University of Bonn,

Department of Computer Science IV

Management of Layered Variable Bitrate Multimedia Streams Over DiffServ with A Priori Knowledge

by
Thomas Dreibholz
(Dreibholz@bigfoot.com)
Student-ID #1093879

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Diplom-Informatiker

Bonn, Germany
February, 2001

Thomas Dreibholz
Molbachweg 7
51674 Wiehl-Forst
Germany
mailto:Dreibholz@bigfoot.com
http://www.bigfoot.com/~dreibholz

Gutachter

1. Prof. Dr. Peter Martini, Abteilung IV
2. Prof. Dr. Armin B. Cremers, Abteilung III

Erklärung

Hiermit versichere ich, daß ich die vorliegende Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Bonn, den 20. Februar 2001

Contents

1 Introduction **1**

2 Basics **3**
2.1 Network and Internet Basics . . . 3
2.1.1 Reference Models . . . 3
2.1.2 The Internet Protocols IPv4 and IPv6 . . . 4
2.1.3 The Transport Protocols UDP and TCP . . . 6
2.1.4 The Control Message Protocol ICMP . . . 6
2.1.5 The RTP Protocol . . . 7
2.2 Quality of Service Basics . . . 9
2.2.1 QoS Requirements . . . 9
2.2.2 IntServ . . . 9
2.2.3 DiffServ . . . 10
2.2.4 Traffic Shaping . . . 11
2.3 Traffic Description and Traffic Models . . . 12
2.3.1 Traffic Constraint Function and Empirical Envelope . . . 13
2.3.2 Burstiness . . . 14
2.3.3 Traffic Models . . . 15
2.4 Bandwidth Remapping . . . 16
2.5 Resource Management . . . 17
2.5.1 Utility Functions . . . 18
2.5.2 Composition of Utility Functions . . . 20
2.5.3 Resource Management Definitions . . . 23
2.5.4 QoS Optimization Algorithms . . . 25
2.6 Media Types . . . 27
2.6.1 MPEG-1 Video . . . 27
2.6.2 MPEG-2 Video . . . 31
2.6.3 H.263 Video . . . 32
2.6.4 MP3 Audio . . . 32
2.7 Summary . . . 34

3 The CORAL Project and the Goals of This Work **35**
3.1 The CORAL Concept . . . 35
3.2 RTP AUDIO - A CORAL Implementation Example . . . 37
3.3 The Goals of This Work . . . 38
3.4 Summary . . . 40

4 The A Priori Remapping Interval Calculation **41**
4.1 The Remapping Interval Algorithm Basics . . . 41
4.2 Layering of the Media Types . . . 42

4.3 Extension for Layered Transmission . . . 44
4.4 Extension for Scalable Media Types . . . 45
4.4.1 Frame Rate Scalability . . . 45
4.4.2 Frame Size Scalability . . . 46
4.5 Runtime Optimization . . . 47
4.6 Space Optimization . . . 47
4.7 Summary . . . 49

5 The A Priori Resource/Utilization List Calculation 51
5.1 Resource/Utilization Basics . . . 51
5.2 Resource/Utilization Points . . . 53
5.3 Resource/Utilization Lists . . . 55
5.4 Buffer Delay Translation of Bandwidths . . . 58
5.5 Summary . . . 59

6 The Bandwidth Management 61
6.1 Bandwidth Management Basics . . . 61
6.2 Stream Description Initialization . . . 62
6.2.1 Packet Headers and the Payload⇔Raw Translation . . . 62
6.2.2 Layer to DiffServ Class Mappings . . . 64
6.2.3 Buffering and the Resource/Utilization Lists . . . 66
6.2.4 The Sorting Value . . . 67
6.2.5 Parallelization . . . 68
6.3 Session Description Initialization . . . 68
6.4 The Bandwidth Mapping . . . 70
6.4.1 The Complete Remapping . . . 72
6.4.2 The Partial Remapping . . . 74
6.5 Comments on the Bandwidth Remapping . . . 75
6.6 Summary . . . 77

7 The Traces and the System 79
7.1 The Traces . . . 79
7.1.1 MPEG-1 and MPEG-2 . . . 79
7.1.2 H.263 . . . 80
7.1.3 MP3 . . . 80
7.1.4 The Remapping Intervals . . . 80
7.2 The Resource/Utilization Lists . . . 81
7.3 The System . . . 85
7.4 Summary . . . 88

8 Measurements and Evaluation 89
8.1 Buffering, Layering, Weighting and Scalability Simulations . . . 89
8.1.1 A Buffering Example . . . 89
8.1.2 A Weighting Example . . . 91
8.1.3 The Buffering, Layering and Weighting Comparision . . . 93
8.1.4 The Scalability Comparision . . . 99
8.2 Bandwidth and QoS Management Functionality . . . 104
8.2.1 A Network Quality Change Example . . . 104
8.2.2 A Session and Stream Priority Example . . . 107
8.3 A System Load Scenario . . . 108

8.4 Complete and Partial Remappings 110
8.5 Measurements on a Real Transport Scenario 114
8.6 Summary 116

9 Conclusions **119**

A Trace Statistics **121**

B Buffering Measurement Results **125**

C Scalability Measurement Results **137**

D Abbrevations Index **153**

Chapter 1

Introduction

Only about 12 years ago - in October 1988 - the number of hosts connected to the *Internet* reached 56,000 ([RFC 2235]). The common computer hardware of this time was an INTEL *80286*-based PC-compatible with 640 KBytes RAM and a monochrome graphics card, 20 MBytes hard-disk and a 1200 Baud modem. This was just sufficient to use electronic mail, file transfer, news, Telnet and Gopher - the WWW was not even invented in 1988. Multimedia-capable hardware was not affordable for most users and probably nobody expected the tremendous worldwide success the Internet would have during the next few years.

But since this time, lots of things have changed. Powerful and cheap CPUs, 2D/3D-accelerated graphics hardware, high-quality sound cards and high-speed networks have become available at quite low prices and the *Internet* growed to more than 100,000,000 hosts (93,047,785 in July 2000, [NW-IDS]). With the availability of powerful hardware and high network bandwidths, there is an increased demand for real-time multimedia network applications like audio on demand (AoD), video on demand (VoD), *Internet* radio and TV, multimedia conferences, tele-working and much more. In the next few years, these technologies will cause revolutionary changes of economic and social issues which will make the *Internet* one of the most important inventions in history - if it is not already one of them.

Basically, the *Internet* was designed to suppose a best effort service which does not give guarantees for bandwidth, end-to-end delay, loss rate and jitter (statistical variance of packet interarrival time). This service is sufficient for the elastic traffic (capable of adjusting bandwidth requirements to network congestion) of TCP-based protocols like electronic mail, news, file transfer or WWW. For multimedia traffic - which has got tight quality requirements for bandwidth, delay, loss rate and jitter - the best effort service is of course insufficient. For example, a delay of some seconds or a continously changing quality because of packet loss due to network congestion are not acceptable for video conferences. Therefore, additional steps for a guaranteed quality of service (QoS) are necessary.

To achieve QoS guarantees, two approaches have been developed and refined by the Internet Engineering Task Force ([IETF]): *Integrated Services* (IntServ) and *Differentiated Services* (DiffServ). The first one - IntServ - provides end-to-end bandwidth reservations on a per-flow basis, introduced by the Resource ReSerVation Protocol (RSVP, [RFC 2205]). All routers on the path from the sender to the receiver have to store each flow's reservation information and handle the packets appropriately. Therefore, the more flows have reservations, the more CPU power is required to manage the reservations. The second approach - DiffServ - fixes this so called scalability problem using a set of quality of service classes (DiffServ classes) having different priorities. The packets are mapped by the sender or a router at a network border to a DiffServ class by marking them; the expensive per-flow knowledge is not necessary anymore. But since DiffServ does not provide end-to-end reservation, QoS management is necessary here. This has leaded to the CORAL project.

