new queueing disciplines.

The ALTQ/CBQ includes the implementation of Weighted Round Robin and Packet Round Robin to be used as the general scheduler. A modified top-level link sharing algorithm is used for the link-sharing scheduling. It should be noted that even though ALTQ is announced to use WRR as the general scheduler it actually employs DRR in its source code.

By default, ALTQ/CBQ provides fixed provisioning of resources to each class and thus the excess bandwidth can not be distributed among other classes. To be able to exploit the excess resources ALTQ/CBQ introduces an option *borrow*. With the borrow option a class is allowed to borrow the excess bandwidth from its parent. A class can borrow if its parent is underlimit or if it has an underlimit ancestor. This can lead to a non-work conserving behavior in some cases. Non-work conserving service can be avoided by using the *efficient* option that enables a class in an overlimit situation to send a packet even if all its ancestors at that moment are also overlimit. [RG99]

Chapter 4

Differentiation of Internet traffic

4.1 Application characteristics and utility

There are many ways to divide applications into different groups. One way is to make a distinction between elastic and real-time applications [She95]. For each group qualitative utility functions can be shown that present properties of applications on a qualitative level (Figure 4.2, 4.3 and 4.4). Utility functions describe how the performance of an application depends on the network parameters (e.g. delay and bandwidth).

Another way of characterizing the differences between applications is to divide them to delay sensitive and bandwidth sensitive (Figure 4.1). There are, however, applications that are both delay and bandwidth sensitive. An example of such an application is interactive video conferencing that requires at the same time low delay and high bandwidth in order to work properly. VoIP is an application that is very delay sensitive and requires a guaranteed bandwidth without packet loss[1]

4.1.1 Elastic applications

Examples of elastic applications are TCP based protocols for file transfer like P2P and FTP. For elastic applications, it is typical that the increase in network resources enhances the performance of an application but in a rather conservative way. The elastic applications benefit from increased bandwidth

[1] See Section 4.1.2.2 for more details about voice traffic.

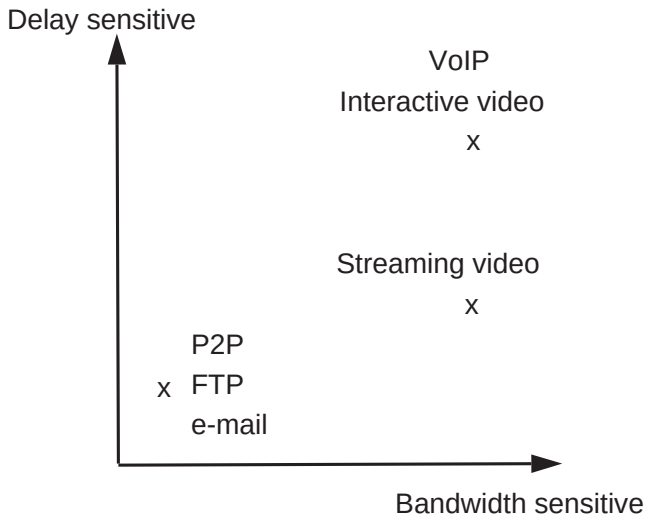


Figure 4.1: Delay and bandwidth sensitive applications

but they are able to operate with only a minimal amount of network resources. This behavior is illustrated in Figure 4.2 that presents the utility function for elastic applications.

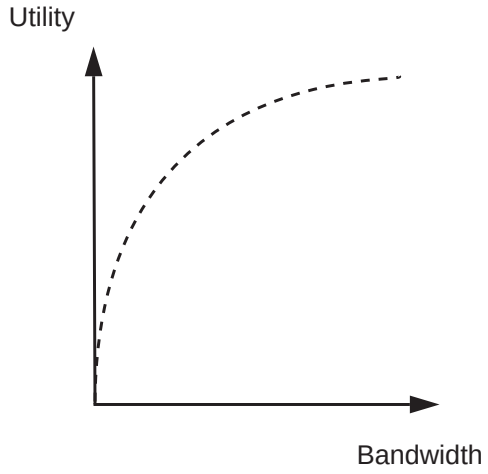


Figure 4.2: Utility function for elastic applications

4.1.1.1 The battle between mice and elephants

It is a well-known fact that the Internet traffic is heavy tailed [PF95]. This means that most of the Internet traffic is carried by a small number of long lasting connections (*elephants*) while a large portion of the connections are short in lifetime (*mice*). Typical elephants are P2P applications and FTP file transfers. They are more robust to changes in the network resources: a

user does not usually mind if downloading a large file takes some seconds or even some minutes longer. However, the applications using short connections are usually interactive in nature meaning that they need their data within a certain time limit. A typical application with interactive requirements is the Word Wide Web (WWW). A user expects a page to load within some seconds after which the user resets the connection as there seems to be no reply[2].

In a congested network, the performance of a short connection can collapse and a user browsing the Internet may have to wait a long time downloading even a small text file. The reasons for collapse in the performance can be found from the operation of TCP, the transport protocol used in WWW. The three major factors in TCP that may affect on the performance of short connections are: [GM01]

1. In the beginning of a transfer the sending window is initiated conservatively to the minimum possibly value regardless of the available network resources.

2. For short connections the expiration of a retransmission timer is usually the only way to detect packet loss because the duplicate ACK mechanism requires some time (several packets) to activate. The use of a duplicate ACK for packet loss detection is usually much more efficient than using a timeout timer.

3. TCP uses the RTT estimate to calculate the retransmission timeout. As for the first control and data packets, no sampling data is available, TCP has to use an initial timeout value that has usually very high value (several seconds). Losing these packets can cause a long timeout during which TCP is unable to send any data.

There are also measurements [TMW97][FRC98] that reveal that short connections are bursty in nature which may affect the performance of long lasting connections due to occasional packet drops. Due to these facts, we can conclude that short and long TCP connections are interfering with each other and they should be separated.

[2]The user patience threshold is measured to be 15 seconds for WWW downloads after which the user hits the stop/reload button in the browser [Peu02]

4.1.2 Real-time applications

Real-time applications can be categorized into two main groups: hard real-time and soft real-time. Applications with hard real-time requirements are those that need their data within a strict delay bound. Data arrived late has no value and it can be simply discarded. Typical applications with hard real-time requirements are applications with conversational properties. Such applications are for instance telephony and video confererencing. The utility function for applications with hard real-time requirements is shown in Figure 4.3. As long as bandwidth requirements are met, the application performance is constant. However, when the available bandwidth drops below the operational point the queues fill up and the performance falls straight to zero.

Transferring data from a hard real-time application over a network has very strict requirements on one-way delay in order to maintain the conversation bi-directional[3]. In addition the variation in delay (jitter) should be within limits. A small amount of jitter can be compensated by using a playback buffer whose purpose is to absorb variations in delay and provide a smooth play out. The disadvantage of the playback buffer is that it increases the total end-to-end delay.

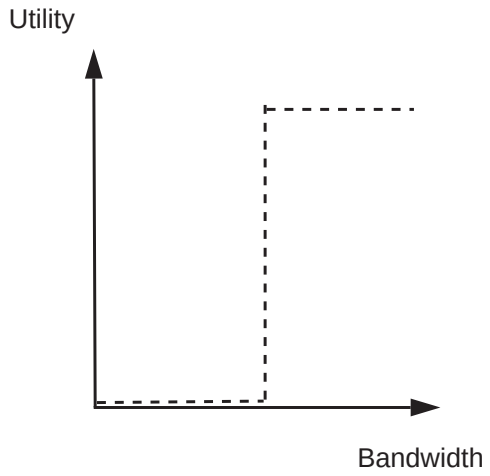
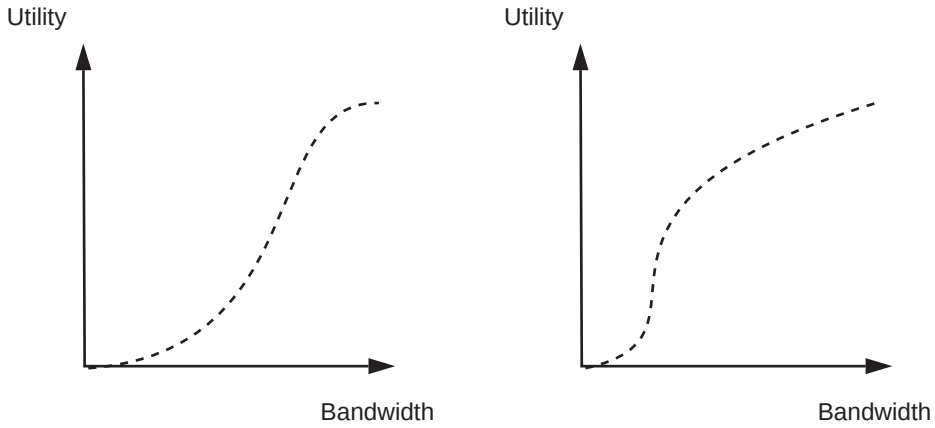


Figure 4.3: Utility function for applications with hard real-time requirements

Applications with soft real-time requirements are more robust to changes in delay and bandwidth as hard real-time applications. Typical soft real-time applications are streaming media applications that do not require two-way communication. With streaming media, the playback buffer can be large com-

[3] One-way delay should be less than about 300 ms in order to maintain a bi-directional conversation [LPY98]

pared to video conferencing, as there is no need for bi-directional communication. This relaxes the strict delay requirements as packets can be delayed quite a lot without any significant meaning to the operation of the application. Soft real-time applications can be further divided into delay-adaptive and rate-adaptive. Delay adaptive applications are rather tolerant for occasional delay variations and packet drops. Rate-adaptive applications are able to adjust their transmission rate in the case of network congestion. The utility curves for soft real-time applications in Figure 4.4 show that the drop in performance is not that sharp as with hard real-time applications (Figure 4.3).



(a) Delay-adaptive applications (b) Rate-adaptive applications

Figure 4.4: Utility functions for applications with soft real-time requirements

4.1.2.1 Video

Video traffic has an operation area ranging from some kbps to several Mbps. The large scale in transmit rate is due to several different compression and encoding schemes. In addition, the content of the video material has sometimes a large influence on the required transfer rate. More complex scenes and motion in the picture requires more data in order to obtain a certain level of quality.

The encoding scheme has a big influence on the characteristics of the video stream. The encoding schemes can be divided into two categories: 1) constant bit rate (CBR) and 2) variable bit rate (VBR). CBR video maintains the transfer rate during the transmission on the same level varying only little over time. On the other hand, VBR video may have peak values that differ many Mbps from the average rate. In that sense CBR video encoding is more predictable and eases the network resource provisioning needed for the video transfer. Despite that, VBR is a more commonly used encoding scheme in

video transfer than CBR. This is because with VBR encoding one can achieve better quality with the same available bandwidth as with CBR. Unfortunately, VBR encoding scheme has, due to its bursty nature, usually a low utilization degree if the data rate is high.

Many different compression schemes have been developed for different purposes. H.261 and H.263 are examples of compression algorithms that have been developed for low bit rate video transfer like video conferencing. MPEG-4 is an example of compression algorithm that enables coding of video and audio material at high quality. MPEG-4 results usually in a very variable bit rate stream as can be seen in Figure 4.5 that presents the bandwidth usage of a movie compressed with MPEG-4 using high quality.

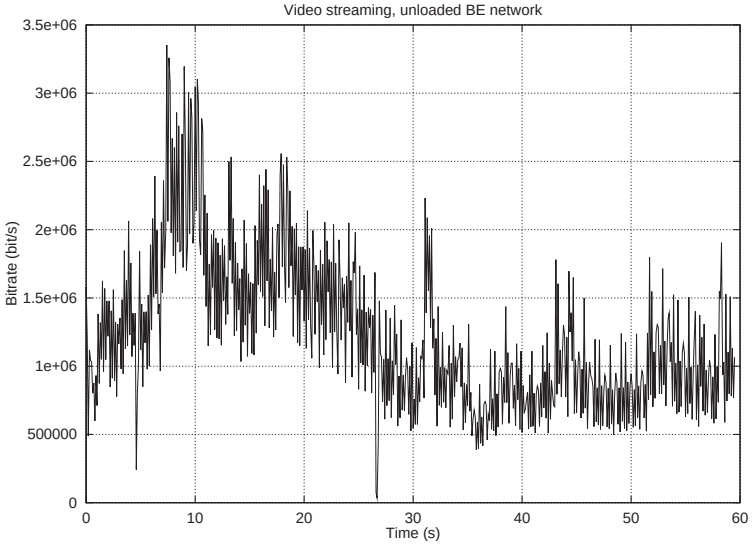


Figure 4.5: An example of bandwidth usage of a MPEG-4 compressed video

4.1.2.2 Voice

Voice is a typical example of an application with hard real-time requirements. The voice terminals generate a relatively low bit rate stream with fixed size packets (usually quite small) depending on the used codec. For instance, a G-723.1 codec can generate as low as 5.3 kbps stream with decent quality. Regardless the used codec, the traffic stream is typically at most only some tens of kbps. The traffic generated from voice terminals can be reduced even more by using silence detection[4] at the end terminals. This, however, increases the variability and therefore lowers the predictability of the voice stream.

[4]Silence detection is a method for identifying and removing the silent parts from the audio signal when the user does not speak.

4.2 Mixing the traffic

It is natural that mixing the applications (elastic and real-time) discussed before may have significant interference issues due to very different traffic characteristics and requirements. This interference makes it difficult to guarantee the operation of various traffic types in a single network. The most vulnerable for interference are real-time applications with hard requirements, as they need some amount of resources to perform and can not adjust to network congestion as elastic applications. However, there is also interference even between short and long TCP flows as was explained in section 4.1.1.1.

The interference between different application types have been studied in [Luo00][TNC+01]. These studies show through simulations and experiments that the different traffic types are interfering each other and that they should be separated.

Chapter 5

Implementing Differentiated Services

5.1 Gap between theory and reality

A real system has various limitations and complexities introduced by the software and hardware. Especially when implementing mechanisms in a general-purpose hardware a number of limitations arise. For instance a standard PC is designed to be a robust system for various different types of tasks. An extra effort is usually devoted to error handling and exceptional processing that lowers the performance. Also backward compatibility is important in a multipurpose environment and a lot of historical legacy from the long incremental evolution can be found in the software and hardware. Original design principles in the hardware and the operating system can set limitations that are, if not impossible, at least very difficult to overcome. Quite often these important issues are overlooked by the research people when designing a new theoretical model to be used in the real systems. [Cho04]

5.2 Alternate Queueing

ALTQ [Cho99] is a framework for the BSD systems to support a wide range of QoS components. ALTQ started as a challenge to implement theoretical QoS mechanisms into an open source operating system running on a standard PC hardware. ALTQ was originally designed to be a common research platform for traffic managing for the research community. Through the years, it has evolved

into a traffic managing subsystem that has been used by various people and organizations also for daily operational use. [Cho04]

5.2.1 ALTQ Design

The ALTQ framework can be divided into three major components:

1. Interface to the operating system.
2. QoS components.
3. Management tools in the user space.

The interface to the operating system component includes the functionalities to interact with the hardware and to actually make use of the QoS mechanisms. The QoS components are the building blocks that actually provide QoS. These include different queueing disciplines, traffic conditioning, scheduling etc. The management tools are located in the user space to offer an interface for the user or other management softwares. Through this interface, ALTQ can be configured and the status of a queueing discipline observed. The system architecture of an ALTQ traffic control model is presented in Figure 5.1.

5.2.2 Implementation issues

In this section, some mismatches between theoretical models and the real implementations are discussed in the ALTQ case.

5.2.2.1 Queue operations

In theory, a queue model needs only two simple operations, namely enqueue and dequeue. A packet is enqueued when it is stored to the queue. The packet is dequeued when removed from the queue to be sent out to the output interface. In real implementations, these two operations usually are not enough and additional operations are needed in order to cope with various situations.

ALTQ introduces a set of new queue operations to support different types of queueing disciplines. These new queue operations had to be implemented in the network drivers, as the existing queue model was not flexible enough for other

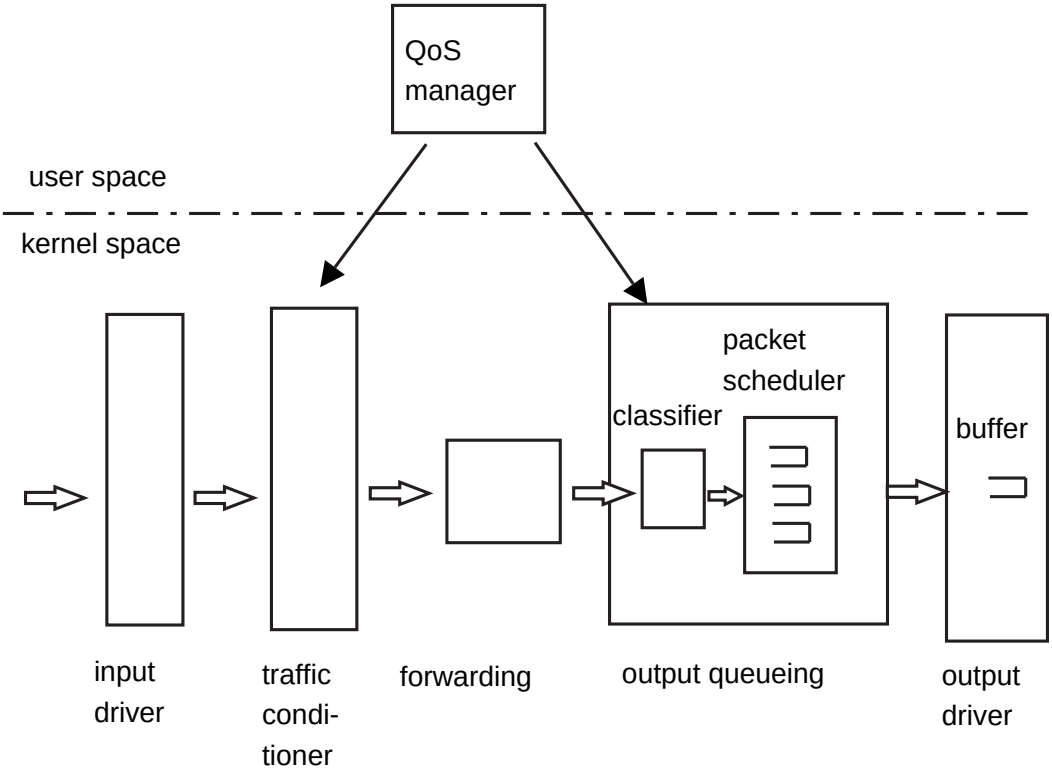


Figure 5.1: ALTQ system architecture [Cho04]

than FIFO queues. An important new queue operation is the poll operation that can be used to peek at the next packet to be dequeued. Luckily, for most of the drivers, the modifications for queue operations are straightforward macro replacements but the amount of supported drivers due to modifications is limited. [Cho01] [Cho04]

5.2.2.2 Output buffer model

A typical theoretical output buffer model includes only a single queue for each output link. This is, however, not true in a real PC system that includes two distinct output buffers located in the software and hardware. The software output buffer is realized by the operating system, which we can manipulate quite easily. However, the other buffer located in the network card is harder to control and it can have a significant impact to the performance such as increased delay and bursty dequeues.