The CORAL project (*CO*mmunication protocols for *R*eal-time *A*ccess to digital *L*ibraries, see also [AKM+00]) is a system for transmission of scalable multimedia streams from one server to several

clients. Reservations of bandwidth are supported using DiffServ. The available bandwidths of each DiffServ class are managed by a QoS manager, which maps bandwidth to streams using QoS descriptions of each stream. To be cost efficient, streams may be layered. That is, a low-resolution video can be transmitted over a high-priority but expensive class as *base layer*. Additional data to extend this base layer to high resolution can be transmitted over a low-priority but cheap class (e.g. best effort) as so called *enhancement layer*. In case of network congestion, the packets will be dropped appropriate to their priorities. Therefore, the important base layer will have a lower loss rate than the not so important extension layer.

For *constant bitrate* (CBR) traffic, it is only necessary to reserve bandwidth once - the requirements do not change. But for *variable bitrate* (VBR) traffic, it is not acceptable to reserve the peak rate, since e.g. in an MPEG video it is often 10 or more times larger than the average rate ([Gum98]). Therefore, a regular update of the bandwidth mappings is necessary. For already completely stored medias like video and audio files, it is obvious to do an a priori analyzation of their transport properties. This information can be used to do the remapping as efficient as possible. Especially, this includes the usage of cost-optimized buffering to reduce bandwidth requirements and therefore the transport cost.

The design, implementation and evaluation of an efficient solution to manage layered and scalable variable bitrate multimedia streams in the CORAL project, based on a priori analyzation of the medias, is the global goal of this work. It is structured as follows: First, chapter 2 gives an introduction into the basics of multimedia traffic and QoS. This includes traffic descriptions using traffic models, the cost-optimized, a priori calculation of bandwidth remapping intervals, DiffServ in IP networks and efficient mapping of bandwidth using utility functions to evaluate the effects of changes to the user satisfaction. Finally, some standard multimedia formats like MPEG video and MP3 audio are examined. A detailed description of the CORAL project and this work's goals is given in chapter 3. Chapter 4 develops an a priori algorithm for cost-optimized calculation of remapping intervals for layered streams. Next, chapter 5 covers the a priori calculation of resource/utilization lists. Further, chapter 6 introduces an efficient management system for multimedia streams using methods developed in previous chapters. Chapter 7 describes the implemented system, which is finally evaluated by simulations and real network measurements in chapter 8. The work closes with conclusions in chapter 9.

The complete LINUX-based implementation can be downloaded here:
http://www.bigfoot.com/~dreibholz/diplom/

Chapter 2

Basics

This chapter gives and introduction into the basics of this work. First, the network fundamentals are explained, followed by the basics of Quality of Service (QoS). Next, efficient description of variable bitrate traffic using the so called empirical envelope and its approximation by traffic models are introduced. Further, an algorithm for the a priori calculation of optimal bandwidth remapping intervals is presented. This is followed by resource management basics and an approximative QoS optimization algorithm. The chapter closes with an explaination of standard video and audio formats.

2.1 Network and Internet Basics

The transport of data over a network (e.g. file transfer) is a complex problem. Therefore, it is useful to divide it up into hierachical layers where each layer solves a specific problem (e.g. error correction or repetition of lost packets). In such a hierarchy, layer *n* uses defined services of layer *n-1* and provides defined services to layer *n+1*. The services and operations offered by a layer to the next one are specified in the so called *interface*. Layer *n* on station A communicates with layer *n* on station B using a defined *protocol*. The entities of the corresponding layers on each station communicating together are called *peers*. As long as there are no changes in the layers' interfaces, it is possible to replace one layer's implementation by another one (e.g. telephone line -> satellite connection) without any changes necessary in other layers.

2.1.1 Reference Models

The most important reference models for layered networks are the OSI Reference Model (see table 2.1) and the TCP/IP Reference Model (see table 2.2) of the Internet Engineering Task Force ([IETF]). Since the TCP/IP model, which is a simplification of the OSI model, describes the reality more accurately, only this model is shortly explained here. Details about both models can be found in [Tan96],

7	Application Layer
6	Presentation Layer
5	Session Layer
4	Transport Layer
3	Network Layer
2	Data Link Layer
1	Physical Layer

Table 2.1: The OSI Reference Model

Application Layer	e.g. HTTP, FTP, Telnet, SSH, NNTP, NFS, . . .
Transport Layer	e.g. UDP, TCP
Internet Layer	e.g. IPv4, IPv6
Host to Network Layer	e.g. Ethernet or FDDI

Table 2.2: The TCP/IP Reference Model

Length	Content
4 Bit	Version (4)
4 Bit	Internet Header Length
8 Bit	Type of Service (⇔ Tr. Class)
16 Bit	Total Length
16 Bit	Identification
3 Bit	Flags: unused/DF/MF
13 Bit	Fragment Offset
8 Bit	Time to Live (⇔ Hop Limit)
8 Bit	Protocol (⇔ Next Header)
16 Bit	Checksum (**Header** only!)
32 Bit	Source Address
32 Bit	Destination Address
variable	Options

Table 2.3: The IPv4 header

Length	Content
4 Bit	Version (6)
8 Bit	Traffic Class (⇔ TOS)
20 Bit	Flowlabel
16 Bit	Payload Length
8 Bit	Next Header (⇔ Protocol)
8 Bit	Hop Limit (⇔ Time to Live)
128 Bit	Source Address
128 Bit	Destination Address

Table 2.4: The IPv6 header

Length	Content
8 Bit	Next Header
8 Bit	Extension Header Length

Table 2.5: The IPv6 extension header

section 1.4. The TCP/IP model consists of four layers, which are:

1. **The Host-to-Network Layer**
   This layer contains the physical transport between two network nodes. Important protocols of this layer are for example Ethernet, FDDI and ATM.

2. **The Internet Layer**
   The transport from a source host to a destination host over several nodes in heterogeneous networks is done here. This contains routing between network nodes. The most important protocols of this layer are IPv4 (see [RFC 791]) and IPv6 (see [RFC 2460]), which are both described in section 2.1.2.

3. **The Transport Layer**
   Flow- and congestion control, error correction and repetition of lost packets are done in this layer. It contains connection-less protocols like UDP ([RFC 768]) and connection-oriented ones like TCP (see [RFC 761]). Both protocols are described in section 2.1.3.

4. **The Application Layer**
   Application-specific protocols can be found in this layer. Examples are file transfer (FTP, see [RFC 959]), WWW (HTTP, see [RFC 2068]), Telnet (see [RFC 764]) or electronic mail (SMTP, see [RFC 821]).

It should be denoted, that not all protocols exactly belong to one of this layers. For example the ICMP protocol (see section 2.1.4, [RFC 792] and [RFC 2464]) - which lays between internet and transport layer - or RTP (see section 2.1.5, [RFC 1889]) - which lays between transport and application layer.

2.1.2 The Internet Protocols IPv4 and IPv6

As mentioned above, the IPv4 ([RFC 791]) or the newer IPv6 ([RFC 2460]) protocol - simply called IP in the following text - are usually used for implementing the internet layer. IP ensures routing of packets from one station through connected networks of different standards (e.g. Ethernet, FDDI and ATM) to another station using unique IP addresses for each station. The IP header formats are shown in table 2.3 (IPv4) and table 2.4 (IPv6). The fields have the following functions:

Version is the IP version number: 4 for IPv4 and 6 for IPv6.

Source Address/Destination Address are the source and destination IP address of the packet. IPv4 has got an address length of 32 bits, which results in a theoretical maximum of 4,292,967,296 possible addresses. But the real value is much lower due to inefficient mapping. Since the *Internet* is growing at exponential rates, it will run out of addresses very soon. Therefore, IPv6 uses 128-bit addresses, so there is a theoretical maximum of

$$340,282,366,920,938,463,463,374,607,431,768,211,456$$

addresses. In reality, it allows even in the most inefficient mapping scenario more than 1,000 addresses per square meter of the earth's surface. In more realistic scenarios[1], there will be many trillions.
For text representation of IPv4 addresses, the dotted decimal notation is normally used (e.g. 131.220.88.99). But for IPv6, three new styles are defined in [RFC 1884]:

- The colon hexadecimal notation gives the address in eight 16-bit blocks. Examples: fe80:0000:0000:0000:260:97ff:fe68:24b5, 1080:0:0:0:800:700:6:15.
- The compressed colon hexadecimal notation replaces **one** group of zeros by '::'. Examples:
  fe80::260:97ff:fe68:24b5, 1080::800:700:6:15, ::1 (loopback), :: (unspecified)
- The notation for IPv4/IPv6 mixed environments: Six blocks of 16-bit in (compressed) colon hexadecimal notation and the remaining 4 bytes in dotted decimal notation. Examples:
  0:0:0:0:0:ffff:131.220.88.99, ::ffff:131.220.88.99, ::1.2.3.4, ::ffff:127.0.0.1

Hop Limit ⇔ Time to Live is a counter which is decreased by one on each router the packet is passing. If it has reached zero, the packet is discarded. This is necessary to avoid packets rotating in an infinite loop due to misconfigured routing tables. The hop limit is called *Time to Live* (TTL) in IPv4 because the original idea of this field was to give the packet's timeout in seconds.

Protocol ⇔ Next Header is the protocol number of the next upper layer, e.g. TCP or UDP (see section 2.1.3). Standard protocol numbers are specified in [RFC 1700]. In IPv6, an IPv6 header can be followed by an IPv6 extension header (see table 2.5). In this case, the *Next Header* field contains the number for an IPv6 extension header and the extension header's *Next Header* field contains the protocol number.

Internet Header Length/Total Length ⇔ Payload Length contain the IPv4 packet's header and total length (header + payload) and the IPv6 packet's payload length. Since the IPv6 packet header has a constant length of 40 bytes, it is only necessary to store the length of the payload. In IPv4, the header may contain additional options of variable size resulting in the necessity to store its length.

Checksum is a checksum to detect errors in the IPv4 header but **not** in the payload. Since it is necessary to calculate a new checksum everytime the header changes (TTL decreases on each router), this functionality has been removed for IPv6 resulting in faster routing.

Identification/Flags/Fragment Offset are used in IPv4 for fragmentation: If packets are too large[2] to be sent over a specific underlying network (e.g. larger then 1500 bytes in an Ethernet),

[1] See [RFC 1715] for details about address assignment efficiency.
[2] All nodes have to support at least an IP packet length of 576.

IP also does fragmentation, that is dividing the large packet into several smaller packets and joining them at the destination station. *Identification* is the same for all fragments of a packet. The *fragment offset* is the offset of the packet's payload inside the original packet, the *MF* (More Fragments) flag show whether or not the packet is fragmented. If the *DF* (Don't Fragment) flag is set, the packet may not be fragmented. In IPv6, fragmentation can only be done by the source station resulting in simpler routing. In case of fragmentation, IPv6 uses an extension header to store fragmentation information.

Traffic Class ⇔ Type of Service marks a packet to belong to a specific DiffServ class. A detailed description of this field[3] can be found in section 2.2.3.

Flowlabel can be used to mark a packet to belong to a specific flow. It is network-unique in conjunction with the source address. An equivalent field in IPv4 does not exist. See section 2.2.2 for details.

2.1.3 The Transport Protocols UDP and TCP

The unreliable User Datagram Protocol (UDP, see [RFC 768]) and the reliable Transmission Control Protocol (TCP, see [RFC 761] and [RFC 1323]) protocol are the most important protocols of the transport layer. Both contain endpoint identification using so called *port numbers*: The IP address identifies source and destination host, but the port number identifies the so called *socket*, that is the interface between an application and the transport layer.

UDP provides an unreliable, connection-less service comparable to sending and receiving raw IP packet plus the endpoint identification and a checksum for the complete UDP packet. If flow and congestion control, repetition of lost packets, etc. are necessary, it has to be realized in the application layer. The main usage for this protocol are multimedia transmissions, where fast transmission is more important than reliability. For example in an audio conference, lost or damaged packets causing a few noise are acceptable but it is not acceptable to wait for retransmission.

The reliable service provided by TCP is connection-oriented - it is necessary to establish a connection before data can be sent and to release it if the transmission is complete. This can be compared to a phone call where a connection is established by dialing a number and after transmission released by hanging up.

TCP is also stream-oriented, that is an application on station A writes a number of bytes to the TCP socket, which are packaged by TCP into one or more packets. The application itself has no knowledge about this packaging. Station B receives the packets and reconstructs the original byte sequence which is then given to the application via the socket. Since lost packets are requested again, the transmission is reliable. Further, TCP does flow and congestion control by reducing or increasing the bandwidth corresponding to the receiver's capabilities and network congestion. Such traffic is therefore called *elastic traffic*.

Applications of TCP are all kinds of reliable transmissions: File transfer (FTP, see [RFC 959]), WWW (HTTP, see [RFC 2068]), Telnet (RFC, see [RFC 764]) and many more. Since the multimedia system described in this thesis only uses UDP for transmission, a detailed description of TCP will not be given here. It can be found in [Tan96] or the RFCs mentioned above.

2.1.4 The Control Message Protocol ICMP

The Internet Control Message Protocol ICMP - more exactly ICMPv4 (see [RFC 792]) for IPv4 and ICMPv6 (see [RFC 2464]) for IPv6 - are used for various control purposes based on IPv4 and IPv6. Its main applications are:

[3] In RFCs before [RFC 2460], there was a 4-bit field called *Priority* instead of *Traffic Class* and the length of *Flowlabel* was 24 bits (4 bits more than now). This was changed to be compatible with the *Type of Service* (TOS) field of IPv4.

Length	Content
2 Bit	Version (2)
1 Bit	Padding
1 Bit	Extension
4 Bit	CSRC Count
1 Bit	Marker
7 Bit	Payload Type
16 Bit	Sequence Number
32 Bit	Time Stamp
32 Bit	SSRC
variabele	CSRC[0..16]

Table 2.6: The RTP header

- replying errors to a sender, e.g. a destination TCP or UDP port is invalid, errors in the IP header, an IP packet has reached the maximum number of hops, etc.,
- router management (see RFCs above and [RFC 2462] for details) and
- tests.

In this work, only the test functionality will be used. Since there is no difference between ICMPv4 and ICMPv6 for this functionality, no distinction between both protocols will be made in the following text. It will simply be called ICMP.

The test functionality consists of two message types: *echo request* and *echo reply*. A source sends an echo request to a destination which will send it back using an echo reply. The delay between sending the request and receiving the reply is the so called *round trip time*.

2.1.5 The RTP Protocol

The Real-time Transport Protocol (RTP, [RFC 1889]) is a framework for the unicast and multicast transport of multimedia data, usually based on UDP. It consists of two protocols:

- RTP for the transport of the payload data itself and
- RTCP (Real-time Transport Control Protocol) for transport of application-specific control information and reception quality feedback from receivers.

The RTP packet header format can be found in table 2.6. It consists of the following fields:

Version is the RTP protocol version: Currently it is 2.

Padding/Extension/Marker can be set to show, that the packet is padded to a specific size[4], has got a header extension or requires special handling by the receiver.

Payload Type is an identification for the payload data's type, e.g. MPEG video. A number of standard types are defined in [RFC 1890] (RTP profiles).

Sequence Number increases by one for each RTP packet sent. It can be used to preserve the packets' order and calculate the number of packets lost.

[4]Padding may be necessary for encryption. For example, a packet size at a multiple of 1024 bits can be required to run a certain encryption algorithm.

Time Stamp increases monotoneously and linearly with the system time. Packets belonging to a group (e.g. a video frame) may contain the same time stamp. It can be used to calculate the jitter (see definition below).

SSRC identifies the sender by its *synchronization source* (SSRC), a 32-bit value (see description below) .

CSRC may identify additional senders contributing to the packet's data. For example, a so called *mixer* can join several streams to one stream. In this case, the senders' SSRCs can be given here as *contributing sources* (CSRCs).

CSRC Count is the number of CSRCs or zero for none.