Most network cards have a large buffer to maximize link utilization. This makes e.g. a delay constrained scheduler useless as packets can be delayed a long time in the device buffer before transmitting them to the wire. Thus, no

guarantees on delay can be given even to the high priority packets. When the device buffer is large, the packets are dequeued from the queue to the device buffer in a bursty manner as a large number of packets are dequeued at the same time. This reduces also control of the scheduler over the order of the packets. [Cho04]

Usually minimizing the number of interrupts in the network card is recommended as it lowers the CPU load. However, when using a queueing scheme that needs a fine grained operation, frequent interrupts are needed. Frequent interrupts enable the packet scheduler to be work conserving that is essential when maximizing link utilization.

To overcome the problems caused by the imperfect output buffer model ALTQ introduces a component called a token bucket regulator between the network card and the queueing discipline (Figure 5.2). The idea of the token bucket regulator is to regulate the amount of packets to be buffered in the device buffer. The amount of dequeued packets can be controlled by the tokens. Tokens arrive to the token bucket at the average token rate. A packet can be dequeued as long as there are enough tokens in the token bucket. For every dequeue operation tokens in the bucket are subtracted by the size of the packet. The size of the token bucket can be used to control the size of the bursts i.e. the number of packets to be dequeued at the same time. The token bucket size is automatically set by the ALTQ based on the interface bandwidth but the user can also configure it manually.

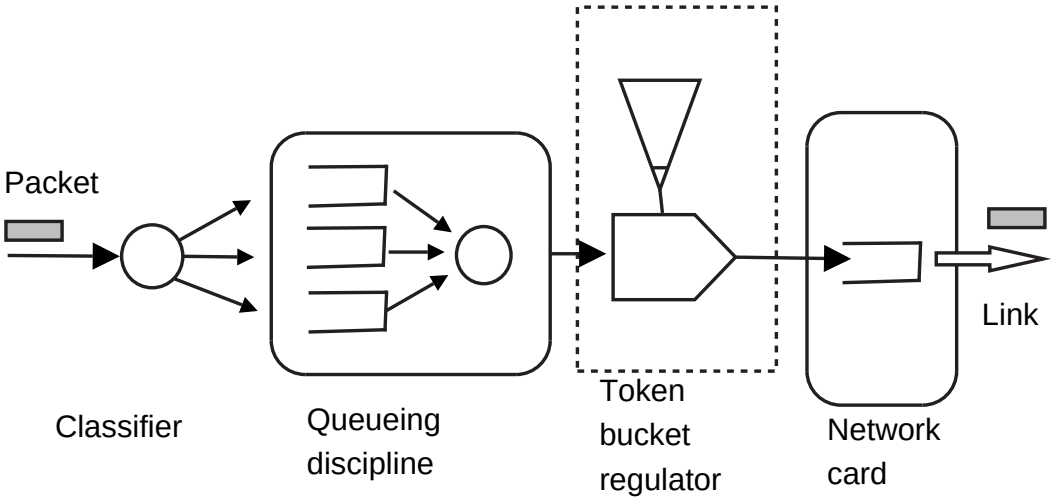


Figure 5.2: ALTQ output queue model [Cho04]

5.2.2.3 Effect of kernel timer resolution

The kernel operates in the rhythm of an internal clock that interrupts at regular intervals called ticks. A typical value for the kernel clock is 100 or 1000 Hz and it can be configured in FreeBSD using the HZ variable located in the kernel configuration file. The precision of the clock affects on some operations in the traffic management. For instance, when performing traffic shaping, the granularity of the kernel usually plays an important role. It is clear that traffic shapers need to be able to transmit waiting data packets at precisely calculated points of time in order to perform in a satisfying manner. If we consider a kernel with a 100 Hz timer, the accuracy of traffic shaping would be 10 ms. On a 100 Mbps Ethernet this means that we are able to send 125 packets of size 1000 bytes during the timer interval. This is too coarse and accurate traffic shaping in many cases is not possible. Therefore, the timer interval in kernel should be less than the packet transmission time.

The kernel timer plays a crucial role in the ALTQ/CBQ implementation. The internal clock frequency defines the minimum time between interrupts and therefore affects the minimum suspension time. The suspension time is the time during which the class is not able to borrow. The suspension time is calculated in clock ticks (Figure 5.3). Therefore, for a 100 Hz clock it means a minimum suspension time of 10 ms, which is usually too coarse. The higher the frequency of the internal clock is, the smaller the minimum suspension time can be. High clock rate can cause too much system overhead and therefore the value should not be set too high. The optimum kernel clock frequency is related to the clock rate of the processor; the higher the CPU clock rate the higher the kernel clock can be set.

5.3 Linux Traffic Control

Support for Differentiated Services on Linux is part of the more general Traffic Control (TC) architecture. TC includes a set of mechanisms and queueing operations to control incoming and outgoing traffic on a router. Linux TC includes basic building blocks that can be used to execute traditional traffic control actions (defined in Section 3.2) in the DiffServ architecture.

The traffic control in the Linux kernel can be divided into the following major components: [Alm99]

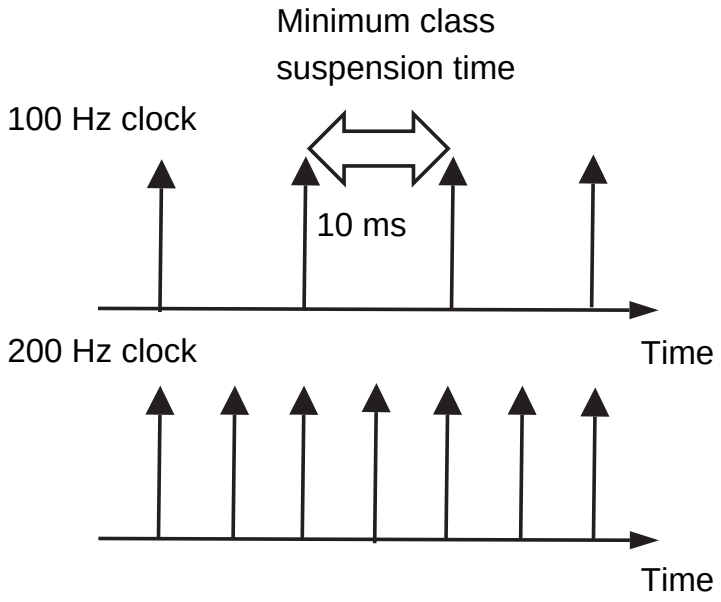


Figure 5.3: The effect of kernel timer on a suspension time of a class [RG99]

- Queueing disciplines
- Classes
- Filters
- Policing

Queueing discipline is an essential element in the Linux traffic control that controls how packets are enqueued on the device including scheduling and queueing operations. Queueing disciplines can be nested as shown in Figure 5.4. Filters can be used to differentiate packets into different classes.

The individual traffic control components are manipulated from the user space by using a *tc* configuration utility. The location and the role of each component need to be expressed in every configuration command leading to redundancy and hard to read configuration files. In other words, tc lacks the functionality of presenting the structure of traffic control components in an intuitive way. Therefore configuring traffic control through tc can be considered rather hairy leading easily to misconfigurations. An example of tc configuration is shown in Figure 5.5 where two FIFO queues are created with priority queueing. As can be seen already this simple configuration is rather hard to read. The issue of complicated configuration has lead to a project called Linux traffic control next generation (tcng) that tries to solve this problem.

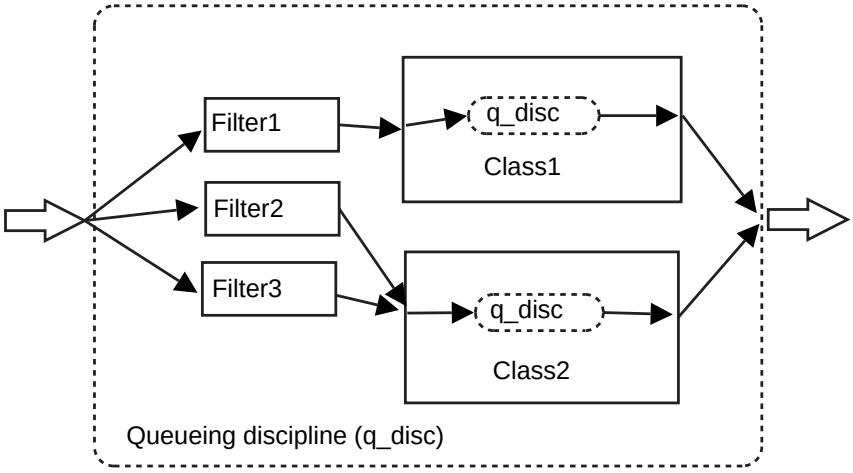


Figure 5.4: Nested queueing disciplines with two classes

5.3.1 Linux Traffic Control - Next Generation

Linux traffic control next generation is a project that aims to extend the existing traffic control in Linux to a compact and more user-friendly form. To achieve this tcng adds another abstraction layer between the tc configuration language and the user/application as shown in Figure 5.6. [Alm02]

The traffic control compiler (tcc) is used to translate configuration scripts written in the new tcng language to a common internal representation. From this representation, the commands in the tc language are generated. This kind of layering has benefits, as it does not require any changes to the original kernel or tc utility. [Alm02]

5.3.1.1 The tcng language

The tcng language has a C-like syntax that enables a more structured and easy to understand way to create configuration scripts. The configuration begins with the interface name following with the role (ingress or egress) and the entire classification. An example of a tcng script is shown in Figure 5.7 which will be translated to a tc form presented in Figure 5.5. Compared to tc syntax tcng offers a compact and easy to understand way of creating configurations.

```
tc qdisc add dev eth0 handle 1:0 root dsmark indices 4 \
 default_index 0

tc qdisc add dev eth0 handle 2:0 parent 1:0 prio

tc qdisc add dev eth0 handle 3:0 parent 2:1 bfifo limit 20480

tc qdisc add dev eth0 handle 4:0 parent 2:2 bfifo limit 102400

tc filter add dev eth0 parent 2:0 protocol all prio 1 tcindex \
 mask 0x3 shift 0

tc filter add dev eth0 parent 2:0 protocol all prio 1 handle 2 \
 tcindex classid 2:2

tc filter add dev eth0 parent 2:0 protocol all prio 1 handle 1 \
 tcindex classid 2:1

tc filter add dev eth0 parent 1:0 protocol all prio 1 \
 handle 1:0:0 u32 divisor 1

tc filter add dev eth0 parent 1:0 protocol all prio 1 \
 u32 match u8 0x6 0xff at 9 offset at 0 mask 0f00 \
 shift 6 eat link 1:0:0

tc filter add dev eth0 parent 1:0 protocol all prio 1 \
 handle 1:0:1 u32 ht 1:0:0 match u16 0x50 0xffff at 2 \
 classid 1:1

tc filter add dev eth0 parent 1:0 protocol all prio 1 \
 u32 match u32 0x0 0x0 at 0 classid 1:2
```

Figure 5.5: tc script example implementing two FIFO queues with priority queueing [Pap04]

5.4 ALTQ vs. Linux TC

ALTQ and Linux TC are similar in some ways. Both implementations define queueing disciplines and a set of queue operations. The DiffServ components can be loaded dynamically as kernel modules. The main differences come from the architectural differences in the kernel. Linux has a network device layer that can be used to abstract the underlaying device driver. This means that the queueing operations can be done within the device layer and modifications only need to be done in this layer. In FreeBSD, such abstraction layer does

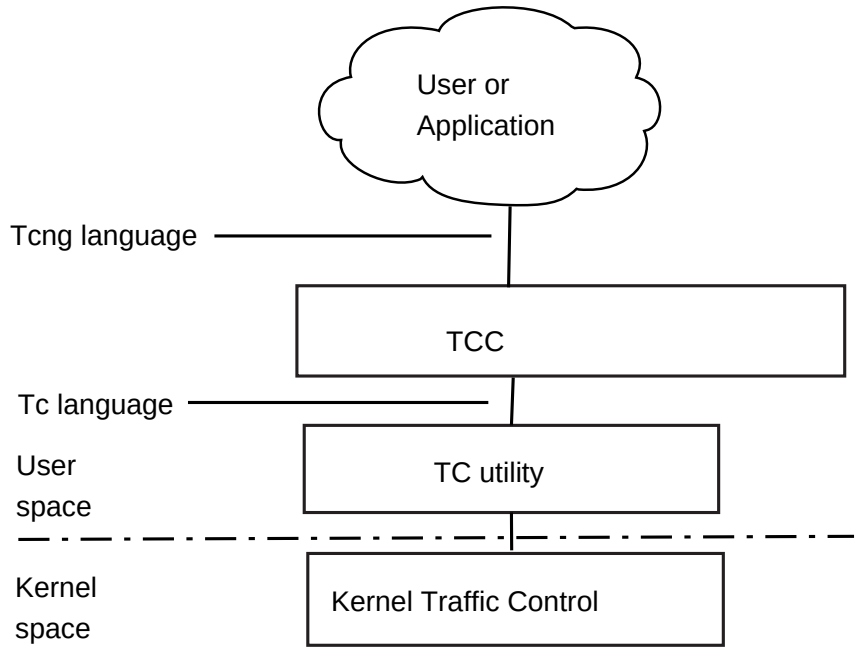


Figure 5.6: Traffic control in Linux in the next generation model

```
dev "eth0" {
          egress {

          class (<$high>) if tcp_dport == PORT_HTTP;
          class (<$low>) if 1;

          prio {
              $high = class (1) {
                     fifo (limit 20kB);
              }
                     $low = class (2) {
                            fifo (limit 100kB);
                     }
             }
      }
}
```

Figure 5.7: tcng example implementing two FIFO queues with priority queueing

not exist. As a result, modifications need to done to the device drivers. This is unfortunate as it lowers the amount of supported device drivers.

ALTQ includes an API that can be used to dynamically change the configuration parameters. This makes it easier to utilize QoS components from the upper layer programs. Linux TC does not have such an API.

Table 5.1: Comparing FreeBSD ALTQ and Linux TC

ALTQ	**Linux TC**
Need to modify network drivers	Device layer abstraction
Hardware buffer transparency	Hardware buffer is a problem
Documented	No good documentation available
Configuration scripting language	Original TC difficult to configure
DiffServ components as kernel module	DiffServ components as kernel module
ALTQ API	No API available

A major drawback in Linux TC is that it does not take into account buffering at driver level. Large buffers at driver level may negate the effect of the scheduler by delaying higher priority packets. In other words, a large buffer in the network card can spoil the whole idea of delay differentiation if the offered load is more than the physical link capacity. As explained in Section 5.2.2.2 ALTQ employs a token bucket regulator between the scheduler and the network card to solve this problem.

The summary of comparison between ALTQ and Linux TC is presented in Table 5.1.

Chapter 6

Traffic tracing and analysis tools

This chapter describes the tools that were used in the measurements and after measurements for post processing the captured data.

6.1 Packet capturing

Tcpdump[1] is a tool for capturing the network traffic. Tcpdump is a popular packet capture utility that stores the packet traces to a widely used pcap format. Tcpdump was used in the measurements to capture the TCP/IP headers of packets for later analysis.

Traffic capture was performed both at the receiving and sending side of the traffic sources. Because the link speed in the network is as low as 10 Mbps, the use of Tcpdump creates only a minimal interference to the system. When using higher link speeds the packet capture should be done on the same LAN as the receiver but not on the same machine. This could be done e.g. by using a splitter that mirrors the traffic to the packet capturing machine. The Tcpdump capture files were analyzed with a Tcptrace tool described in section 6.2

[1] Available from http://www.tcpdump.org

6.2 Tcptrace

Tcptrace [2] is a command line tool written by Shawn Ostermann at Ohio University for analysis of packet capture files. Tcptrace can be used to produce both numerical and graphical information about throughput, round trip times, window advertisement etc. The graphs can be viewed and processed to EPS format by using a graphing tool called xplot[3].

6.3 SmartBits

In order to improve the precision in one-way delay and jitter measurements the use of a specialized measurement device capable of time stamping in the hardware is highly recommended. Therefore, SmartBits 600 (SMB-600) was used for accurate delay measurements. SMB-600 provides 10 ns resolution for time stamping and displays the results with 1 μs accuracy. SMB-600 was equipped with one 10/100 Base T Ethernet (6 port) SmartMetrics card.

6.4 Altqstat

Altqstat is a user space program for ALTQ to get status on the queueing discipline. With altqstat it is possible to trace queue lengths, packet drops, transferred bytes etc. Altqstat is run in the user space that communicates with the kernel through an API. Normally altqstat is used for debugging purposes to understand better what is happening in the system. As our needs are a bit different, some modifications had to be done to the altqstat code.

6.4.1 Modifications to altqstat

Two modifications needed to be done to the original altqstat program:

1. Increasing the internal resolution
2. Modifying the output formatting for easier post processing

[2] Available from http://irg.cs.ohiou.edu/software/tcptrace/
[3] Available from http://www.xplot.org/

The resolution in the altqstat is normally one second. This polling interval can be set by using the *-w* (wait) parameter. This is, however, too coarse for our needs and therefore the resolution was increased to one microsecond by replacing a function *sleep()* with a more accurate *usleep()* function. However, using a microsecond resolution increases the system overhead due to I/O operations and creates very large log files. Hence, altqstat was used in the measurements with a 10 ms resolution.

The output formatting in altqstat is good for interactive observations but bad for post processing. Therefore, the output formatting was modified to a format that is easy to parse and process. A tabulator separated column style was chosen as the output format. These files can be visualized directly using e.g. Gnuplot graphing tool.

6.5 Perl scripts

Numerous Perl scripts were written in order to parse different log files and to process data to a desired format. Perl is a high-level programming language with excellent file and text manipulation facilities. This makes it particularly suitable for our data processing needs.

Chapter 7

Measurement setup

7.1 Overview

For the measurements, an isolated fully functioning DiffServ network with various traffic sources was built. ALTQ traffic management software was used in all experiments to provide the support for differentiated services.

7.2 Technology and topology

The measurement network was built on a standard PC hardware with Ethernet interfaces. This technology was chosen due to its low price and flexibility. PC offers a good platform for research and development with low cost and flexibility to make changes both to the hardware and software. The network includes altogether 20 PCs. The network topology was designed to have crossing traffic and dissimilar delay paths. With the available hardware, we ended up with the topology presented in figure 7.1.

The routers in the network are 1.3 GHz AMD machines with 256 MB of RAM. These routers use a FreeBSD 4.5 operating system that is patched with ALTQ using 1000 Hz kernel clock. The routers have four 10/100 Mbps Ethernet interfaces that are configured to 10 Mbps full-duplex mode. Therefore, the link speed in the network is 10 Mbps and the maximum transfer unit is 1500 bytes.