Further, some algorithms are defined in [RFC 1889] to calculate the *loss rate* (fraction of packets lost during transmission) and the *interarrival jitter* (statistical variance of the packet arrivals) from the packets' sequence numbers and time stamps. In the RTP specification, the jitter is defined as follows: Let R_n be the RTP time stamp and S_n the arrival time stamp of packet n). Then, the interarrival jitter is:

$$\text{Jitter}_{\text{New}} := \text{Jitter}_{\text{Old}} + \frac{1}{16} * \left| D_{i-1,i} - \text{Jitter}_{\text{Old}} \right|,$$

where

$$D_{i,j} := (R_j - R_i) - (S_j - S_i).$$

Of course, both timestamps have to be given in the same units, e.g. microseconds. In the following text, *jitter* always refers to this RTP definition of the interarrival jitter. Every media transmission (e.g. audio or video) gets its own *RTP session*. An RTP session consists of at least two members (e.g. a sender and a receiver or members of a conference both acting as sender and receiver). Each of them is identified by a session-unique SSRC (*synchronization source*) number - a random 32-bit value - and a globally unique CNAME (*canonical end-point identifier*) - the user and host name. For example, a user with CNAME "user@host.domain.xy" can be member of an audio session with *ssrc1* and member of a video session with *ssrc2*.

While RTP is used for payload transport only, the RTCP protocol is used for transmission of control information. Five different packet types are defined:

- Sender Reports (SR):
  Every sender regularly transmits its current NTP[5] timestamp (time in microseconds since January 01, 1970) and RTP timestamp, the amount of packets and bytes sent and finally receiver reports (see next type), if the sender is also a receiver (e.g video conference).

- Receiver Reports (RR):
  These reports are regularly sent from receivers to senders, containing loss rate, jitter, time stamp of the last received sender report and time between reception of the sender report and transmission of the current receiver report. Since RTP itself does not guarantee reliable transmission nor does it realize flow and congestion control, error correction, etc., this has to be implemented in protocol layers above RTP, based on these receiver reports.

- Source Description (SDES):
  This type contains various information about an RTP member. Several subtypes are defined,

[5]NTP denotes the Network Time Protocol. See [RFC 1305] for details.

the most important one is CNAME (see above). Others contain user name, mail address, phone number, location etc.. Application-specific data can be transmitted using the PRIV (private) subtype.

- BYE: This type is sent when a member leaves a session.
- APP: Application-specific data is sent using this type. Examples are control commands like changes of quality, media or position, sent from a receiver to a sender.

A detailed description of these types can be found in [RFC 1889].

2.2 Quality of Service Basics

As already mentioned in section 2.1.2, IP provides only a best effort service. For traffic which can adapt the data rate to the network's current capacities, this service is sufficient. But for multimedia traffic like video and audio conferences it is not.

2.2.1 QoS Requirements

For multimedia streams, guarantees for the four QoS (quality of service) requirements are necessary: *Bandwidth*, maximum *transfer delay*, maximum *loss rate* and maximum *jitter*.

1. **Bandwidth**
   Multimedia streams have got at least a minimum bandwidth requirement, e.g. this is the lowest possible resolution and frame rate of a video. Bandwidth reservation is necessary therefore.

2. **Maximum transfer delay**
   In phone calls for example, transfer delays of several seconds are inacceptable. Therefore, a given maximum delay should be ensured.

3. **Maximum loss rate**
   In a video transmission, a loss of e.g. 20% of the transmitted packets may be acceptable due to interpolation of the missing picture parts. The same loss for an audio transmission may e.g. cause inacceptable noise. An upper limit for the maximum loss rate is therefore necessary.

4. **Maximum jitter**
   For example in audio or video conferences having a very low delay requirement, buffering of the incoming packets is not possible. In this case, a jitter limit is required.
   Various definitions of the jitter are possible. One of them - the RTP protocol's definition - can be found in section 2.1.5.

There are two approaches to implement bandwidth reservations for the internet layer: *IntServ* (Integrated Services) and *DiffServ* (Differentiated Services).

2.2.2 IntServ

IntServ uses per-flow reservations using the Resource ReServation Protocol (RSVP, see [RFC 2205] for details). The streams are identified[6] by their IP addresses and port numbers (UDP or TCP). In

[6] It should be denoted here, that IntServ's (**internet layer**) identification requires information from the Internet layer (IP address) and the **transport layer** (UDP/TCP port numbers)! Therefore, an access from a lower layer to the upper layer's data is necessary. This conceptual problem can be avoided by using IPv6 with flowlabels.

Length	Content
6 bit	DiffServ Code Point (DSCP)
2 bit	Currently Unused (CU)

Table 2.7: The *Traffic Class*/*Type of Service* field

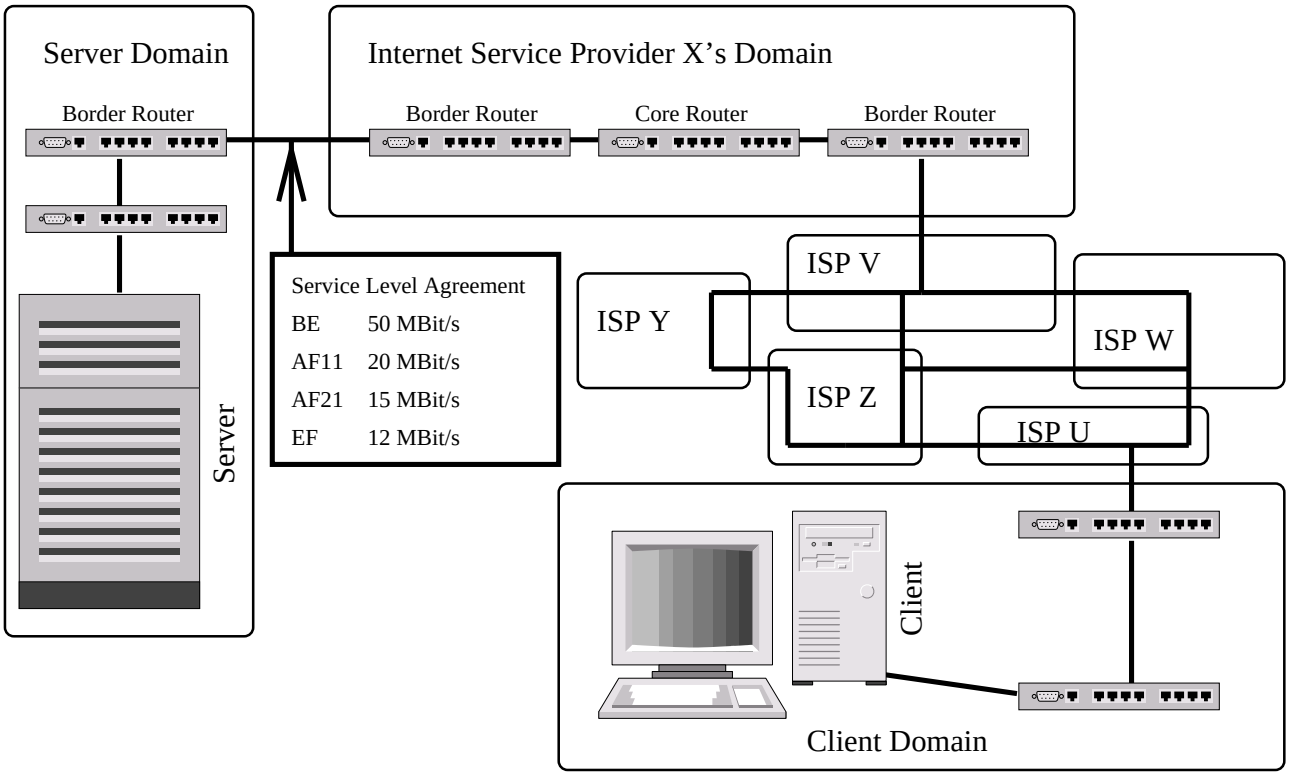


Figure 2.1: An example for DiffServ domains

IPv6, identifying streams is simplified by the flowlabel: Source address and flowlabel are network-unique.

This requires routers to store reservation information for each flow and causes IntServ's scalability problem: Especially in large networks with a large amount of streams using reservation, there are high CPU power and memory requirements for the routers to ensure sufficient speed. More details about this problem can be found in [RFC 2208].

2.2.3 DiffServ

The second approach - DiffServ - solves this problem by only reserving a small set of different *DiffServ classes*. Therefore, routers only have to store some class information. The streams' packets are marked - either by the sender itself or a router - to belong to one of the DiffServ classes and are handled by the router appropriately to their class. In the IP header, this is done using the *Type of Service* (IPv4, see table 2.3) or *Traffic Class* (IPv6, see table 2.4) field.

The allocation of the fields' bits, which is equal for both IP protocols, is shown in table 2.7. It consists of the 6-bit *DiffServ Code Point* (DSCP) which identifies the DiffServ class and two bits currently unused. A detailed description and codepoint definitions can be found in [RFC 2474].

The available classes and the traffic limits for each class are negotiated between the server's domain owner and the Internet Service Provider (ISP) in a *service level agreement* (SLA). An example for DiffServ domains can be found in figure 2.1. Every link between two DiffServ domains requires an SLA between both providers. A router inside a domain (*core router*) only has to support the

domain's classes. For *border routers* at a domain's edges, additional functionality may be required:

Classifiers determine the DiffServ class of a packet.

Meters measure the traffic of the different classes. It is required to verify whether or not a class exceeds the SLA's limits.

Markers map a packet to one of the classes by setting the *Traffic Class* or *Type of Service* field.

Shapers may delay packets by buffering them. See section 2.2.4 for details about traffic shaping.

Droppers drop packets exceeding the limits of the SLA.

As DiffServ classes, various implementations are possible. But the most common are *Expedited Forwarding* (EF) and *Assured Forwarding* (AF). They define *per-hop behaviours* (PHBs) for each router:

The Expedited Forwarding PHB

This PHB - also called *premium service* - is defined in [RFC 2598]. Its goal it to achieve assured-bandwidth, low-delay, low-loss and low-jitter services through DiffServ domains comparable to "virtual leased lines". This is done using only very small or even none queues in the routers and therefore requiring the sender to shape its traffic (see section 2.2.4 for details).

Streams using the EF class may not exceed the given rate limit, packets exceeding this limit will be dropped. EF traffic should be forwarded independent of all other traffic passing the node simultaneously in other classes.

The Assured Forwarding PHB

The AF PHB defined in [RFC 2597] consists of up to four classes (AF1, AF2, AF3 and AF4) having different priorities. The users may exceed the assured traffic limits, but in case of congestion, packets exceeding these limits will be dropped. Therefore, each AF class can have up to three *drop precedences*. Packets having higher drop precedences will be dropped at a higher probability. This results in up to twelve[7] AF classes: AFnm with $1 \leq n \leq 4$; $1 \leq m \leq 3$.

The forwarding behavior of an AF class has to be independent of other classes. Also, long-term congestion has to be avoided using an active queue management algorithm like RED (random early drop, [FJ93]). RED uses two smoothed congestion level thresholds. If the smoothed congestion is between level 1 and level 2, packets are dropped randomly using linearly increasing probability. Above level 2, all packets are dropped. The dropping probability is shown if figure 2.2. Due to its longer queues, the delay and jitter are usually higher than for the EF PHB.

2.2.4 Traffic Shaping

Most traffic is sent in bursts, e.g. a video at a frame rate of 10 frames/s will send a sequence of usually many packets every $\frac{1}{10}th$ second. Since the routers' queues are limited and especially for the EF class very small, they can be overflown by such a burst of packets. Therefore, it is necessary for the sender to smooth the packet rate by buffering the packets: For example, instead of sending 10 packets every $\frac{1}{10}th$ second, send 2 packets every $\frac{1}{50}th$ second. This can be implemented using a so called *leaky bucket* (see figure 2.3, left side). It has got an input buffer of size σ, large enough to store the whole burst and a constant output rate ρ. The canonical comparision to this is a real leaky bucket which

[7] More classes and drop precedences may be defined for local usage.

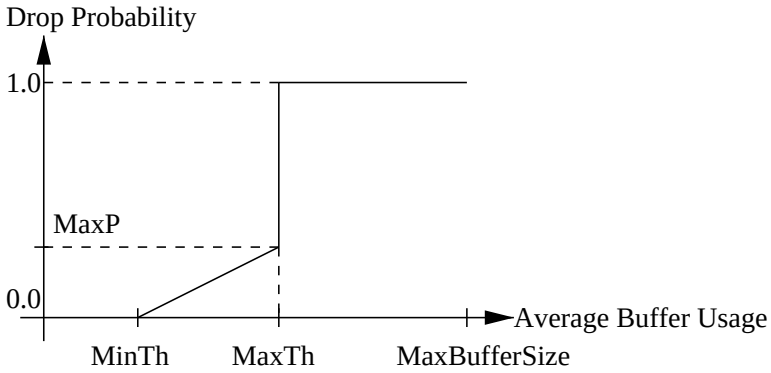


Figure 2.2: The RED dropping probability

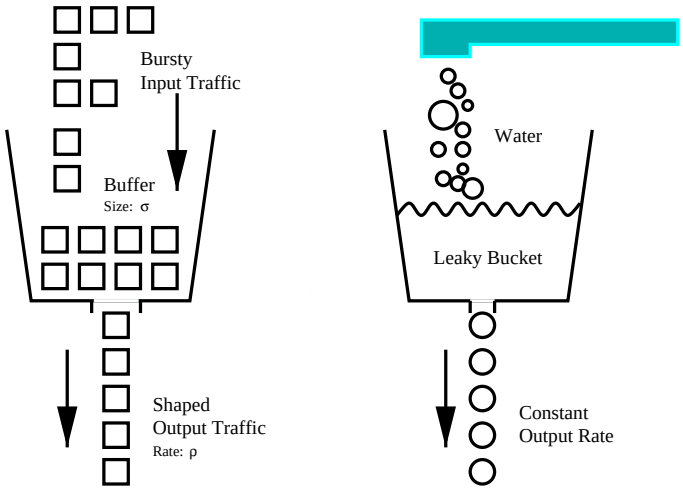


Figure 2.3: A leaky bucket and its canonical comparision

can be filled with water (see figure 2.3, right side). It can be filled with any rate but the output rate remains constant - as long as it does not become empty.

Multimedia streams have constraints for the maximum transfer delay. Therefore, it is necessary to *police* the stream's traffic running through a leaky bucket. That is, checking whether the delay introduced by the buffering is below a given *maximum buffer delay*. The delay constraint is violated, if

$$\text{BufferSize} > \text{MaxBufferDelay} * \text{OutputRate}. \tag{2.1}$$

This means, that if there are more bytes in the buffer than can be sent during the maximum allowed time, the delay constraint is violated. In this case, a *buffer flush* is required, that is emptying the whole buffer by dropping all contents inside.

2.3 Traffic Description and Traffic Models

Using constant bitrate (CBR) streams, it is only necessary to reserve a constant bandwidth once. For variable bitrate (VBR) streams, it would be possible to reserve the peak rate. Unfortunately, for most media types this is extremely inefficient. For example in MPEG-1 videos, a factor 10 and more between the long-term average rate and the peak rate is not unusual (see measurements in [Gum98], page 26). A more efficient solution for the traffic description is therefore necessary.