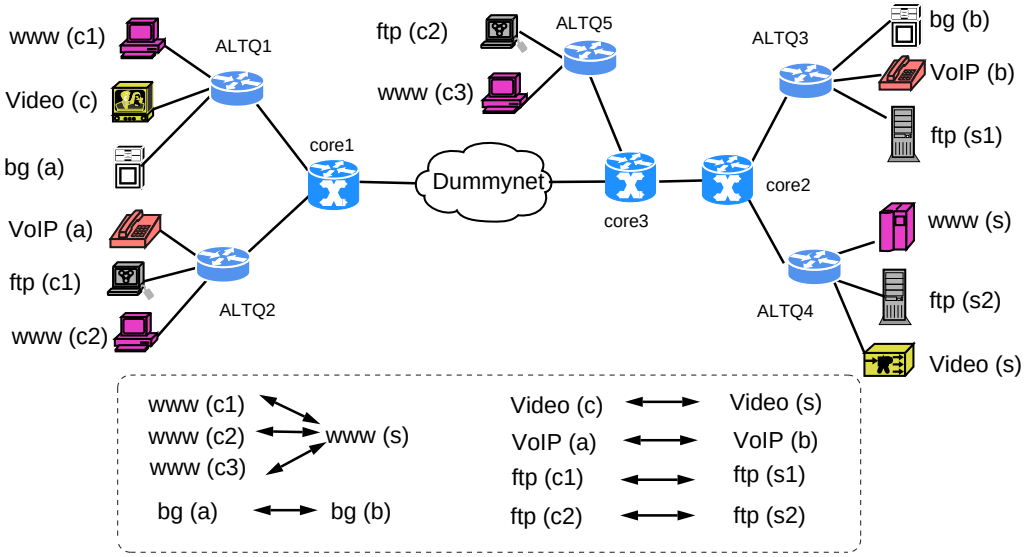


Figure 7.1: Measurement network topology and connection pairs

7.3 Baseline delay measurements

SmartBits was used to determine end-to-end delay in an unloaded network (Figure 7.1) and to verify that the delay emulation using Dummynet really works. This will give us also the baseline when analyzing the delay measurements. In this measurement UDP packets of size 1024 B are sent at the rate of 10 kbit/s bi-directionally during one minute. The one-way-delay in an unloaded BE network (altq daemon disabled) is described in table 7.1. The delay with altq daemon enabled is shown in table 7.2. We can see that the average delay in the high delay path is about 35 ms and 3.5 ms in the low delay path. The delay emulation appears to work properly. Enabling altq daemon increases the average delay by less than 0.1 ms.

Table 7.1: End-to-end latency in the unloaded BE network

	Min delay (ms)	Avg delay (ms)	Max delay (ms)
ALTQ1→ALTQ3	34.40	34.84	35.29
ALTQ3→ALTQ1	34.42	34.82	35.30
ALTQ5→ALTQ4	3.55	3.59	3.68
ALTQ4→ALTQ5	3.55	3.58	3.61

Table 7.2: End-to-end latency in the unloaded network with ALTQ

	Min delay (ms)	Avg delay (ms)	Max delay (ms)
ALTQ1→ALTQ3	34.476	34.907	35.339
ALTQ3→ALTQ1	34.596	34.962	35.364
ALTQ5→ALTQ4	3.6363	3.6639	3.7171
ALTQ4→ALTQ5	3.6399	3.6683	3.7212

7.4 Traffic sources

A motivation for traffic generation was to produce a realistic mix of applications found in the modern Internet that covers the basic traffic types. The testbed includes several traffic generators capable of emulating a wide range of applications. These traffic generators are used to create both constant bit rate (CBR) and variable bit rate (VBR) traffic. As a transport protocol we use UDP or TCP depending on the application type. All TCP traffic is created by using standard PC hardware running Linux or FreeBSD operating system in order to provide a full TCP stack. Also additional dedicated measurement hardware (SmartBits 600 and Adtech AX/4000) was used to create UDP based traffic and to measure accurately one-way delay and delay variation. The characteristics of traffic generator PCs are presented in Table 7.3.

Table 7.3: Traffic generator PCs used in measurements

	CPU (MHz)	OS	RAM (MB)
Video (c)	Intel 133	Linux 2.4.20	32
Video (s)	Intel 133	Linux 2.4.20	64
www (c1)	Intel 133	Linux 2.4.20	32
www (c2)	Intel 133	Linux 2.4.20	32
www (c3)	Intel 133	Linux 2.4.20	32
www (s)	AMD 1300	FreeBSD 5.0	256
ftp (c1)	AMD 1300	FreeBSD 4.5	256
ftp (c2)	Intel 433	Linux 2.4.20	256
ftp (s1)	Intel 133	Linux 2.4.20	64
ftp (s2)	Intel 133	Linux 2.4.20	32
Controller	Intel 433	W2k	128

7.4.1 Voice over IP

SmartBits 600 with the SmartVoIPQoS test application was used to generate packets that simulate the call patterns of voice traffic. With SmartVoIPQoS, we were able to simulate several users using VoIP services and determine the

achieved quality of service. We used G-711 μ-law codec with 20 ms framing time. This combination produces packets of size 218 bytes with a constant transmission rate of 87.2 kbps (50 packets/s). We emulated altogether 20 simultaneous voice calls with bi-directional behavior.

7.4.1.1 Perceptual Speech Quality Measure

The overall quality of a telephone conversation can be determined based on the metrics that the measurement device is capable of measuring i.e. packet loss, one-way delay, and jitter. These metrics can be mapped to a Perceptual Speech Quality Measure (PSQM) voice scoring system. PSQM (ITU-T recommendation P.861) is a scoring system that illustrates the overall quality through one quantity. It was originally developed to test audio codecs but it has been also widely used for evaluating VoIP systems. PSQM is a more objective method than MOS for scoring voice quality and is the method SmartVoIPQoS uses to rate voice quality. The basic operation of the PSQM process is shown in Figure 7.2. The PSQM algorithm compares the original signal with the signal that has been through the coding and decoding process and outputs the PSQM score.

The mapping between measurement results and the PSQM values is done by SmartVoIPQoS using a predefined PSQM matrix. This matrix is created by applying PSQM values from low to high for test signals with various levels of impairment (packet loss, jitter). The PSQM values range from 0 (most desirable) to 6.5 (least desirable). The PSQM values are affected by packet loss, jitter and the type of codec used. With the G.711 codec, the best achievable value is 0.4.

PSQM scores can be roughly converted into a MOS value. A PSQM score of 0 (very good quality) translates to a MOS value of 5, and a PSQM value of 6.5 (very bad quality) translates to a MOS value of 1.

7.4.2 Video streaming

RUDE (Real-time UDP Data Emitter)[1] was used to transmit a real time traffic pattern to the collector for RUDE (CRUDE). RUDE/CRUDE is capable of time stamping the packets with 1 μs resolution. The traffic pattern was

[1] Available from http://rude.sourceforge.net/

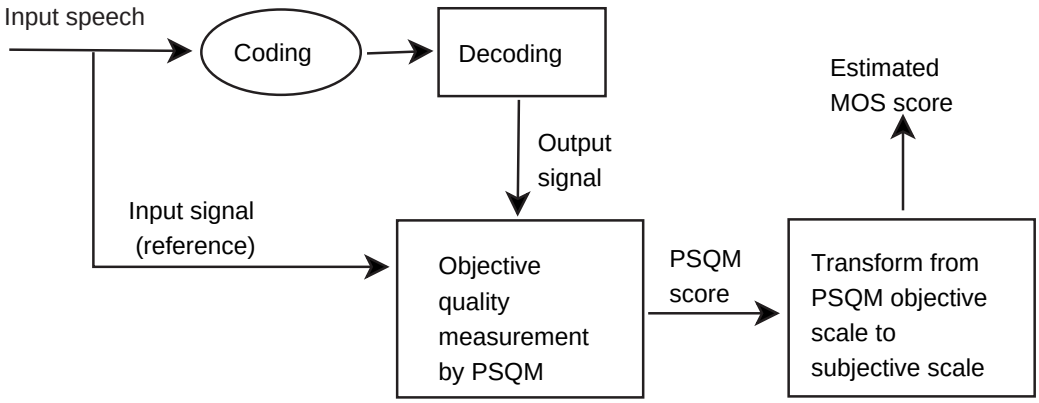


Figure 7.2: PSQM testing process

generated from a publicly available video trace [FR00]. We used a video trace from a movie 'Mr. Bean'. This movie was encoded with the MPEG-4 video codec with 25 frames/s and a Group of Pictures (GOP) of I BB P BB P BB P BB. The mean bit rate for the video stream was 130 kbps with a peak rate of 595 kbps. The video stream data rate profile using a 100 ms resolution during the first minute is shown in figure 7.3. We can see the bursty nature of the MPEG-4 encoded video. This is due to the highly varying frame size within I B and P frames as can be seen in Figure 7.4 that presents a histogram and a cumulative distribution function for video packet sizes. To avoid fragmentation the original movie frames exceeding the MTU were split into smaller pieces.

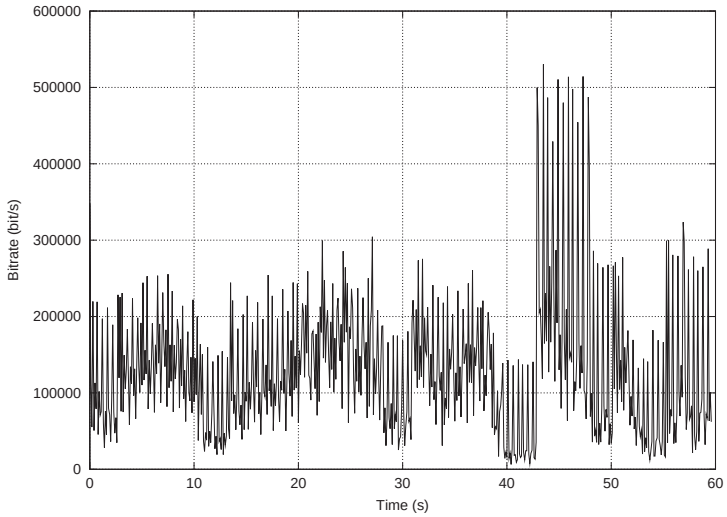
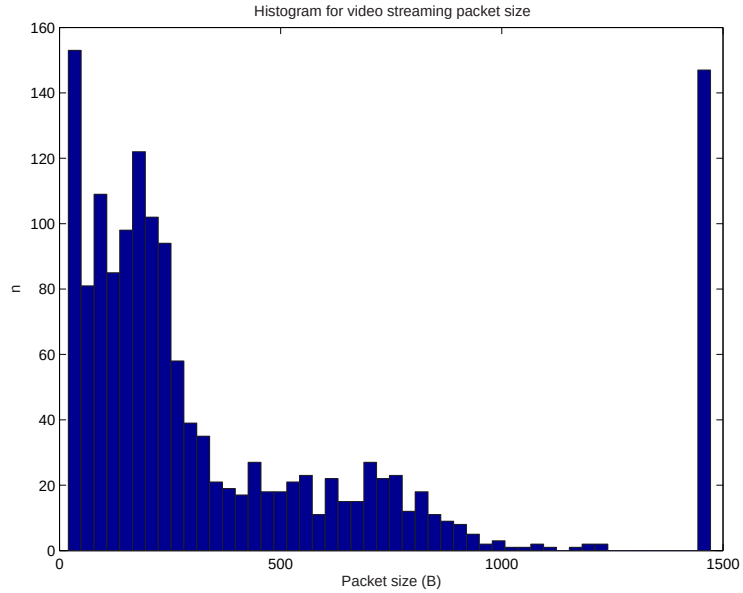
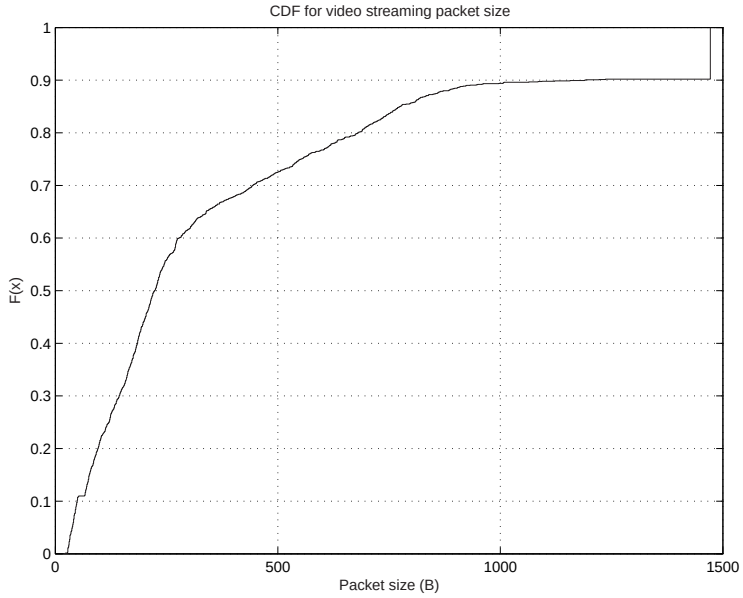


Figure 7.3: Video data rate profile in an unloaded BE network



(a) Packet size histogram



(b) Cumulative distribution function for packet sizes

Figure 7.4: Packet size histogram and empirical cumulative distribution function for video packets

7.4.3 World Wide Web

To model HTTP client-server transactions we used a real WWW-server (Apache 2.0) with Siege[2] clients. Siege is a benchmarking utility that can be used to simulate a large number of users from the same client machine. Siege clients were configured with the following parameters:

[2] Available from http://www.joedog.org/siege/index.php

- *Reading time*: Reading time is the time between two consequent page request i.e the time the user uses to read the page before fetching another page. A random number between 0 and 12 seconds was used as a reading time.
- *Concurrent users*: Concurrent users defines the number of users that communicate with the web server. It should be noted that as it takes a time for the user to read the page, this parameter does not represent the amount of concurrent sessions. The number of concurrent users was set to 55. This creates about 20 concurrent sessions and 500 page fetches per client during the measurement period.
- *Object size*: The object size defines the size of a page in bytes in the web server. The object size was modeled as a geometric distribution with a mean size of 10 kB.
- *Protocol*: The protocol can be chosen between HTTP 1.0 and HTTP 1.1. HTTP version 1.0 was used in all experiments.

The throughput profile of a Siege client (www c1) is shown in Figure 7.5. As object sizes are small and the user idles during the reading time, the traffic generated is very bursty in nature. The packet size distribution is shown in Figure 7.6.

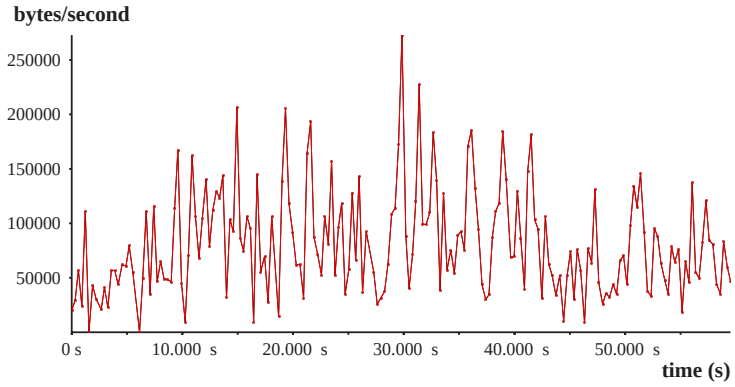
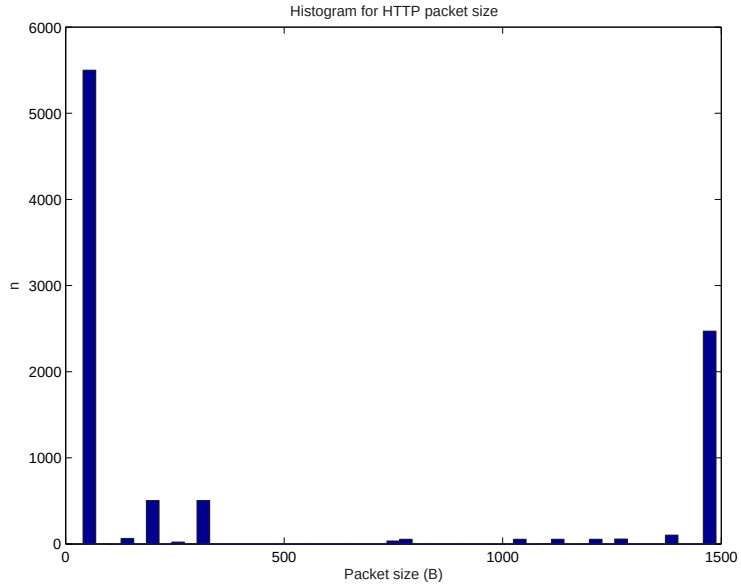


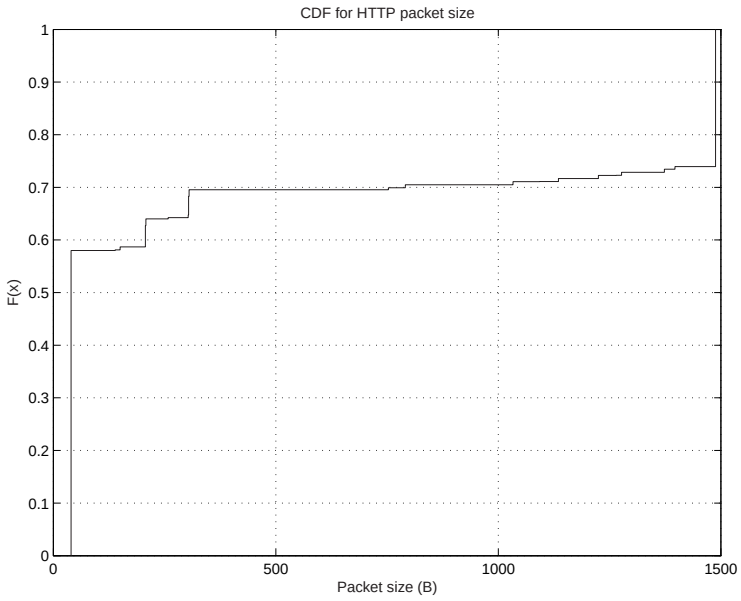
Figure 7.5: WWW client throughput in an unloaded BE network with 300 ms resolution

7.4.4 File Transfer Protocol

FTP connections were modelled with a client-server application called Kilent-Server written in Perl by Markus Peuhkuri. This application was modified to