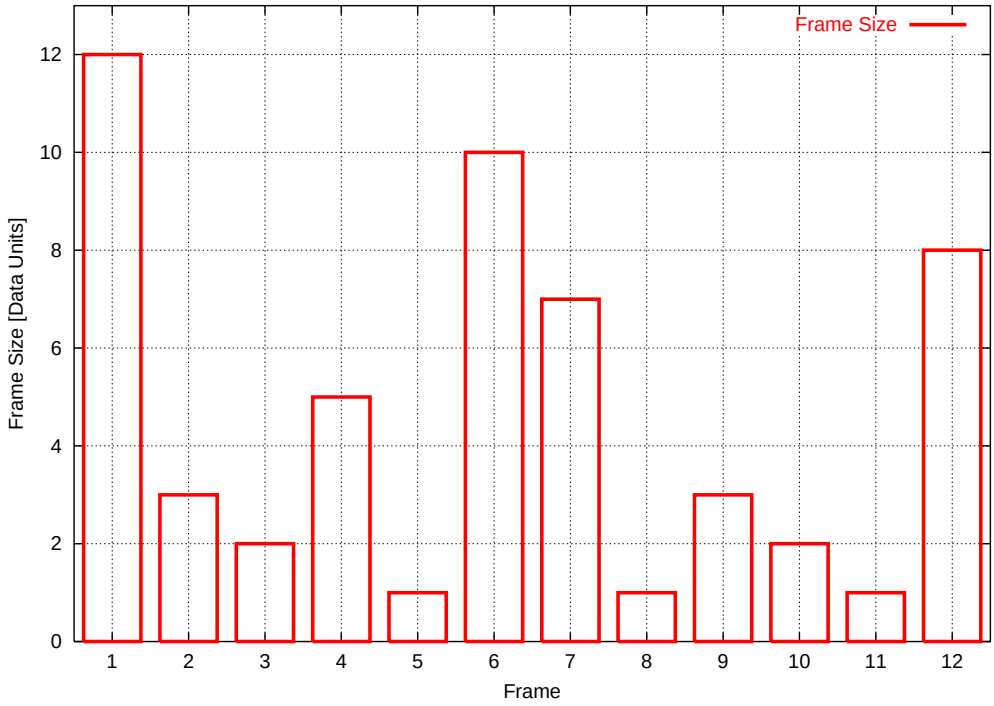


Figure 2.4: An example stream consisting of 12 frames

Delay: t	$E^*(t)$	Required bandwidth: $b(t) = \frac{E^*(t)}{t}$	$\frac{b(t)}{Peak\ Rate}$
1	12	12.0 (Peak rate)	100 %
2	17	8.5	71 %
3	18	6.0	50 %
4	23	5.8	49 %
5	25	5.0	42 %
6	28	4.7	40 %

Table 2.8: The example's empirical envelope and bandwidth to be reserved

2.3.1 Traffic Constraint Function and Empirical Envelope

Let $A[t_1,\ t_2]$ denote the number of bytes generated in the interval from time t_1 to time t_2. Then, the *traffic constraint function* A^* is defined as the worst-case traffic characterization of the traffic A *([KWL+95], [LW96])*. It should satisfy the following properties:

1. *Time-invariance*:

$$A[\tau,\ \tau + t] \ \leq \ A^*(t) \quad \forall t > 0,\ \tau > 0$$

   The traffic constraint function should be independent of the starting time τ. In this case, policing is independent of the starting time, too.

2. *Subadditivity*:

$$A^*(t_1) \ + \ A^*(t_2) \ \geq \ A^*(t_1 + t_2) \quad \forall t_1, t_2 \ \geq \ 0$$

   This allows the arrivals $A[t_1,\ t_2]$ to attain the bound given by A^*: It will be feasible, that $A[\tau, \tau + t] \ = \ A^*(t)$ for any $t > 0$.

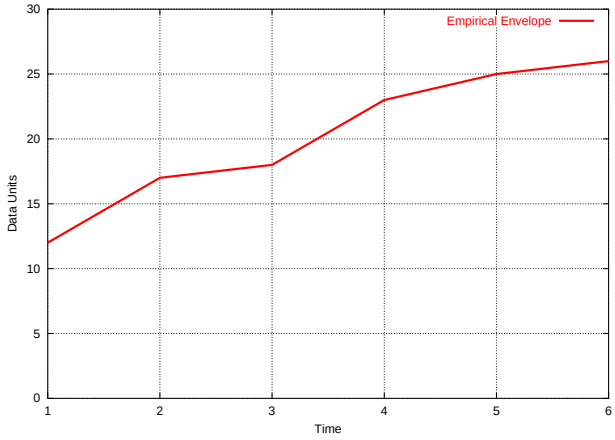


Figure 2.5: An empirical envelope

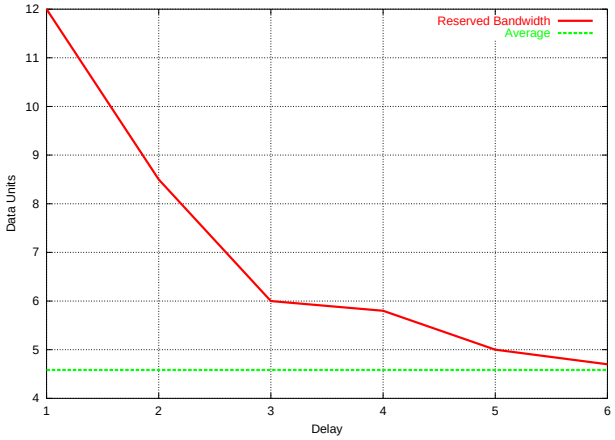
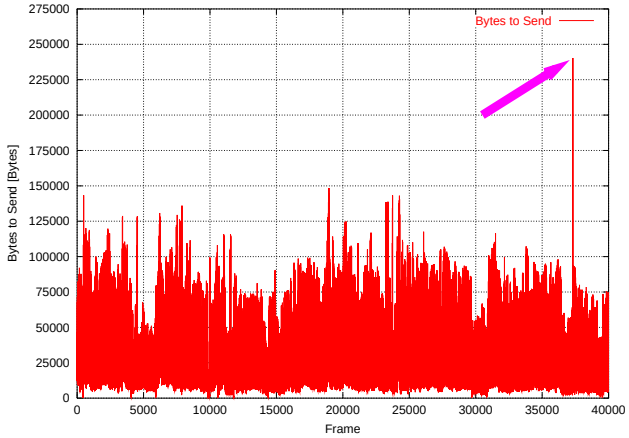


Figure 2.6: Bandwidth to be reserved



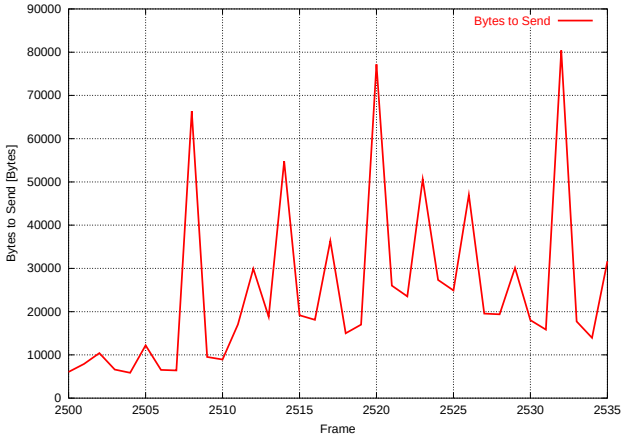


Figure 2.7: An example for long-term and short-term burstiness

The tightest traffic constraint function is the *empirical envelope* ([KZ97]): It is defined as follows:

$$E^*(t) \;=\; \sup_{\tau>0} A[\tau,\, \tau + t] \quad \forall t > 0$$

In other words, this is the maximum number of bytes sent in any interval of length t.

An example of a VBR stream of 12 frames can be found in figure 2.5. Each bar shows the size of a frame. The corresponding empirical envelope can be found in table 2.8 along with the bandwidth required for each delay ($b(t) \;=\; \frac{E^*(t)}{t}$) and the fraction of the peak rate. Plots of the empirical envelope and the required bandwidth can be found in figure 2.5 and 2.6. As shown in table 2.8 and figure 2.6, for a small delay of 3 frames, the bandwidth requirement already decreases to 50% of the original value. The rate of 4.7 units/frame at a delay 6 frames is even very close to the average rate of 4.583 units/frame.

The result is that buffering can save a lot of bandwidth allowing more customers to use a link simultaneously, especially if there is a large variance in the frame sizes.

2.3.2 Burstiness

A measurement value for the variance of frame sizes is the so called *burstiness* of a stream:

Model	Storage	Traffic Constraint Function
(σ, ρ)	$\sigma,\ \rho$	$A^*(t) = \sigma + \rho * t$
$(\overrightarrow{\sigma}, \overrightarrow{\rho})$	$(\sigma_i,\ \rho_i)$	$A^*(t) = \min_{1 \le i \le m} \{\sigma_i + \rho_i * t\}$
D-BIND	$(R_i\ , I_i)$	$A^*(t) = \begin{cases} t * R_1 & (t \le I_1) \\ \frac{I_{j-1} I_j * (R_{j-1} - R_j) + t * (R_j I_j - R_{j-1} I_{j-1})}{I_j - I_{j-1}} & (I_{j-1} < t \le I_j) \end{cases}$

Table 2.9: Traffic models and their traffic constraint functions

$$\text{Burstiness} = \frac{\text{Peak Rate}}{\text{Average Rate}}.$$

But it is important to denote here, that buffering is only realistic for delays of up to a few seconds. *Short-term burstiness* in the scope of some frames can easily be smoothed by buffering. Higher delays are usually not acceptable for the users and also require large buffers at the clients. Therefore, an additional concept is necessary to handle *long-term burstiness* in the scope of minutes or hours. This will be described in section 2.4.
An example is shown in figure 2.7. The left side shows the complete trace of the MPEG-1 video "*The Simpsons*"[8] consisting of 40000 frames at 25 frames/s. Note especially the lower and higher regions at a length of about 500 to 1000 frames. Smoothing by buffering would result in inacceptable delays of many seconds here! A small fraction - the frames from 2500 to 2532 - are shown on the right side. This figure has got another scaling for a better representation of the short-time behavior. Such short-time bursts can easily be smooted using buffering.

2.3.3 Traffic Models

Since it would be too expensive to store a complete empirical envelope, *traffic models* are required to approximate it. The most important traffic models are the (σ, ρ) and $(\overrightarrow{\sigma}, \overrightarrow{\rho})$ models ([KWL+95] and [LW96]) and the D-BIND (Deterministic Bounding INterval-Dependent, [KZ97] and [KWL+95]) model. Their traffic constraint functions are shown in table 2.9 (from [KWL+95]).

The (σ, ρ) model simply describes the traffic by giving the two parameters for a leaky bucket (see section 2.2.4 for a description): The buffer size σ and the output rate ρ. Therefore, the traffic constraint function is

$$A^*(t) = \sigma + \rho * t. \tag{2.2}$$

Since this approximation is quite inaccurate, the model has been extended to the $(\overrightarrow{\sigma}, \overrightarrow{\rho})$ model using an ordered set of leaky bucket parameters. In this model, the traffic constraint function is given by the minimum parameter pair:

$$A^*(t) = \min_{1 \le i \le m} \{\sigma_i + \rho_i * t\}. \tag{2.3}$$

Here, $A^*(t)$ is a convex function consisting of m piecewise linear segments. It is important to denote here, that this always implies a convex traffic constraint function. Since the empirical envelope itself is not necessarily convex, this also implies an inaccuracy. An example will be given below.
Another problem of these models based on leaky bucket parameters is the difficulty to choose 'good' leaky bucket parameters ([KZ97]): The network's state is dynamic and may not even be available at connection setup. If bandwidth is available and buffers are scarce, the sender may choose a too low rate ρ which implies a too high buffer size σ, and vice versa. Such 'bad' settings may cause a new connection to be rejected unnecessarily. For example, a stream's traffic description has been calculated assuming a large buffer size but only a low-bandwidth link. Now, the available network

[8] Trace source: [Wür95], simpsons.tar.gz.

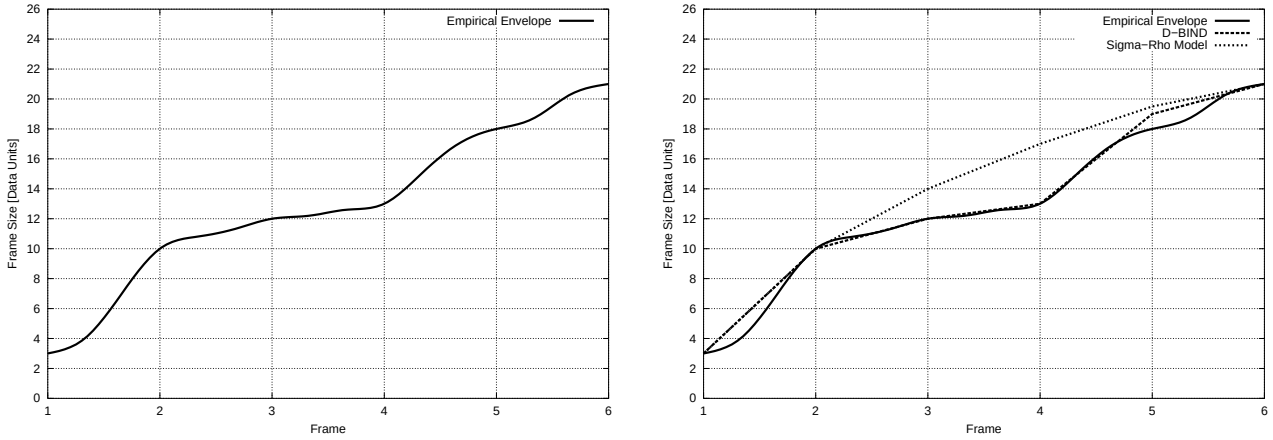


Figure 2.8: A traffic model comparision

bandwidth increases, but the buffer size remains constant (e.g. a fixed-size leaky bucket in a border router). This may result in new streams to be rejected because of too few buffer space.

To cope with these problems, the D-BIND model describes the traffic using a set of rate/interval length pairs (R_i, I_i). These contain the required data rate R_i for an interval of length I_i, that is $R_i = \frac{A^*(I_i)}{I_i}$. The resulting traffic constraint function is the linear interpolation between the D-BIND points:

$$A^*(t) = \begin{cases} t * R_1 & (t \leq I_1) \\ \frac{I_{j-1} I_j * (R_{j-1} - R_j) + t * (R_j I_j - R_{j-1} I_{j-1})}{I_j - I_{j-1}} & (I_{j-1} < t \leq I_j) \end{cases} . \tag{2.4}$$

Here, it is not necessary to approximate the traffic constraint function by a convex function anymore. Therefore, D-BIND has got a higher accuracy than the $(\overrightarrow{\sigma}, \overrightarrow{\rho})$ model. An example can be found in figure 2.8. The right side shows approximations of an empirical envelope using the $(\overrightarrow{\sigma}, \overrightarrow{\rho})$ model and D-BIND model. For better visibility, the original empirical envelope is plotted again on the left side. As it is shown, the D-BIND approximation is much nearer to the original due to its non-convex shape. This advantage is also proven by measurements in [KZ97]. However, the more accurate approximation has got a price: Policing a stream using a non-convex D-BIND description is more complex. For a detailed discussion of this, see [KZ97].

2.4 Bandwidth Remapping

Using one of the traffic models described above, it is now possible to give a traffic constraint for a stream. As mentioned in section 2.3.2, short-term burstiness can be smoothed by buffering. Now, it is necessary to examine long-term burstiness. Having again a look at the left side of figure 2.7, there is a large peak of about 240,000 bytes at about frame 37,000, marked by the arrow. Calculating a traffic description using the D-BIND or $(\overrightarrow{\sigma}, \overrightarrow{\rho})$ traffic model as described in section 2.3.3, it is necessary to allocate a bandwidth near the peak rate for very low buffer delays. But since the few other peaks are all below 150000 bytes, a lot of bandwidth would be wasted. This is called *over-provisioning*. The same effect also can also be found for the higher and lower areas of about 500 to 1000 frames. These are scenes of lower or higher bandwidth requirement causing the long-term burstiness. In this case, even a high buffer delay of up to a few seconds would not be able to improve this over-provisioning.

To cope with this problem, the trace can be partitioned into several *intervals*. For each interval, an own traffic description is calculated and used. This results in parts having lower bandwidth requirements allocating less bandwidth and vice versa. Now, a *bandwidth remapping* is necessary at the

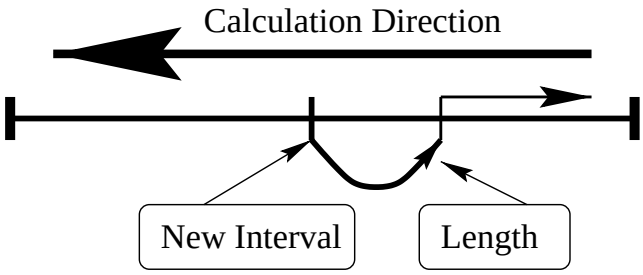


Figure 2.9: Graphical explaination of the remapping intervals calculation

interval borders. Therefore, these intervals are called *remapping intervals*.