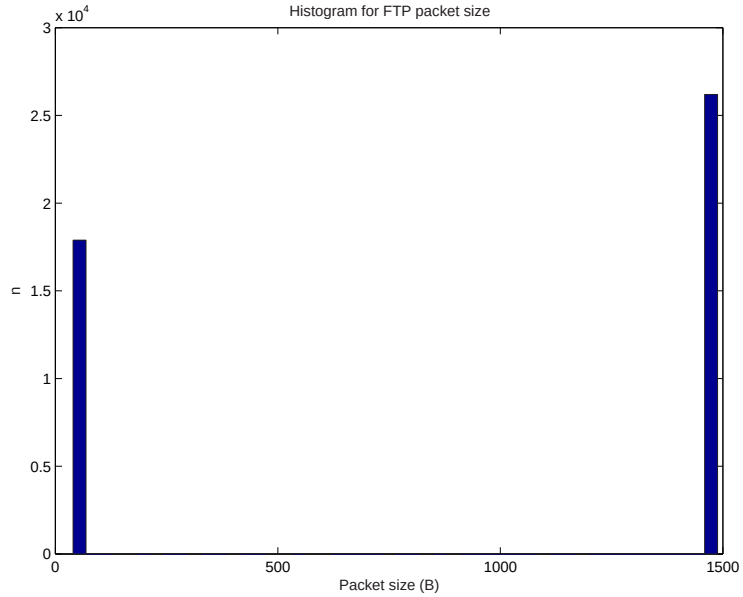
(a) Packet size histogram



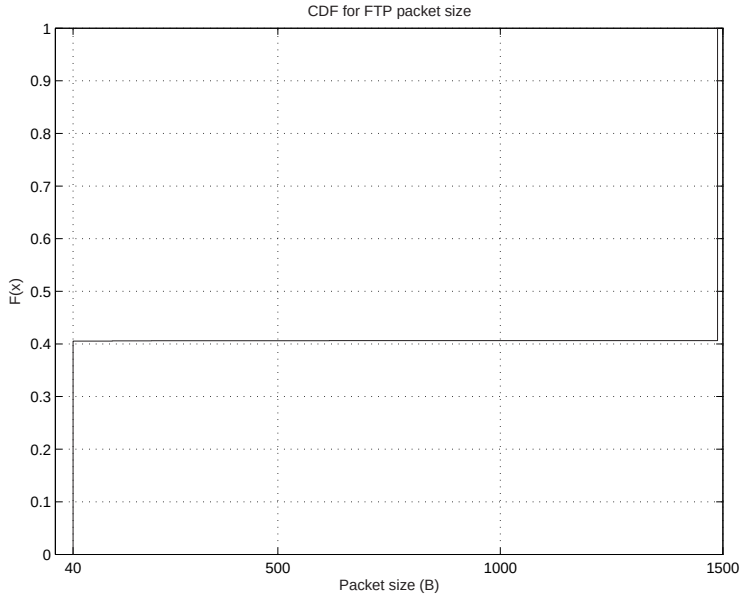
(b) Cumulative distribution function for packet sizes

Figure 7.6: Packet size histogram and empirical cumulative distribution function for HTTP packets

better emulate FTP transactions. An additional byte counter was included to the Kilent-Server application in order to easily create different size traffic transfers. Each FTP client establishes 20 connections to the server machine. This emulates 20 users downloading a single file from the server. The file sizes were geometrically distributed with a mean size of 3 MB. As the files are rather large, the majority of packets sent in the network are either of the size of the MTU or 40 bytes (acknowledgements) as can be seen in Figure 7.7.



(a) Packet size histogram



(b) Cumulative distribution function for packet sizes

Figure 7.7: Packet size histogram and empirical cumulative distribution function for FTP packets

The throughput of an FTP client (ftp c1) in the measurement network without other traffic is plotted in Figure 7.8. We can see that the ftp client is able to make almost the full use of the resources, as there are no idle times as was the case with the WWW client.

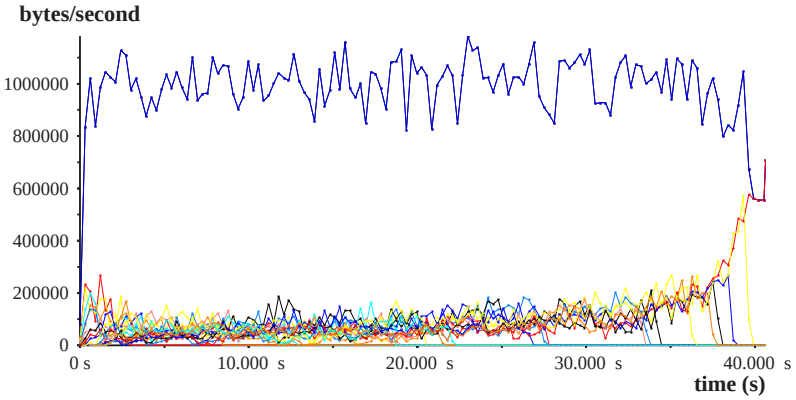


Figure 7.8: FTP client throughput (per connection and total) in an unloaded BE network with 300 ms resolution

7.4.5 Background traffic

The Adtech AX/4000 traffic generator was used to create UDP/IP traffic to the network. Adtech's IP generator was configured to create a Markov modulated poisson process with random bursts. The packet size was modeled with a quad-modal distribution (i.e. four Gaussian distributions superimposed). This traffic type does not aim to model any particular application. It is used to create upstream congestion to the network.

7.5 Buffer management

The buffer sizes for traffic classes at the router's output link were chosen to satisfy two operational aspects. For real-time traffic, a small buffer size with a simple tail drop algorithm was used in order to provide minimum latency. For TCP based traffic a bigger buffer using the RED algorithm was used to enable high link utilization and to prevent synchronization. The widely used "rule-of-thumb" [VS94] was used as a reference for setting the buffer size to a reasonable level for TCP sources. The rule states that the delay-bandwidth product describes the required buffer size. This holds quite well for low-speed links with only few TCP connections. However, recent studies have shown that the rule-of-thumb for setting the size equal to the bandwidth-delay product is outdated and incorrect for high-speed routers serving highly aggregated traffic [BSM04][AKM04]. The maximum buffer size in ALTQ/CBQ that can be set for a class is 200 packets.[3]

[3]The maximum buffer size can be increased by modifying the sources and recompiling the kernel

7.5.1 Setting the buffer size

The buffer length can be set in ALTQ using the *maxdelay* parameter. ALTQ calculates the buffer size (in packets) based on the given *maxdelay*, *average packet size* and the *bandwidth* assigned for the class using the equations 7.1 and 7.2. NS_PER_MS and NS_PER_SEC are scaling factors where the former refers to 10^6 and the latter to 10^9. Due to this strange way of setting the buffer size in ALTQ, we need to calculate it beforehand and configure it to ALTQ through the *maxdelay* parameter. The default value of 1500 bytes for average packet size was used in all measurements.

$$Buf_{size} = (maxdelay * NS\_PER\_MS)/(nsPerByte * avg\_pkt\_size) \quad (7.1)$$

$$nsPerByte = 1/bandwidth * NS\_PER\_SEC * 8 \quad (7.2)$$

7.6 Clock synchronization

The system clocks at different network nodes do not necessary show the same time and often they run even with different frequency. This is usually the case when dealing with the timing functionality built inside the PC hardware. In our measurement network, we have two different needs for clock synchronization:

1. Scheduling of events at the same time.
2. Measuring one-way delay in the network.

7.6.1 Network Time Protocol

Network Time Protocol (NTP) is a widely used architecture for synchronizing time among distributed client and server machines. NTP was first described in RFC 958 [Mil85] but it has evolved many changes after that. The newest standardized version of a full NTP architecture is version 3 [Mil92]. NTP version 4 is a significant revision of the NTP standard, which is also the current development version.

NTP version 4.1.0-8 is used in the test network for clock synchronism. One of the traffic generator PCs (ftp s1) is acting as the NTP server. All other PCs in the network are synchronized to this server by running a NTP daemon that polls the server machine and adjust the local clock to the same time. The clock offset during 24 hours period is shown in Figure 7.9. We can see that the offset oscillates within about -1 ms and + 1 ms. Therefore, by using the local clock in the server machine as a reference clock we achieve about 1 ms accuracy. For a higher accuracy, we could use e.g. a Global Positioning System (GPS) receiver as the reference clock.

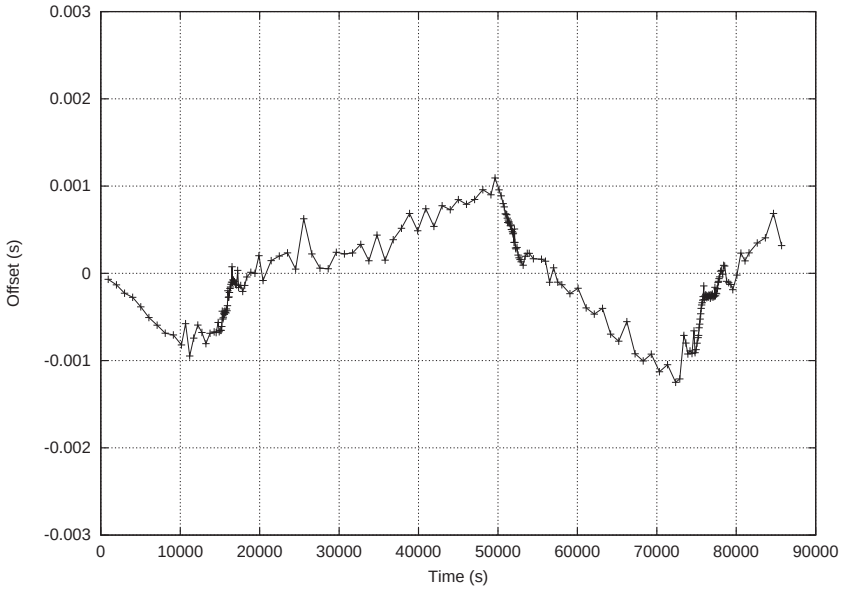


Figure 7.9: Clock offset during 24 hours period

7.7 Delay emulation

An important factor when evaluating the achieved performance in the network is the effect of end-to-end delay. Our test network has a very low end-to-end delay and the delay is about the same in all traffic paths. Therefore, we needed to artificially create delay to our network for certain paths. For delay creation we used the dummynet [Riz97] network emulator.

7.7.1 Dummynet

Dummynet is a tool for emulating queue and bandwidth limitations, delays, packet losses, and multipath effects. Dummynet is integrated into the FreeBSD

operating system but it can be used also as a standalone version that fits on a floppy disk.

The emulation of limited network resources is carried out in the dummynet by passing the packets through two control queues, namely R-queue and P-queue. A P and R queue pair is needed in each communication direction. These queues are located between the protocol layer under observation and the lower layer (Figure 7.10). The R and P queues are used to implement a communication link called a *pipe*. The following processing rules are used when exchanging traffic between two protocol layers: [Riz98]

1. Packets are first inserted in the R-queue bounded by the maximum queue size simulating the effect of limited size buffer. A queueing policy (usually FIFO with taildrop) is used to define which packets are inserted to the queue.
2. Packets are transferred from the R-queue to the P-queue at a maximum rate of B bytes per second simulating the bandwidth limitations on the communication media.
3. Packets remain in the P-queue for a predefined amount of seconds in order to emulate delay on the link. After packets are delayed they are moved to the next layer in the protocol stack.

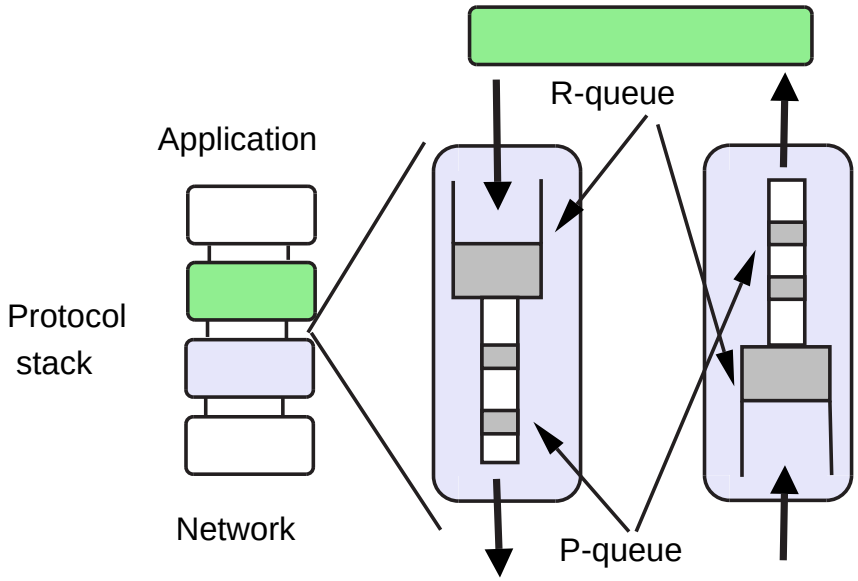


Figure 7.10: The operation principle in dummynet

Dummynet was used in the core of our network to create a high delay path. Therefore two clients, www(c3) and ftp(c2) are connected to the server through

the low delay path whereas the other clients are using the high delay path (Figure 7.1).

7.8 Measurement procedure

The PC-based traffic sources were scheduled to begin their transmission at the same time. A one-minute warm-up period was used before the actual measurements. VoIP traffic was started about 7 seconds from the beginning of measurement as a 45 second burst. The measurement period was 60 seconds during which the data was collected. The data was post-processed offline after the measurements. Measurements were repeated five times in order to get statistical validity.

7.9 Terminology

7.9.1 Delay

One-way delay is computed from the receiver (Rx) and transmit (Tx) time stamps by simply calculating the difference as noted in equation 7.3.

$$Delay \; = \; (Rx \; timestamp) \; - \; (Tx \; timestamp) \tag{7.3}$$

7.9.2 Jitter

Jitter is calculated by comparing the delay of each packet and its subsequent packet. Let's assume we have a flow containing packets 1,2,3,4 and so on. The jitter for each subsequent packet pair (individual jitter) is calculated as: [Inc01]

$Jitter_{1,2}$= ||Packet 2's (Rx Time - Tx Time) - Packet 1's (Rx Time - Tx Time)||

$Jitter_{2,3}$= ||Packet 3's (Rx Time - Tx Time) - Packet 2's (Rx Time - Tx Time)||

$Jitter_{3,4}$= ||Packet 4's (Rx Time - Tx Time) - Packet 3's (Rx Time - Tx Time)||

⋮

The average jitter is then defined as:

$$Average\ jitter = \frac{\sum individual\ jitters}{total\ number\ of\ packets\ received} \tag{7.4}$$

7.9.3 Packet loss

Packet loss is defined as a ratio between received (Rx) and sent (Tx) packets:

$$Packet\ loss = \frac{Tx\ packets - Rx\ packets}{Tx\ packets} * 100\% \tag{7.5}$$

7.9.4 Throughput

The throughput is the amount of data transferred from one place to another in a specified amount of time:

$$Throughput = \frac{Data\ transferred}{Transfer\ time} \tag{7.6}$$

In this work we use two different throughput definitions, namely per connection and aggregate throughput. Per connection throughput defines the throughput for an individual connection (during its lifetime) whereas aggregate throughput presents the total throughput for all connections.

Chapter 8

Results

8.1 The level of differentiation

This section presents the results from the measurements where we increase the level of differentiation step by step. From these measurements we want to get the answer to the following questions:

1. How many traffic classes are needed?
2. What traffic types need to be separated?

Table 8.2 shows the classification of traffic into classes and the provisioning in different differentiation levels. The provisioning in these measurements is done in a conservative way. We knew the amount of real-time traffic and used this information when provisioning the real-time class. It should be noted that 2% of the link bandwidth is reserved for the control traffic. ALTQ requires a control class always to be defined. The control class is used here to transmit traffic initiated from SSH and NTP. The delivery of NTP packets is essential in order to keep the clocks in synchronism. Dropping policies and buffer sizes are shown in Table 8.1.

8.1.1 Best Effort model

The measurements were made first in a conventional best effort (BE) network meaning that there is no differentiation between data flows. This is the situation on the current Internet. Previous studies [Luo00][AL03][LA04] have shown

Table 8.1: Queue management algorithm and buffer size (packets) for traffic classes

	VoIP	**Video**	**HTTP**	**FTP**
1 class	RED (215)			
2 classes	FIFO (15)		RED (200)	
3 classes	FIFO (15)		RED (70)	RED (130)
4 classes	FIFO (5)	FIFO (10)	RED (70)	RED (130)

Table 8.2: Provisioning of the link capacity in the measurements

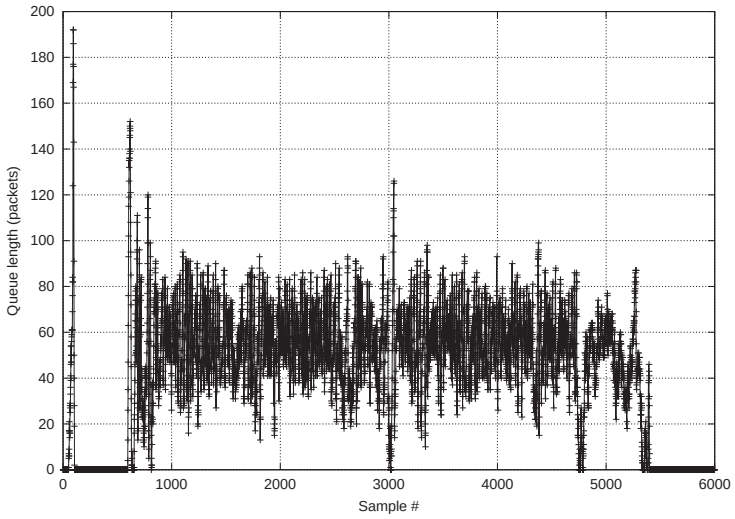
	VoIP	**Video**	**HTTP**	**FTP**
1 class	9.8 Mbps			
2 classes	3.0 Mbps		6.8 Mbps	
3 classes	3.0 Mbps		2.0 Mpbs	4.8 Mbps
4 classes	2.0 Mbps	1.0 Mbps	2.0 Mbps	4.8 Mbps

that three or four traffic classes are needed. These results were derived from simulations that used similar traffic patterns as in this experimental study.

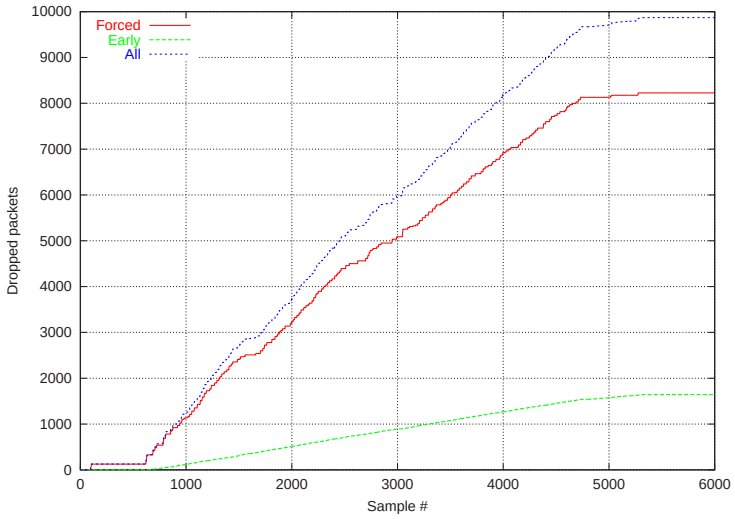
Figure 8.1 shows the queue length and dropped packets (cumulative) using 10 ms resolution at the bottleneck link. We can see how RED is trying to keep the average queue length low by dropping packets before the buffer overflows (early drops in Figure 8.1(b)). Despite that, the buffer is constantly filling up. This causes extra delay for interactive applications and lowers the throughput of elastic applications due to packet drops.

Figure 8.2 presents the one-way delay for subsequent packets in a video stream plotted in a xy-diagram. The variation in delay (jitter) can be observed from the deviation of points from the line $y=x$. We can see that the delay is spread to a large area varying from about 40 ms to 310 ms with about 180 ms center. For an application with hard real-time requirements like video conferencing, the delay is too much to overcome through buffering. For a video streaming with non bi-directional behavior, a large playback buffer could compensate the high delay. However, high packet loss (7 %) lowers the quality of streaming video to a very low level.

The statistics for VoIP are shown in Table 8.3. The one-way delay is on average 180 ms that varies between the minimum (92 ms) and maximum (250 ms) value. Average packet loss for VoIP traffic is 9.7 %. The high packet loss combined with a jitter of 5.5 ms yields a PSQM score of 2.9. The high PSQM and delay values indicate that the use of telephony is impossible in a single class case.



(a) Queue length in the BE case



(b) Cumulative packet drops in the BE case

Figure 8.1: Queue length and packet drops in the BE case

A sequence number plot for a FTP connection downloading a file is shown in Figure 8.3. The throughput can be observed from the slope of the curve. The decreasing gradient indicates a reduced data rate.

Figure 8.4 shows the per object (connection) and aggregate (client) throughputs for the TCP clients. We can see that FTP clients are getting more bandwidth than WWW clients. This is due to idle times between the page requests in the HTTP connections and therefore FTP is capable of utilizating the bandwidth resources more efficiently. The connection throughput for the FTP clients is higher than for the WWW clients as the long lasting TCP connections have reached the steady state. The short-lived WWW connections operate mainly in slow start mode and therefore are not able to efficiently

Table 8.3: VoIP statistics (minimum delay, average delay, maximum delay, jitter, packet loss and PSQM) in the best effort model

#	Min delay	Avg delay	Max delay	Jitter	Ploss	PSQM
1	56.80	183.24	231.40	5.56	10.44	3.01
2	55.51	177.97	233.19	5.28	8.39	2.81
3	145.12	184.48	305.91	5.57	10.33	3.00
4	143.61	184.39	244.79	5.46	10.03	2.97
5	57.75	179.91	243.61	5.45	9.43	2.90
Avg	91.76 ms	182.00 ms	251.78 ms	5.46 ms	9.72 %	2.94
Stdev	48.03	2.92	30.85	0.12	0.84	0.08

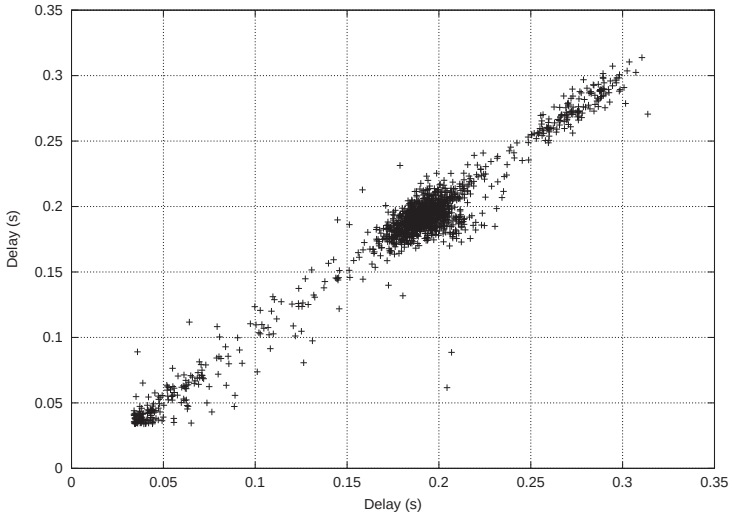


Figure 8.2: Delay and jitter for video streaming in a BE network

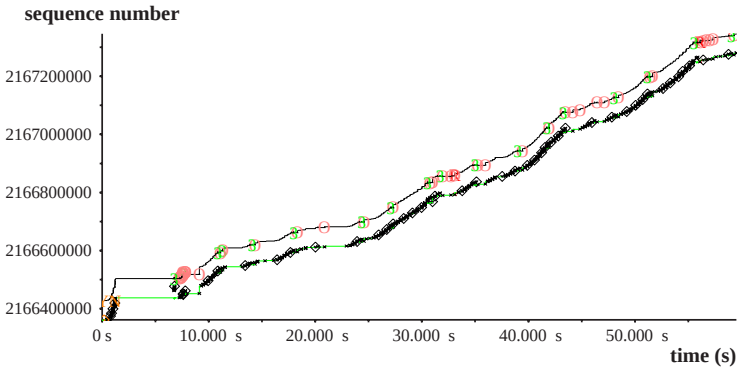
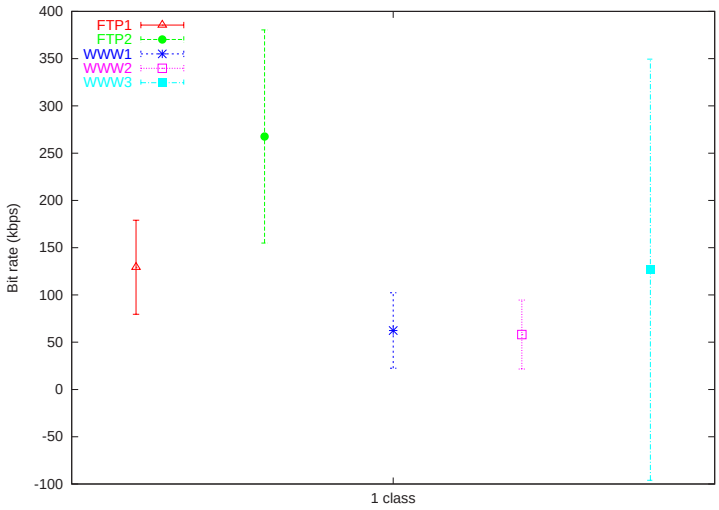


Figure 8.3: Sequence number plot for a ftp connection

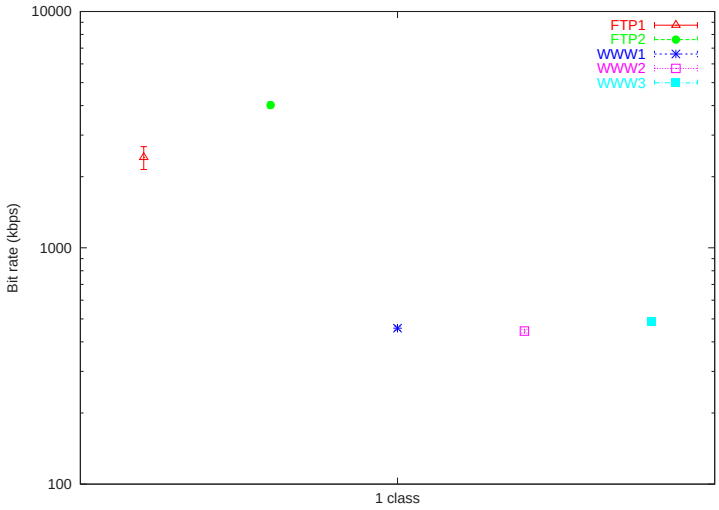
utilize the available bandwidth when file sizes are small.

The effect of dissimilar round-trip times can be easily observed from the Figure 8.4 (a). Connections using the low delay path (*ftp2* and *www3*) are behaving more aggressively and they receive about two times larger throughput than the connections using the high delay path. This behavior is due to the rate of

window opening, which is faster with short RTT connections[1].



(a) Connection throughput



(b) Aggregate throughput

Figure 8.4: Connection and aggregate throughput in best effort model

8.1.2 Two class model

The next step was to separate the real-time (RT) traffic from non real-time (NRT) traffic by adding one traffic class. We assigned 30 % of the bandwidth for the real-time applications (Table 8.2) and observed if this would be enough to bring the utility of these applications to an operational level. This means that the over-provisioning factor is about 1.5. As UDP based traffic is not elastic, we can quite easily calculate the amount bandwidth that is needed.

[1]See [Luo00] for details about effect of RTT to TCP operation

For the TCP based traffic, the provisioning is much more difficult as TCP tries to use all the available bandwidth and without access policing it is possible to achieve that. The CBQ link-sharing tree is shown in Figure 8.5. The classes are not allowed to borrow excess bandwidth from each other.

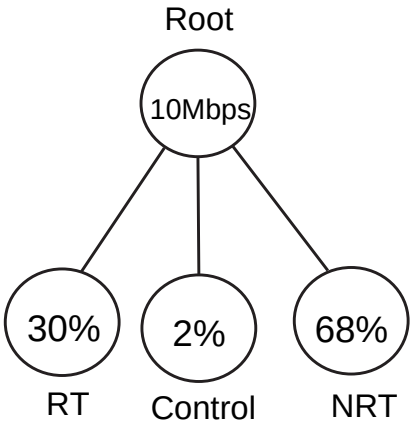


Figure 8.5: CBQ link sharing hierarchy for the two class model

The one-way delay inside the traffic classes is shown in Figure 8.6. Delay in the RT class is bounded to about 40 ms with some minor peaks that can be compensated by buffering. This is however not the case in the NRT class where the delay is oscillating between 40 ms and 220 ms. This large oscillation is usually too much for delay sensitive applications with hard requirements. The oscillation comes from the operation of TCP that tries to find the ideal sending rate. The delay rises when TCP increases the sending rate to a level that causes queues to fill up. By separating the real-time traffic from the elastic TCP traffic, we can improve the predictability and control over delay.

Delay and jitter for video traffic is shown in Figure 8.7. The delay is centered around 40 ms and not spread to a wide area as was the case in the best effort model (Figure 8.2). By applying an extra class for the real-time traffic, we also get rid of the packet loss that degraded the quality earlier to a very low level.

Table 8.4 shows that the use of telephony in the real-time class is possible. There is no packet loss and delay and jitter are within acceptable limits. PSQM score is 0.4, which is the best we can achieve with our G.711 codec. In addition the variation in the results between measurement rounds is reduced dramatically. This is due to over provisioning in the RT class that makes the class behavior more predictable.

Figure 8.8 reveals that the increased quality of service in the real-time class does not come free: over provisioning in the real-time class lowers the through-

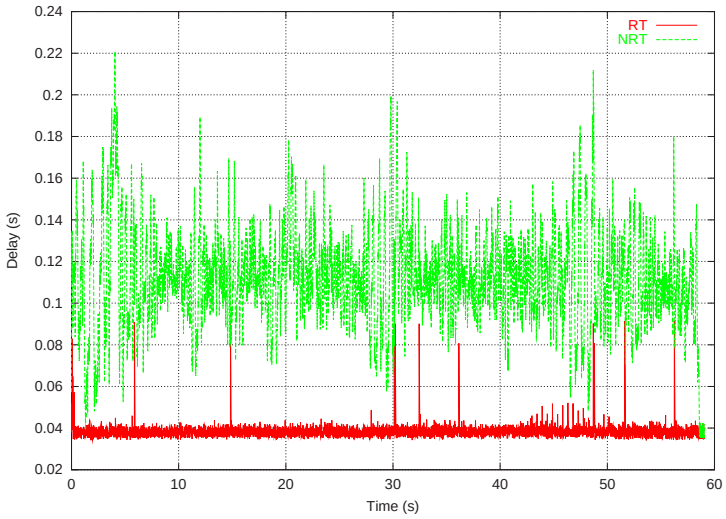


Figure 8.6: Delay for real-time (RT) and non real-time (NRT) class

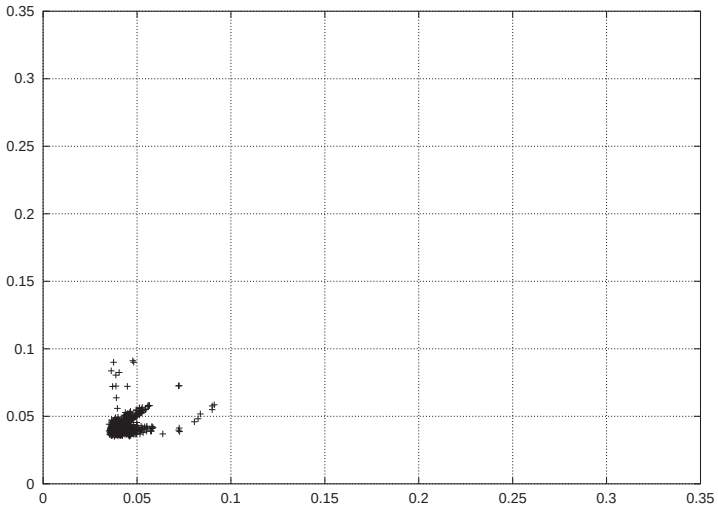


Figure 8.7: Delay and jitter for video traffic in the two class model

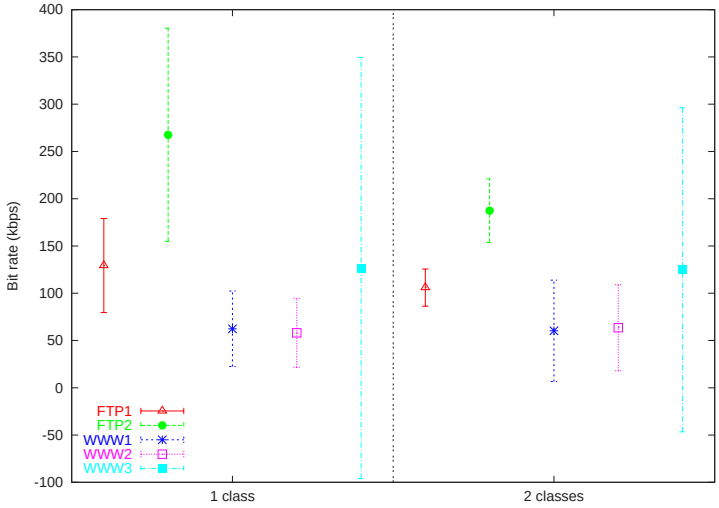
put of FTP connections. However, WWW clients do not suffer from this and their bandwidth usage is almost the same as in the BE case. What should be especially noted is the fact that differentiation helps to lower the variance in the connection throughput and we are able to maintain a more uniform throughput. This is important as the service becomes more predictable and steadier.

8.1.3 Three class model

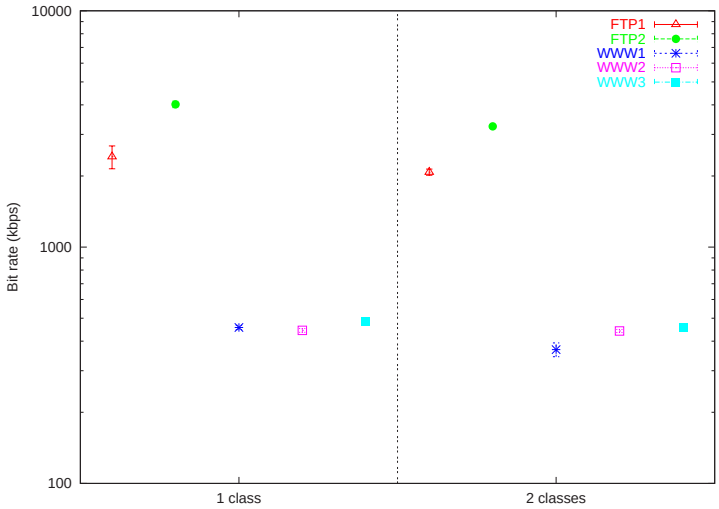
We first studied the interference between delay and bandwidth sensitive applications and separated real-time traffic from the elastic traffic. Now we

Table 8.4: VoIP statistics (minimum delay, average delay, maximum delay, jitter, packet loss and PSQM) in the two class model

#	Min delay	Avg delay	Max delay	Jitter	Ploss	PSQM
1	30.65	34.09	42.99	1.92	0.00	0.40
2	30.62	34.17	43.79	1.93	0.00	0.40
3	30.64	34.16	43.75	1.92	0.00	0.40
4	30.60	34.13	43.59	1.93	0.00	0.40
5	30.59	34.17	44.77	1.91	0.00	0.40
Avg	30.62 ms	34.14 ms	43.78 ms	1.92 ms	0.00 %	0.40
Stdev	0.03	0.03	0.64	0.01	0.00	0.00



(a) Connection throughput



(b) Aggregate throughput

Figure 8.8: Connection and aggregate throughput in the two class model

examine the interference between the short and long TCP flows[2]. In these

[2]The interference with the short and long TCP connections was explained in Section 4.1.1.1

Table 8.5: VoIP statistics (minimum delay, average delay, maximum delay, jitter, packet loss and PSQM) in the three class model

#	**Min delay**	**Avg delay**	**Max delay**	**Jitter**	**Ploss**	**PSQM**
1	30.50	35.46	46.06	2.70	0.00	0.40
2	30.51	35.60	46.03	2.75	0.00	0.40
3	30.51	35.56	46.09	2.77	0.00	0.40
4	30.52	35.43	46.19	2.65	0.00	0.40
5	30.51	35.52	45.50	2.70	0.00	0.40
Avg	30.51 ms	35.51 ms	45.97	2.71 ms	0.00 %	0.40
Stdev	0.01	0.07	0.27	0.05	0.00	0.00

measurements, we separate the WWW and FTP traffic to their own classes and provide one class for real-time traffic as in the earlier case. The link sharing hierarchy is shown in Figure 8.9. An extra middle class is created to keep the depth of the leaf classes equal from the root. FTP and WWW sources are allowed to borrow bandwidth from each other. The provisioning for the real-time (RT) class was kept the same as in the two-class model.

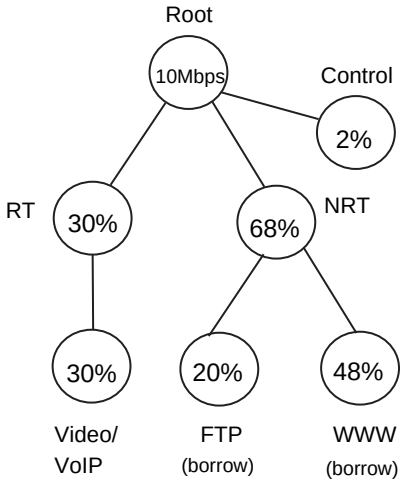
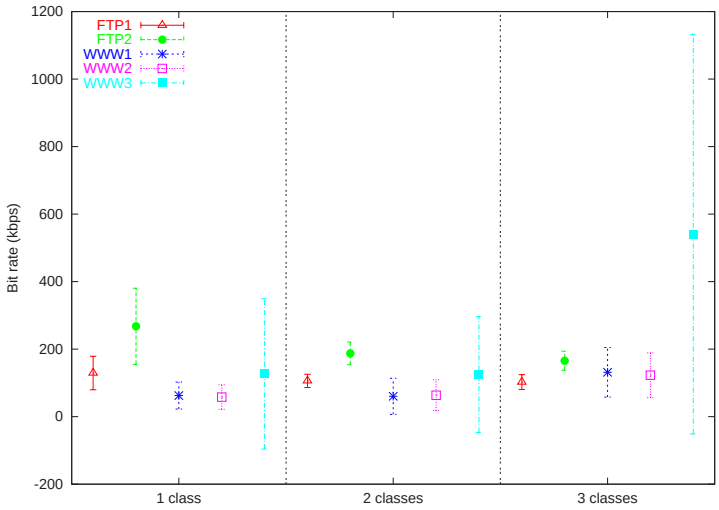


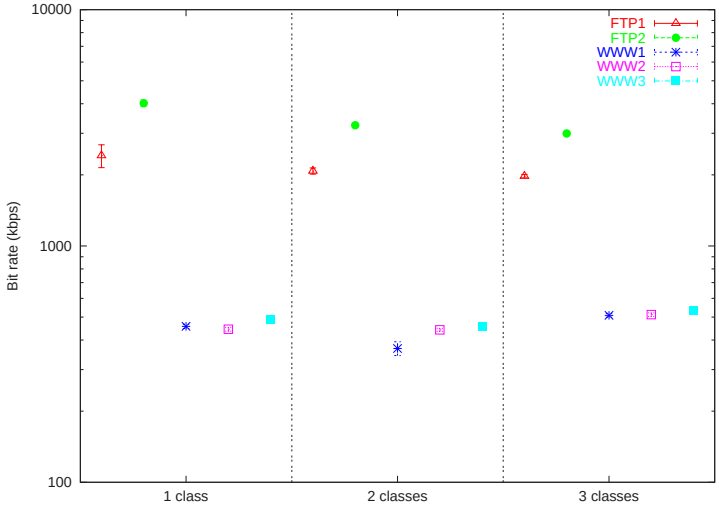
Figure 8.9: CBQ link sharing hierarchy for the three class model

As can be seen in Figure 8.10 the throughput for WWW connections increases when WWW traffic is separated to its own class. The most notable thing is the behavior of the client *www3* that tries to grasp most of the resources available in its class. The variance in the connection throughput with this client is also very high.

The addition of the class for WWW traffic increases slightly the delay and jitter for VoIP traffic (Table 8.5). The increase in jitter and delay is however so small that it has no effect to the overall quality and the PSQM score is still 0.4. Also the video experiences very similar results as in the two-class model.



(a) Connection throughput



(b) Aggregate throughput

Figure 8.10: Connection and aggregate throughput in the three class model

8.1.4 Four class model

The final thing we wanted to see in the level of differentiation was whether voice and video could be mixed together. To study this we separated the voice and video to a different class. We assigned 20 % from the link bandwidth for voice and 10 % for video traffic as can be seen in Figure 8.11. The borrowing is enabled up to the middle class but not up to the root.

The delay in the voice and video class with a different number of traffic classes is shown in Figure 8.12. There are no big differences between the delay behavior except in the best effort case (1 class) where the delay is over 180 ms for both video and VoIP. Table 8.6 shows that packet loss is zero in all levels of

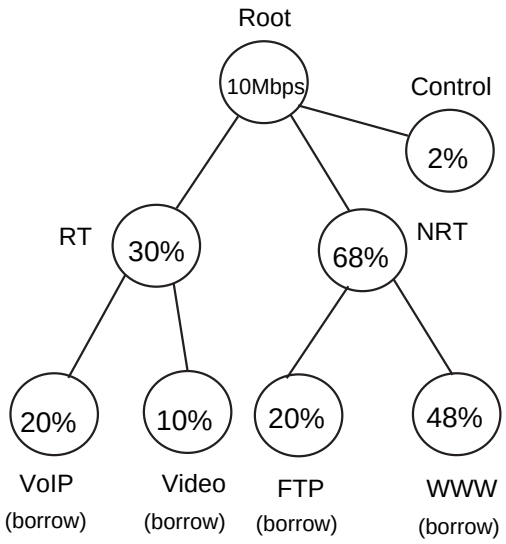


Figure 8.11: CBQ link sharing hierarchy for the four class model

Table 8.6: Packet loss (%) for real-time traffic

	VoIP	**Video**
1 Class	9.7	7.27
2 Classes	0	0
3 Classes	0	0
4 Classes	0	0

differentiation beginning from the two class model. The separation of video and voice did not seem to provide any advantage. The situation might have been different if we would have aggregated many video streams together or used a video stream with higher bandwidth usage.

The throughput for TCP based traffic sources are shown in Figure 8.13. There are no big changes in throughput when we add the fourth class. The aggregated throughput for all traffic sources is shown in Table 8.7. We can see that with more than one traffic class, the bottleneck link is not well utilized. We did not allow real-time traffic and non real-time traffic to borrow bandwidth resources from each other. As we had about 1 Mbps over-provisioning in the real-time class, the link is not totally utilized. The borrowing issues are studied in more depth in Section 8.3.

The queue lengths for different traffic classes are presented in Figure 8.14. The queue in the video class is empty all the time. In the VoIP class some packets are occasionally buffered which is reflected as increase in delay and jitter.

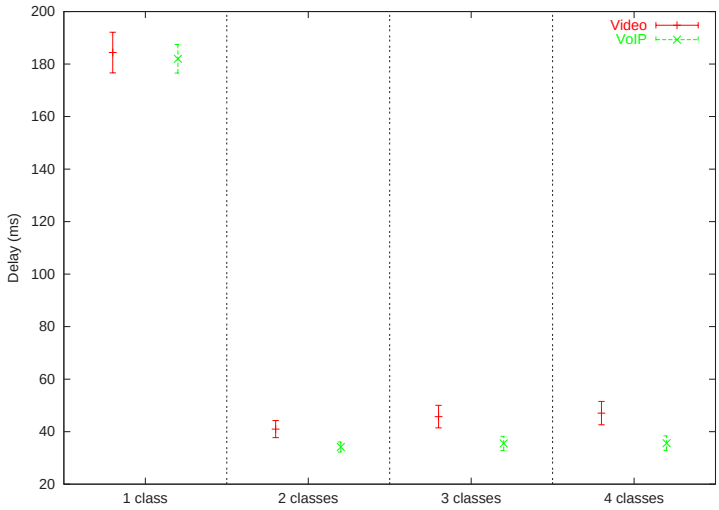


Figure 8.12: One-way delay and jitter for different levels of differentiation

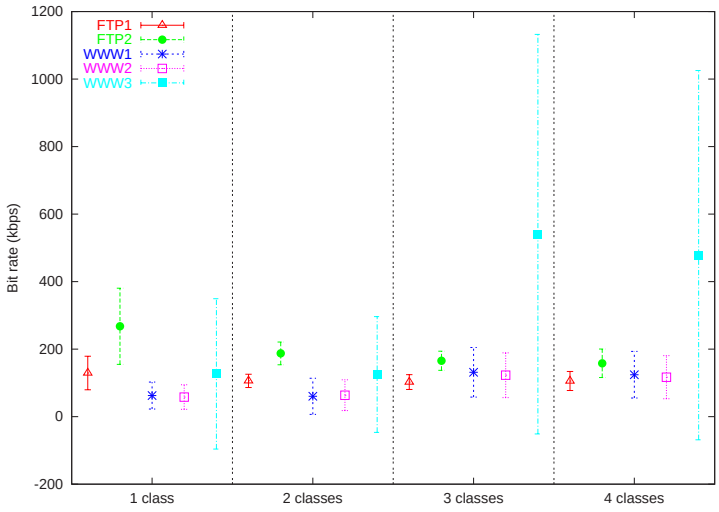
Table 8.7: Throughput for traffic sources (kbps)

	1 class	**2 classes**	**3 classes**	**4 classes**
FTP1	2412.4	2075.6	1973.4	2022.3
FTP2	4019.9	3242.4	2991.7	2994.6
WWW1	457.0	368.3	508.9	502.8
WWW2	444.2	441.6	512.3	502.8
WWW3	486.8	457.8	530.9	523.4
VoIP	1574.5	1744.0	1744.0	1744.0
Video	120.9	130.0	130.0	130.0
Total	9515.7	8459.7	8391.2	8419.9

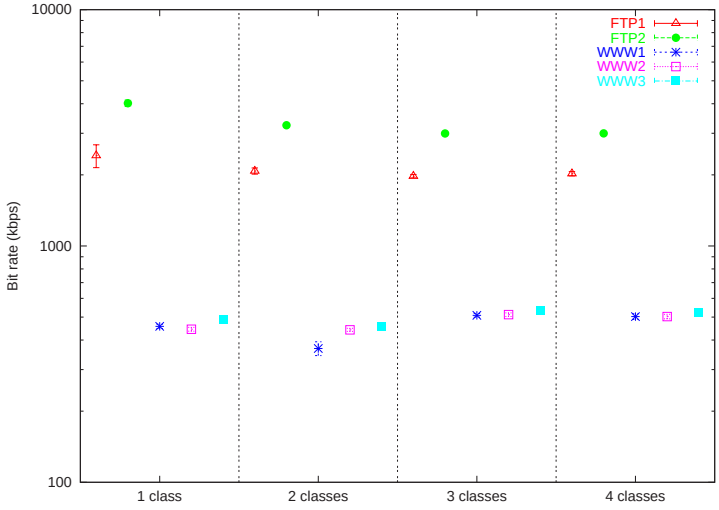
8.2 Differentiation principle

The service differentiation in the Differentiated Services network can be done in many ways. One of the popular models for service differentiation is the so-called Olympic Service Model that consists of three classes. These are called from highest to lowest quality class as *gold*, *silver* and *bronze*. The gold class is the highest quality class that is intended to provide better service for packet forwarding than silver class. Same kind of relationship is between the silver, and bronze class: silver class should provide a better probability for timely forwarding than bronze class. This provisioning model can be coupled with a pricing scheme that makes a higher quality class more costly for the user. This way a network provider can offer better quality of service for the people that are willing to pay more.

The gold class is intended for people who want to use real-time services and be able to transfer large files quickly. The gold class would be interesting for P2P



(a) Connection throughput



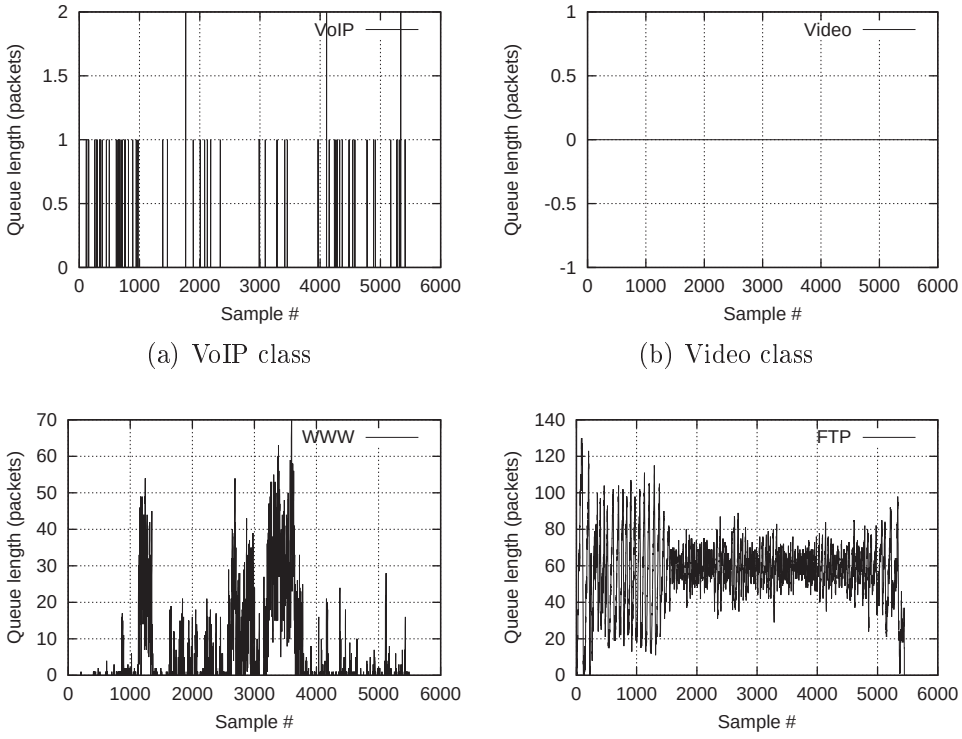
(b) Aggregate throughput

Figure 8.13: Connection and aggregate throughput in four class model

users and therefore the number of FTP connections in this class is set quite high. Silver class is for those who are willing to pay some extra to get better service but not as much as the gold class users. Bronze class is the lowest forwarding class designed for a basic Internet usage.

The traffic sources were modified so that the emulated users were divided into different pricing classes. Therefore, the total traffic volume in the network is about the same as in the earlier cases. The division into price classes and their provisioning is presented in Table 8.9[3]. The classes are set to borrow the excess bandwidth from the parent class. The queue management is shown in

[3]Values are number of data flows for VoIP, Video and FTP. For WWW the number of emulated users is shown.



(a) VoIP class

(b) Video class

(c) WWW class

(d) FTP class

Figure 8.14: Queue lengths for traffic classes in the four class model at the bottle neck link

Table 8.8: Buffer size and queue management algorithm for different price classes

Price class	Buffer size (packets)	Algorithm
Gold	15	RED
Silver	70	RED
Bronze	130	RED

table 8.8. For higher quality classes the queue length is smaller.

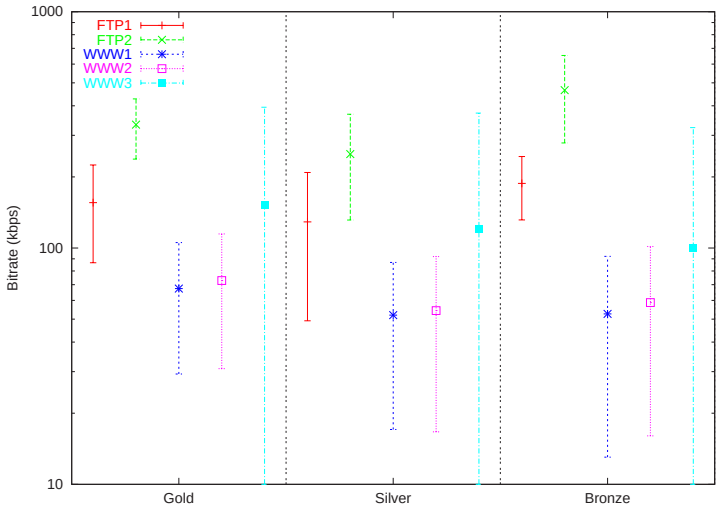
Earlier studies [Luo00][LA04][AL03] have shown that differentiation based on application type gives better overall utility than the differentiation based on price. The Olympic Service Model has been argued not to be appropriate for real-time traffic [BLS01].

Figure 8.15 gives the per connection and per client throughput for all three price classes. We can see that there are no big differences in obtained per connection throughput between service classes. Especially with the WWW connections, the difference is marginal. This means that the end users in all classes would be experiencing similar service from the network for the elastic applications regardless of the service class they have been contracted. The

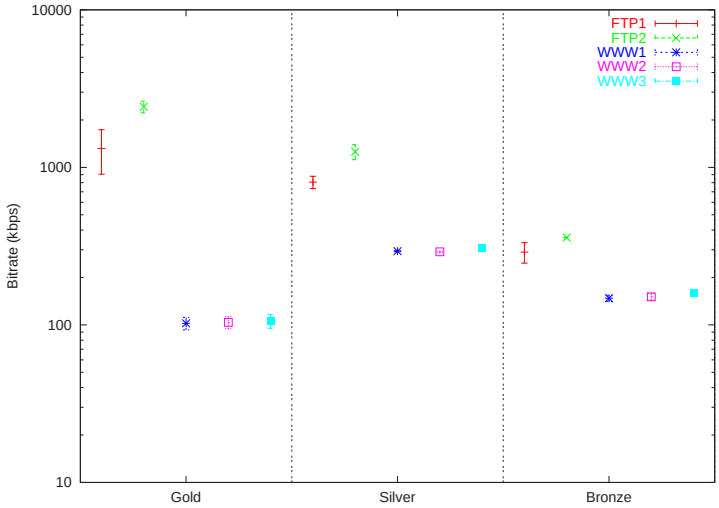
Table 8.9: The division of traffic into price classes

Price class	Provisioning	VoIP	Video	WWW	FTP
Gold	55%	10	1	10	10
Silver	33%	8	0	30	8
Bronze	10%	2	0	15	2

aggregated throughput graph reveals that the FTP clients are using most of the bandwidth allocated for the gold class. These elastic TCP flows use the available bandwidth in the class leaving below minimum amount of network resources for the real-time applications needed to perform well. This means that the level of differentiation needed for real-time applications to work well is lost.



(a) Connection throughput



(b) Aggregate throughput

Figure 8.15: Connection and aggregate throughput in olympic service model

Table 8.10: Packet loss for real-time traffic

	VoIP	**Video**
Gold	4.7	6.8
Silver	8.5	-
Bronze	4.6	-

The buffer overflow in the class can be observed in Figure 8.16 that presents the queue length at the bottleneck link. The queues are constantly filling up as a result of TCP connections that use the available bandwidth in the class. Elastic TCP connections are capable of adapting to packet loss and increased delay and still operate. Unfortunately, this does not hold for real-time traffic. Figure 8.17 shows the one-way delay for video and VoIP streams. We can see that none of the traffic classes is capable of providing a controlled delay behavior. The delay budget needed for conversational communication is used by the network due to packet buffering. High delay combined with packet loss values shown in table 8.10 yields that decent real-time communication is impossible in all service classes (gold, silver and bronze).

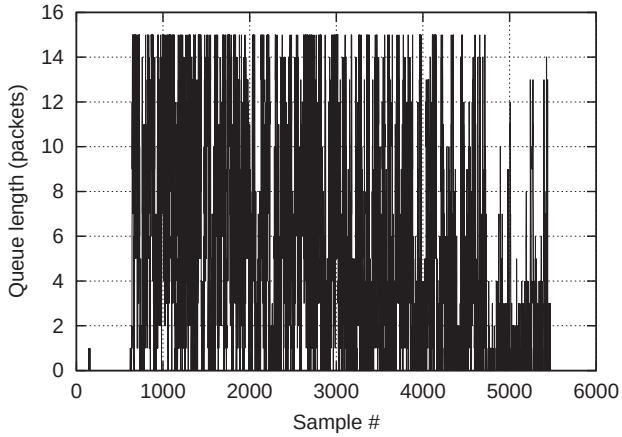
The basic idea of an olympic service model is good. It is fair that the user who pays more gets also better service from the network. However, an olympic service model does not take into account the requirements of the applications nor the interference caused by mixing different application types.

8.3 Provisioning aspects

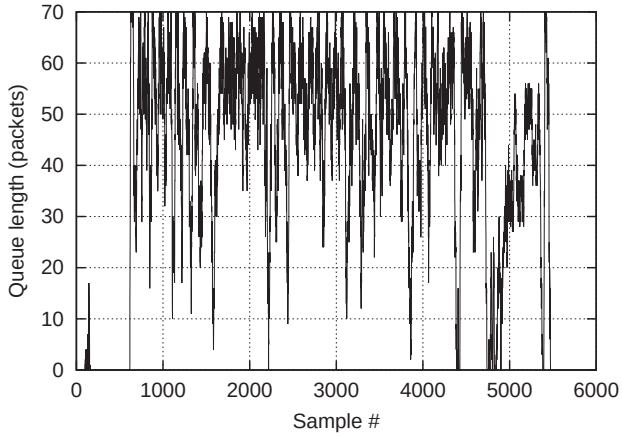
Network provisioning is the key aspect impacting the achieved quality that DiffServ is capable of offering. A proper allocation of resources can give high network utilization and provide good quality of service for the users.

We wanted to see how well the borrowing mechanism in CBQ could help to increase the network utilization in a badly provisioned case. There is usually a gap between theoretical models and real implementations due to limitations in software and hardware.[Cho04] Therefore, we also need to evaluate the behavior caused by the implementation issues.

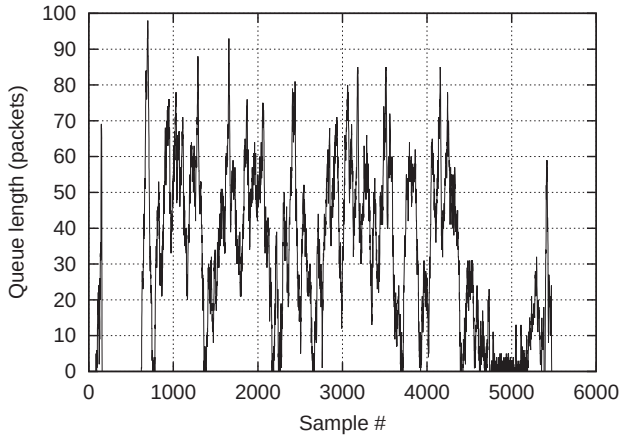
In the first scenario, the CBQ mechanism in the routers is configured with the link sharing hierarchy presented in Figure 8.18 without borrowing bandwidth between classes. A leaf class can send until it has used its round-robin allocation or it becomes overlimit. In the second scenario, we enable the borrowing



(a) Gold



(b) Silver



(c) Bronze

Figure 8.16: Queue length at the bottle neck link for different classes

mechanism up to the top class. Borrowing helps in the resource provisioning, as we do not need to know the exact bandwidth allocation beforehand. When borrowing is enabled, a class can keep on sending in an overlimit situation if

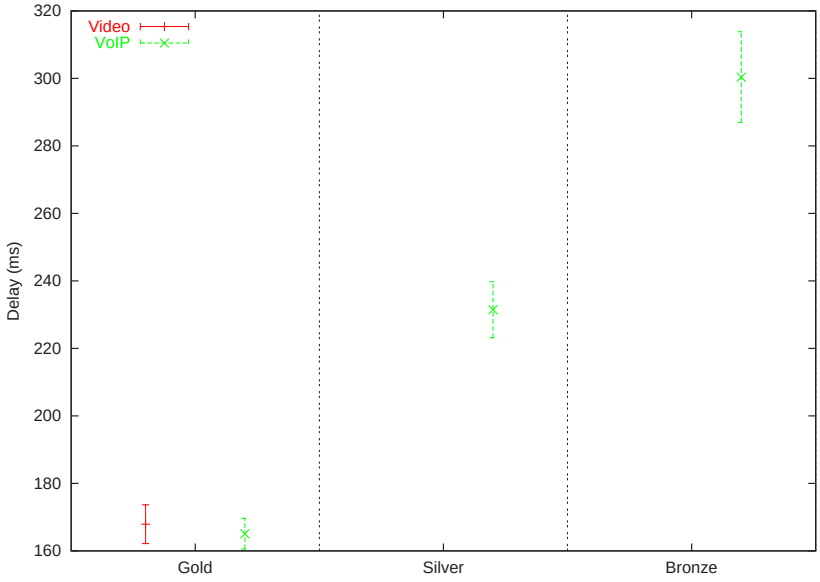


Figure 8.17: The delay and delay variation for the real-time applications

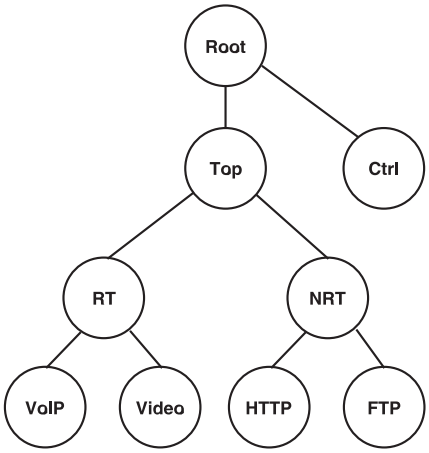


Figure 8.18: CBQ link sharing hierarchy

Table 8.11: Provisioning for the measurement (Mbps)

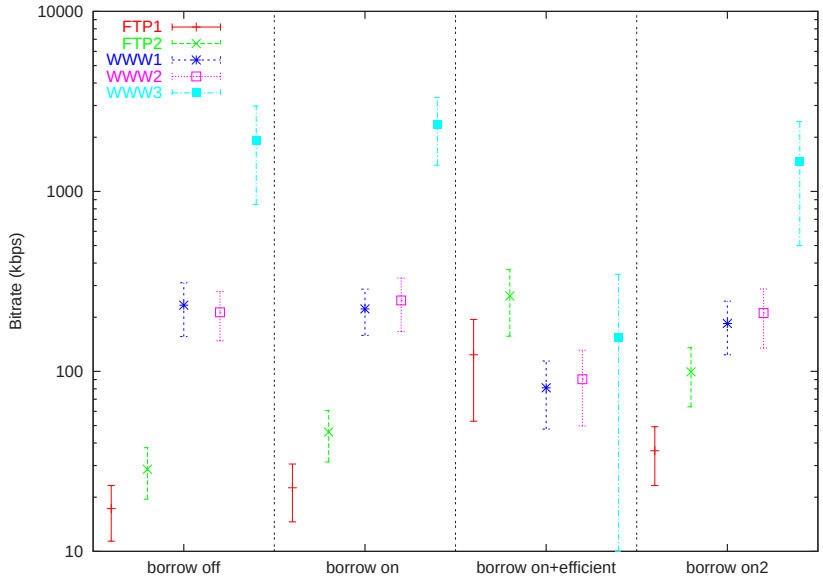
	VoIP	**Video**	**HTTP**	**FTP**
Borrow off	3.0	3.0	2.8	1.0
Borrow on	3.0	3.0	2.8	1.0
Efficient	3.0	3.0	2.8	1.0
Borrow on2	3.0	3.0	1.9	1.9

Table 8.12: Throughput for traffic sources (kbps)

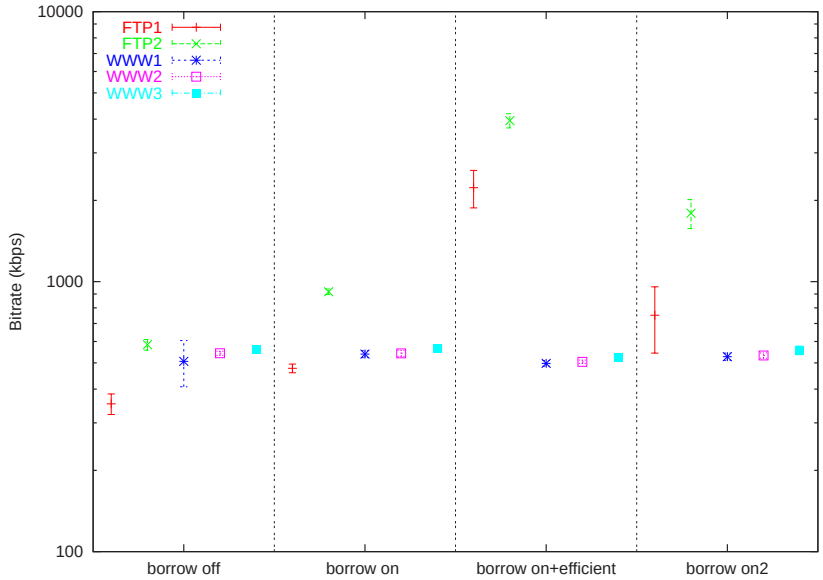
	Borrow off	**Borrow on**	**Borrow on+efficient**	**borrow on2**
FTP1	352.9	477.6	2228.2	750.4
FTP2	584.1	917.6	3946.8	1792.8
WWW1	506.5	539.1	498.0	527.4
WWW2	542.7	542.2	504.9	532.6
WWW3	558.2	565.6	523.5	556.8
VoIP	1744.0	1744.0	1726.4	1744.0
Video	130	130	130	130
Total	4418.4	4916.1	9557.8	6034.0

there are excess resources in the parent class and it has not consumed its round robin allocation. With the *efficient* parameter, the ALTQ/CBQ can make the borrowable classes work-conserving. In that case, the scheduler is capable of sending as long as it has backlogged packets even if not all classes are eligible to send. The measurements were made with two different provisioning presented in table 8.11. The real-time (RT) class is intentionally over-provisioned by a factor about three leading to a high under-provisioning in the non-real-time (NRT) class. The queue management is handled in the same way as in section 8.1.

Figure 8.19 gives the per connection and per client throughput values with different borrowing scenarios. When borrowing is disabled, the long lasting FTP connections are getting their share of the bandwidth but not more as can be expected. The WWW connections make a bad utilization of the resources due their bursty nature. During the idle times, the available resources can not be used in other classes. In addition the excess resources in the real-time class are simply wasted. When borrowing is enabled, the throughput of the WWW connections stays almost at the same level as without borrowing indicating saturation. This means that WWW-clients are not able to create more traffic due to idle times between page fetches. However, the FTP connections are getting a lower throughput than could be expected. This can be explained by how the borrowing mechanism in the ALTQ/CBQ has been implemented. When the class is overlimit, it is suspended. The suspension time (i.e. the time during which the class can not borrow) depends on the class bandwidth share:



(a) Connection throughput



(b) Aggregate throughput

Figure 8.19: Connection and aggregate throughput in different borrowing scenarios

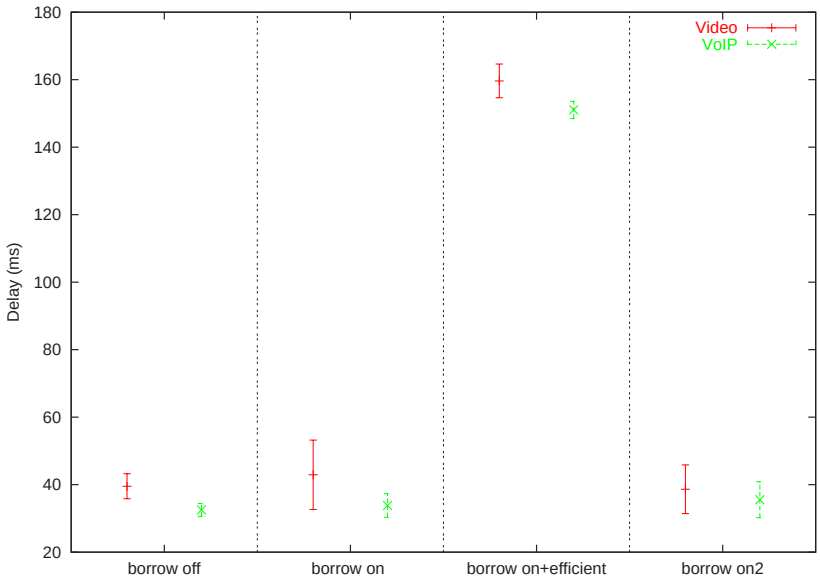


Figure 8.20: The delay and delay variation for the real-time applications

a smaller class gets a longer suspension time. This behavior favors large classes during the borrowing procedure and lowers the throughput in small classes. To minimize the effect of the suspension time the measurements were repeated with the *efficient* option enabled and with equal class bandwidths for FTP and WWW without the *efficient* option. With the *efficient* option, the throughput of the FTP connections increases significantly (Figure 8.19). This, however, lowers the throughput of the WWW connections and predictability of FTP connections. With the equal class bandwidth shares for FTP and WWW traffic the FTP connections can make a better use of the excess capacity.

The aggregated throughput for all traffic sources is shown in Table 8.12. We can see that the total link utilization varies a lot between different borrowing scenarios. Only in the *efficient* scenario the link can be totally utilized as the total throughput for TCP and UDP traffic is almost 10 Mbps. Without borrowing the link utilization is as low as 46 %.

Figure 8.20 shows the delay and delay variation for the real-time applications. We can see that borrowing increases the jitter in all scenarios. This variation in delay can be compensated by using a playback buffer at the end terminal. However, in the work-conserving scenario (third column) packet delays of the order of 150 ms combined with 1 % packet loss are degrading the quality of applications with hard real-time requirements below the operational level.

The CBQ borrowing mechanism in action is shown in Figure 8.21 that presents the number of borrows for FTP and WWW during the measurement period.

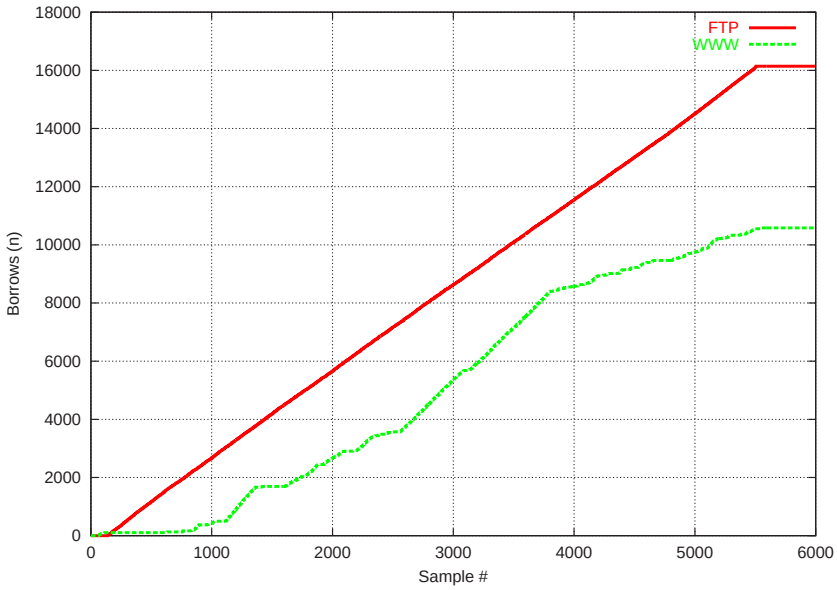


Figure 8.21: The number of borrows for FTP and WWW classes

As the real-time class was highly over provisioned there are no borrows in VoIP or video class. The number of borrows for FTP is linear over the time and is clearly limited by some mechanism. ALTQ includes a mechanism to prevent a class from borrowing too much. When a class is overlimit, it is suspended. The suspension time is calculated using the kernel timer and therefore the resolution of the kernel clock is relevant[4].

ALTQ/CBQ has many parameters that affect on the CBQ operation. *Average packet size* is a parameter that has an effect on the achieved throughput accuracy. If the packet size is set too large, a class in not able to achieve its target rate. On the other hand, if the average packet size is too small, a class can send more traffic that is has been contracted. Usually it is safe to set the average packet size to the size of the MTU as we did in our measurements. ALTQ/CBQ has also *minburst* and*maxburst* parameters that can be used to set the size of a burst. In general, ALTQ/CBQ is parameter sensitive and thus difficult to tune for optimal performance.

The kernel timer plays a crucial role in the ALTQ/CBQ. This internal clock defines the minimum time between interrupts and therefore affects the minimum suspension time. The higher the frequency of the internal clock is, the smaller the minimum suspension time can be. High clock rate can cause too much system overhead and therefore the value should not be set too high. The default value for kernel timer in FreeBSD is 100 Hz. In our experiments we

[4]The effect of kernel timer was explained in more detail in Section 5.2.2.3

Table 8.13: Provisioning of network resources (Mbps)

	Control	**RT**	**HTTP**	**FTP**
BE	0.2	3.0	2.0	4.8
Same	0.2	3.0	2.0	4.8
RT	0.2	3.0	2.0	4.8
Control	0.7	2.5	2.0	4.8

used a 1000 Hz internal clock, which still seems to be too low for a proper borrow operation in ALTQ. Therefore, a value higher than 1000 Hz for the kernel timer should be considered to be set when using an efficient PC.

8.4 Symmetry of differentiation

In a differentiated services network the return path of a TCP connection can be symmetric or asymmetric. The selection of the right return path has influence on the TCP performance as the delay and loss of acknowledgement (ACKs) packages will cause TCP to slow down the transfer speed. In this experiment, we study the effect of delivering data and ACKs in a different class.

To get a better understanding of the relation of ACKs to the throughput we performed measurements that transferred ACKs in:

1. Lowest quality class (BE)
2. Same class as data packets
3. Real-time (RT) traffic class
4. Control traffic class

Table 8.13 presents the network resource provisioning used in these measurements. It should be noted that when delivering ACKs in the control class some resources are moved from the RT class to the control class. Borrowing is enabled in all cases up to the top level (Figure 8.18).

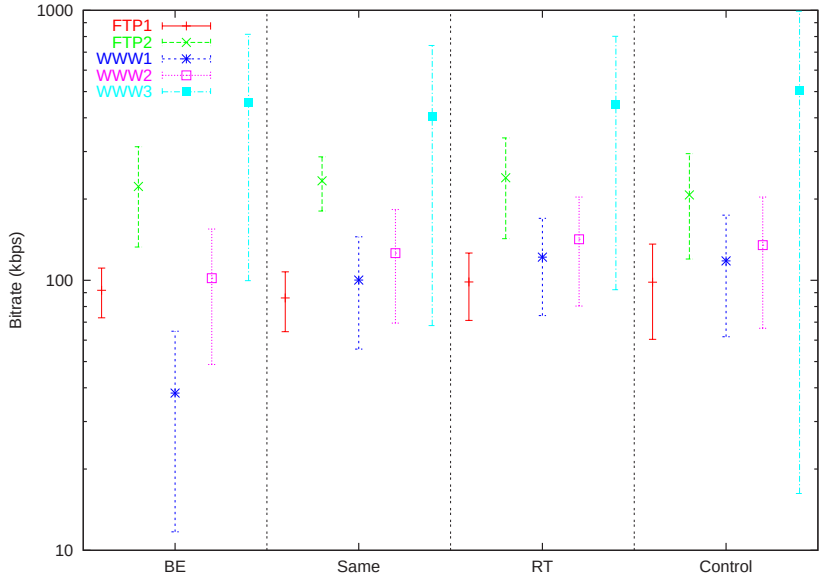
The traffic load to the return path was generated with Adtech AX/4000 that emulated the traffic patterns from the HTTP and FTP sources.

Figure 8.22 shows the connection and aggregate throughput for TCP connections. The most notable effect is the degradation of the throughput for the

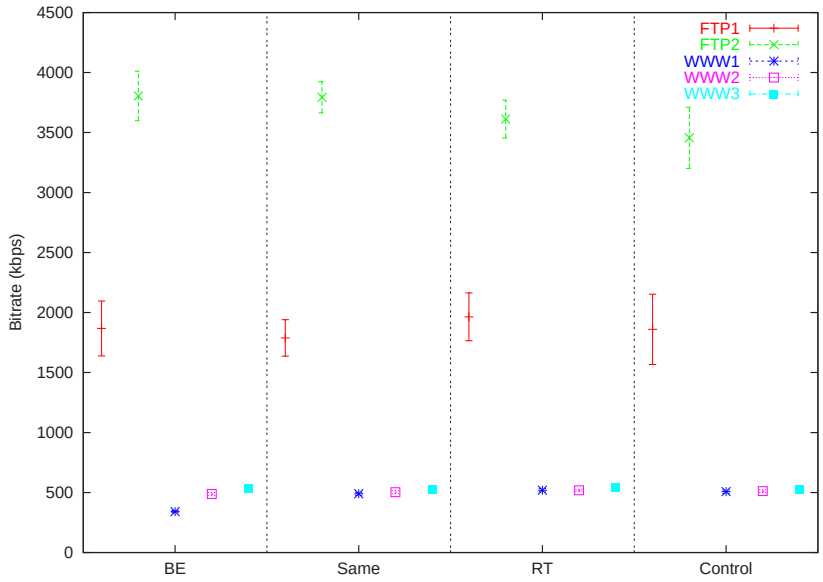
www1 client when delivering ACKs in the BE (FTP) class. The upstream congestion causes packet loss and increased delay that is most notable in the lowest quality class. When delivering ACKs in the other than BE class the differences are minimal. This is in contradiction with the simulations performed earlier on this matter [KS99]. These simulations showed that the appropriate choice of the ACK class has a big influence on the throughput. In our measurements this was however not true. The increased forward path congestion lowered the achieved throughput and the selection of an ACK class did not seem to have big importance. The different results between the simulations might be caused by the different traffic mix and set up. In the simulations, only FTP sources were modeled whereas we used a much wider variety of traffic. Also in the simulations, the FTP sources created the congestion to the network whereas we used unresponsive UDP flows at the return path.

Figure 8.23 shows the effect of ACKs to delay for the real time traffic. For VoIP traffic both communication paths (from a→b and b→a) are shown. Delay in VoIP from user a to b is about 10 ms higher in all cases. In addition, the jitter is higher in every case from a to b. Especially high (about 50 ms) the delay is when the ACKs are delivered in the RT class. This is not surprising as ACKs increase the utilization within the RT class that is reflected as an increased delay.

The delay in the return path for different traffic classes when transferring ACKs in the same traffic class is shown in Figure 8.24. We can see that the delay behavior is quite different between traffic classes. The most uniform delay behavior can be found from the highest priority class (RT class) where the average is delay is about 50 ms. The average delay in the WWW class is about 90 ms but due to busty traffic in this class the variation in delay values is high. In the lowest quality class (FTP class), the average delay is about 220 ms.



(a) Connection throughput



(b) Aggregate throughput

Figure 8.22: Connection and aggregate throughputs in case of asymmetric paths

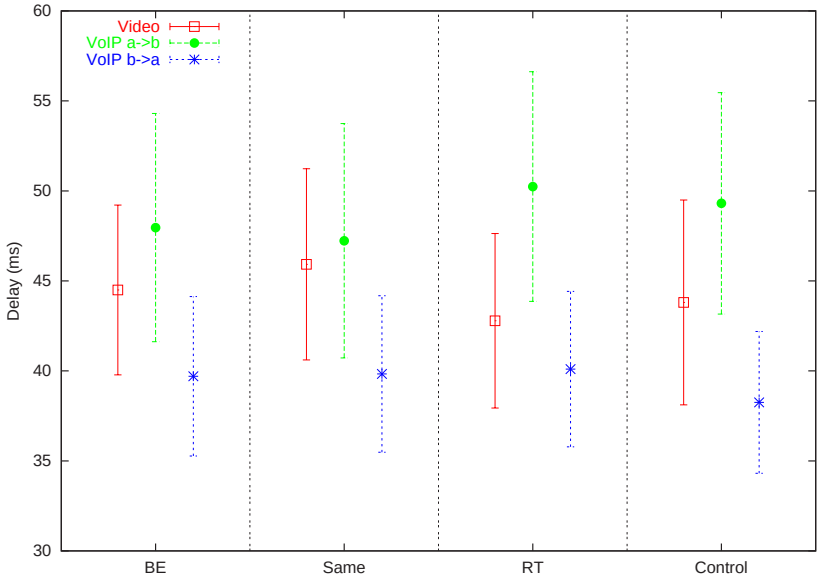
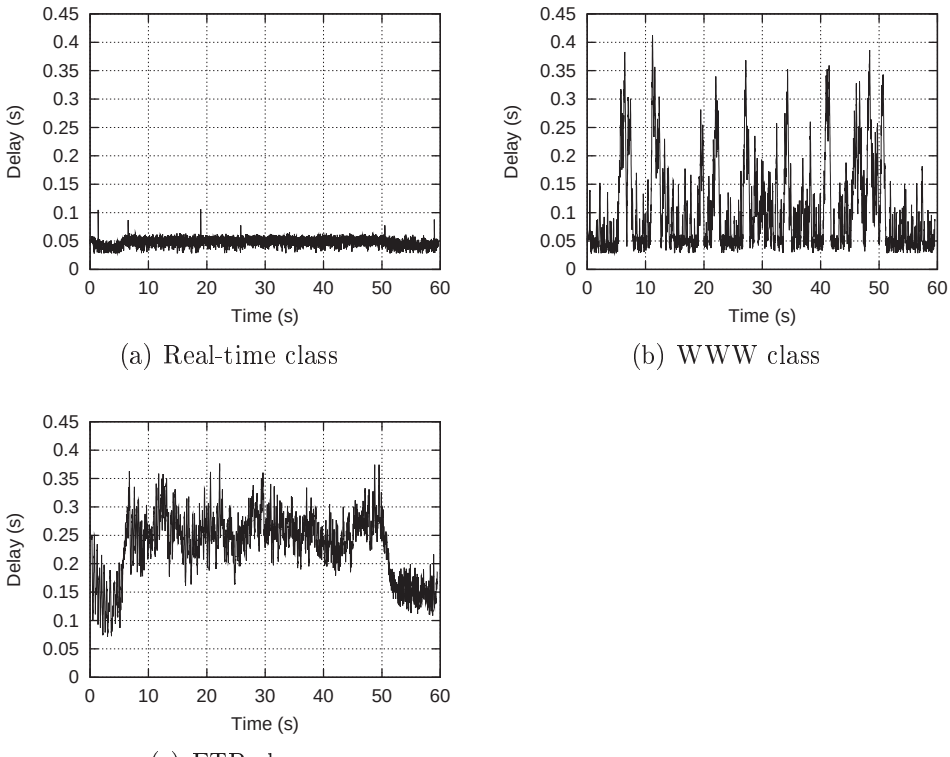


Figure 8.23: Video and VoIP packet delay in case of asymmetric paths



(a) Real-time class

(b) WWW class

(c) FTP class

Figure 8.24: Delay in traffic classes for return path

Chapter 9

Conclusions and discussion

In this work network measurements were carried out in an isolated environment. The testbed included DiffServ capable ALTQ routers and several traffic generators that were used to create traffic with different characteristics to the network. The traffic mix in the network consisted of VoIP, video, WWW, FTP, and background traffic.

The following main aspects were studied in the measurements: the level of differentiation, differentiation principle, differentiation aspects, and the symmetry of differentiation. An objective was to find the correct ways of doing the differentiation and to see the problems that reside in the actual implementations. Another important objective was to compare the measurement results to the simulation results conducted earlier in the same area.

The measurements showed that the differentiation has to be done on the application basis. Applications with different characteristics interfere with each other and they need to be separated. The separation between UDP and TCP traffic is essential if the network is not highly over-provisioned. The popular olympic services model that does not take traffic characteristics into consideration is badly suited for traffic differentiation.

Two traffic classes is the minimum that is needed to provide decent real-time communication in a congested network. In that case, elastic data traffic transferred over TCP need to be separated from the UDP traffic with real-time requirements. When differentiation is added between short TCP flows and long TCP flows, the packet loss can be lowered and throughput increased. The separation of VoIP and video traffic did not seem to have a big impact.

As was expected, some limitations and problems were found from the implementation. The biggest problems with ALTQ/CBQ were dealing with the borrowing feature. Not all excess bandwidth could be utilized and thus some resources were simply wasted in an overlimit situation. The theoretical aspect of CBQ is good, but in practice, it appears to be too complex and hard to implement. ALTQ/CBQ relies too much on kernel timers and that is a major flaw in the implementation. We increased the resolution of the kernel in the measurements from the default value of 100 Hz to 1000 Hz to obtain a better precision. Maybe, a kernel with 10 kHz timer would have performed better. It should be studied how high the resolution of the kernel can be increased without noticeable overhead to the system. The borrowing mechanism in ALTQ/CBQ should be rewritten in order to make it perform well. ALTQ/CBQ is also quite parameter sensitive, which makes it difficult to fine-tune for all scenarios.

The symmetry of differentiation measurements showed that the selection of the ACK class was not very significant for the achieved throughput. TCP appeared to be quite robust to changes in the return path. The most noticeable effect was the degradation of the throughput when delivering ACKs in the best effort class. In that case, the upstream congestion caused packet delays and loss that was most noticeable in the lowest quality class.

The measurements verified the simulation results [Luo00][Ant03] and supported the idea that the traffic characteristics have to be taken into consideration, in order to satisfy the application requirements. The measurements showed also the importance of the resource allocation. The provisioning of the traffic classes can not be static as sometimes it is very difficult to estimate the traffic volume. The amount of traffic in the network usually also varies quite a lot, which can lead to a low utilization. Therefore, a dynamic way of provisioning the network resources is essential. CBQ tries to achieve this by providing the borrowing feature.

The problems in ALTQ/CBQ dealing with adaptive resource allocation raises interest on adaptive scheduling[1]. A very promising adaptive scheduler is the delay bounded HPD which has been shown to achieve the targeted provisioning goal well in the simulations [AL03][AL04]. Delay bounded HPD measures both short and long-term queueing delays for adapting the resource allocation. HPD is a simple algorithm compared to CBQ and it can be implemented quite

[1]The basic idea of adapting scheduling is to dynamically adjust the class resources depending on traffic conditions

easily.

Bibliography

[AKM04] Guido Appenzeller, Isaac Keslassy, and Nick Mckeown. Sizing router buffers. In *Proceedings of ACM SIGCOMM 2004*, 2004.

[AL03] Johanna Antila and Marko Luoma. Scheduling and quality differentiation in differentiated services. In *Proceedings of MIPS 2003*, pages 119–130, November 2003.

[AL04] Johanna Antila and Marko Luoma. Adaptive scheduling for improved quality differentiation. In *Proceedings of MIPS 2004*, pages 143–152, 2004.

[Alm99] Werner Almesberger. Linux network traffic control - implementation overview. *White paper*, April 1999.

[Alm02] Werner Almesberger. Linux traffic control - next generation. *11th International Linux System Technology Conference*, October 2002.

[Ant03] Johanna Antila. Scheduling and quality differentiation in differentiated services. Master's thesis, Helsinki University of Technology, April 2003.

[BBC+98] Steven Blake, David Black, Mark Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. An architecture for differentiated service. Technical Report RFC 2475, IETF, December 1998.

[BCC+98] Bob Braden, David Clark, Jon Crowcroft, Bruce Davie, Steve Deering, Deborah Estrin, Sally Floyd, Van Jacobson, Greg Minshall, Craig Partridge, Larry Peterson, K. K. Ramakrishnan, Scott Shenker, John Wroclawski, and Lixia Zhang. Recommendations on queue management and congestion avoidance in the internet. Technical Report RFC 2309, IETF, April 1998.

[BCF00] Scott Brim, Brian Carpenter, and Francois Le Faucheur. Per hop behavior identification codes. Technical Report RFC 2836, IETF, May 2000.

[BCS94] Bob Braden, David Clark, and Scott Schenker. Integrated services in the internet architecture: an overview. Technical Report RFC 1633, IETF, June 1994.

[BLS01] Albert Banchs, Olga Leon, and Sebastian Sallent. The olympic service model: Issues and arhcitecture. In *Quality of Future Internet Services*, volume 2156, pages 24–26, September 2001.

[BSM04] Dhiman Barman, Georgios Smaragdakis, and Ibrahim Matta. The Effect of Router Buffer Size on HighSpeed TCP Performance. In *IEEE Globecom 2004 - Global Internet and Next Generation Networks*, November 2004.

[BZB+97] Bob Braden, Lixia Zhang, Steve Berson, Shai Herzog, and Sugih Jamin. Resource reservation protocol (rsvp) – version 1 functional specification. Technical Report RFC 2205, IETF, September 1997.

[Cho99] Kenjiro Cho. Managing traffic with ALTQ. In *Proceedings of USENIX 1999 Annual Technical Conference*. USENIX, June 1999.

[Cho01] Kenjiro Cho. *The Design and Implementation of the ALTQ Traffic Management System*. PhD thesis, Keio University, January 2001.

[Cho04] Kenjiro Cho. Fitting theory into reality in the altq case. In *Proceedings of ASIA BSD conference*. USENIX, March 2004.

[FJ93] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *ACM/IEEE Transactions on Networking*, August 1993.

[FJ95] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, 1995.

[FR00] Frank H.P. Fitzek and Martin Reisslein. Mpeg-4 and h.263 video traces for network performance evaluation. Technical report, Technical University Berlin, October 2000.

[FRC98] Anja Feldmann, Jennifer Rexford, and Ramón Cáceres. Efficient policies for carrying Web traffic over flow-switched networks. *IEEE/ACM Transactions on Networking*, pages 673–685, 1998.

[GM01] Liang Guo and Ibrahim Matta. The war between mice and elephants. Technical Report 2001-005, Boston University, Computer Science Department, May 2001.

[HBWW99] Juha Heinänen, Fred Baker, Walter Weiss, and John Wroclawski. Assured forwarding phb group. Technical Report RFC 2597, IETF, June 1999.

[Inc01] Spirent Communications Inc. Smartvoipqos user guide, August 2001.

[IT94] ITU-T. Recommendation e.800 - terms and definitions related to quality of service and network performance including dependability, August 1994.

[JNP99] Van Jacobson, Kathleen Nichols, and Kedarnath Poduri. An expedited forwarding phb. Technical Report RFC 2598, IETF, June 1999.

[KS99] Stefan Köhler and Uwe Schäfer. Performance comparison of different class-and-drop treatment of data and acknowledgements in diffserv ip networks. Technical Report 237, University of Würzburg, Institute of Computer Science, August 1999.

[LA04] Marko Luoma and Johanna Antila. Differentiation of the internet traffic. In *Proceedings of ICN 2004*, February 2004.

[LPY98] Marko Luoma, Markus Peuhkuri, and Tomi Yletyinen. Quality of service for ip voice services - is it necessary? In *Internet Routing and Quality of Service*, pages 242–253, November 1998.

[Luo00] Marko Luoma. Simulation studies of differentiated services networks. *Licentiate thesis*, Helsinki University of Technology, 2000.

[Luo03] Marko Luoma. S-38.180 quality of service in internet. Lecture slides, 2003. Available at http://www.netlab.hut.fi/opetus/s38180/s03/slides/.

[LY99] Jörg Liebeherr and Erhan Yilmaz. Workconserving vs. nonworkconserving packet scheduling: An issue revisited. In *Proceedings of IEEE/IFIP International Workshop on Quality of Service (IWQOS)*, pages 1484–1492. USENIX, May 1999.

[Mil85] David Mills. Network time protocol (ntp). Technical Report RFC 958, IETF, September 1985.

[Mil92] David Mills. Network time protocol (version 3). Technical Report RFC 1305, IETF, March 1992.

[Pap04] Panagiotis Papadimitriou. Linux traffic control essentials, tcng overview, study of a token bucket scenario. WWW, June 2004.

[Peu02] Markus Peuhkuri. Internet traffic measurements – aims, methodology, and discoveries. *Licentiate thesis*, Helsinki University of Technology, 2002.

[PF95] Vern Paxson and Sally Floyd. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, pages 226–244, 1995.

[RG99] Fulvio Rizzo and Panos Gevros. Operational and performance issues of a cbq router. *ACM Computer Communication Review*, pages 47–58, October 1999.

[Riz97] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.

[Riz98] Luigi Rizzo. Dummynet and forward error correction. *In Proceedings of Freenix 98*, June 1998.

[RVC01] Eric C. Rosen, Arun Viswanathan, and Ross Callon. Multiprotocol label switching architecture. Technical Report RFC 3031, IETF, January 2001.

[She95] Scott Shenker. Fundamental design issues for the future internet. *IEEE Journal on Selected Areas in Communication*, 13:1176–1188, September 1995.

[SV95] M. Shreedhar and George Varghese. Efficient fair queueing using deficit round robin. In *Proceedings of SIGCOMM 1995*, pages 231–242, 1995.

[TMW97] K Thompson, G.J Miller, and R. Wilder. Wide-area internet traffic patterns and characteristics. *IEEE/ACM Transactions on Networking*, pages 10–23, 1997.

[TNC+01] Fouad Tobagi, Waël Noureddine, Benjamin Chen, Athina Markopoulou, Chuck Fraleigh, Mansour Karam, Jose-Miguel Pulido, and Jun ichi Kimura. Service differentiation in the Internet to support multimedia traffic. *Lecture Notes in Computer Science*, 2170:381+, 2001.

[VS94] Curtis Villamizar and Cheng Song. High performance tcp in ansnet. *SIGCOMM Comput. Commun. Rev.*, (5):45–60, 1994.

[Wan01] Zheng Wang. *Architectures and Mechanisms for Quality of Service*. Morgan Kaufmann Publishers, 2001.