Algorithm 1 Optimal calculation of remapping intervals from [Gum98]

```
01 void calculateIntervals(N) {
02    for(i = N - 1;i ≥ 0;i--) {
03       for(length = 1;length ≤ N - 1;i++) {
04          Calculate traffic description for interval
05             I = [i, ..., i + length - 1].
06          cost = cost_Remapping(I) + cost_Bandwidth(I) + cost[i + length];
07          if(cost < cost[i]) {
08            cost[i]   = cost;
09            length[i] = length;
10          }
11       }
12    }
13 }
```

Now, an algorithm to calculate the interval sizes is required which generates intervals of the lowest cost. Such an algorithm has been developed in [Gum98], its pseudocode is shown in algorithm 1. The optimal intervals refering to the cost of the bandwidth ($\text{cost}_{\text{Bandwidth}}$) and the cost of the remapping ($\text{cost}_{\text{Remapping}}$, usually a constant) are calculated there.

As shown in figure 2.9, the algorithm runs from the media's last frame to the first one. For every frame, all possible lengths to the end are tested: The interval length which causes minimum cost concatenated with the already calculated minimum-cost intervals to the end will be chosen. Finally, the algorithm has calculated the interval length and traffic description for **every** frame of the media. The algorithm's runtime is $O(n^3)$. To achive this, the traffic description has to be calculated in time $O(n)$ instead of $O(n^2)$ (runtime for the calculation of the empirical envelope approximation). This is possible by reusing previously calculated results and calculating only the difference.

2.5 Resource Management

While the network QoS requirements are only bandwidth, maximum transfer delay, maximum loss rate and maximum jitter ([CCH94a], [CCH94b], [WCH96], [NS96]), applications have got their own requirements or *QoS dimensions* ([LS98], [LLR+99]). For example, the QoS dimensions of a video transmission are picture format, color depth, frames per second and end-to-end delay or sampling rate, bits per sample and channels for an audio transmission. For the user of this application, the only requirement is, that video or audio transmission looks or sounds 'good'. This is called the *perceptual quality*, *user satisfaction* ([Rog98]) or *utilization* ([LS98], [LLR+99]).

These different requirements lead to a so called *layered QoS model* ([ACH95], [CCH94a], [NS96] and [Rog98]), which is shown in figure 2.10. A mapping (also called QoS translation by [NS96]) is

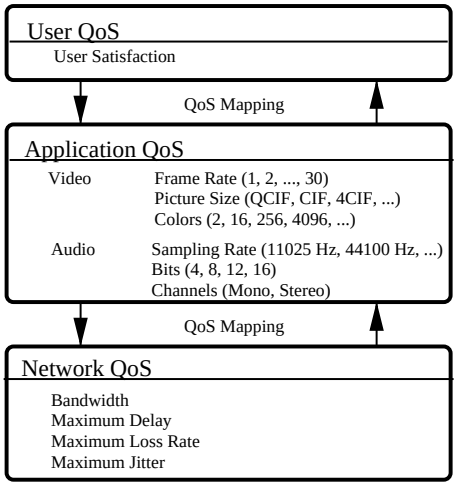


Figure 2.10: The layered QoS model

required between the dimensions of each layer. For example, the user requests 'high' quality for a video. This requirement is translated to a set of possible application QoS settings, e.g. (16CIF[9] pixels at 25 pictures/s in 64K colors, 4CIF[10] pixels at 20 pictures/s in 16M colors}. A 'useful' (e.g. the most cost efficient) possible setting has to be chosen and translated into network QoS requirements, e.g. 5 MBytes/s bandwidth at a maximum delay of 1000ms and a maximum acceptable loss rate of 0.5%. Now, a solution to evaluate the effect of parameter changes on the user satisfaction is necessary.

2.5.1 Utility Functions

For some elastic traffic like file transfer or WWW, the utilization usually increases linearly with the bandwidth (user satisfaction $\sim$ speed). But for multimedia transmission, this is normally not true. For example in an uncompressed audio transmission, increasing the sampling rate from 11,025 Hz to 22,050 Hz (double bandwidth) usually results in a significant improved quality. An increase from 22,050 Hz to 44,100 Hz (again double bandwidth) gives only a small improvement and increasing it to 98,200 Hz results in no improvement. The reason for this is, that the human ear is able to receive lower frequencies better than higher ones. Further, due to the sampling theorem[11] and the ear's limit to a maximum frequency of about 20,000 Hz, sampling rates above about 40,000 Hz are not useful.

First, it is therefore necessary to describe the utilization for a given setting of a QoS dimension (e.g. frame rate, samping rate, picture size etc.) by a so called *utility function*. In the next step, the dimension-wise utility functions have to be composed to the user satisfaction (total utilization, perceptual quality).

Utility functions can be calculated for example in the following ways from a media and a set of different scaling steps (e.g. the original high-resolution video at 30 frames/s and some resolution-reduced versions having frames rates of 5 frames/s to 30 frames/s in steps of 5 frames/s):

- A group of users can rate the quality. But of course this can be very subjective, e.g. the users are very interested in a video and miss some errors or are bored and search for errors very accurately, etc..

- Another way is to use a so called *quality metric* to compare the scaled media to the original one. If the metric also uses a *perceptual model* which tries to "rate" the media like a human user,

[9] 16CIF: 1408×1152 pixels (16× resolution of the Common Intermediate Format 352×288).
[10] 4CIF: 704×576 pixels (4× resolution of the Common Intermediate Format 352×288).
[11] Sampling Theorem: To record frequencies up to *n* Hz, it is necessary to use a sampling rate of at least *2*n* Hz.

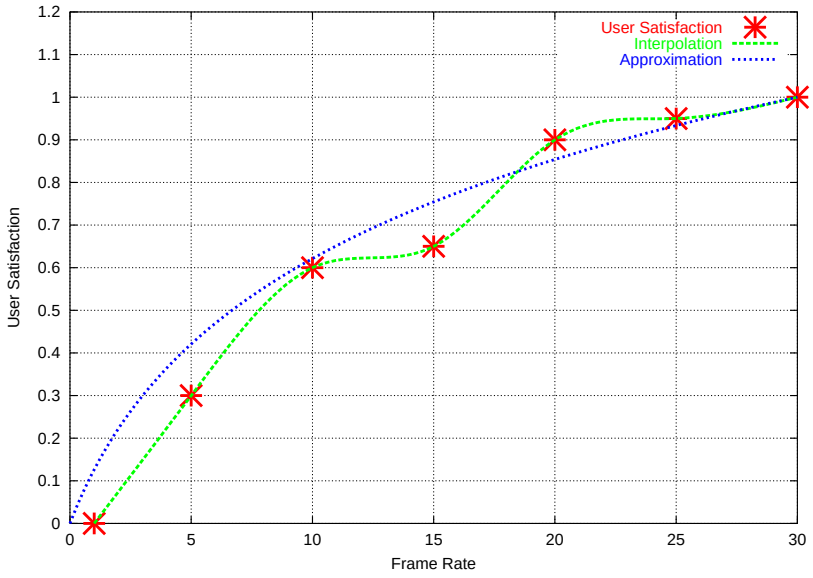


Figure 2.11: Utilization points => interpolated and approximated utility function

this method can be very accurate. More details about metrics for different media types can be found in section 2.6.

- Finally, a set of utility functions for different media categories (e.g. action, sports, talk shows, etc. for video and classic music, rock music, news, etc. for audio) can be calculated using the two methods above. New medias can be mapped to one of these categories using the corresponding utility function. See section 2.6 for an example.

Using one of this methods, some so called *utilization points* are obtained. These points describe the utilization for a given resource, that is the value of the QoS dimension, e.g. frame rate for a video. For simplicity, the lowest quality has got utilization 0.0 (0%) and the highest one utilization 1.0 (100%). Now, a curve can be interpolated from these points. To store the utility function efficiently, it is also recommended to approximate it by a simple function. An example can be found in figure 2.11, the approximation uses the utility function described below, $p = 12.5$.

In [Rog98], an utility function is defined as follows:

$$u(x) := c * \ln(a * x + b), \tag{2.5}$$

$$a := \frac{1}{p - 10},$$

$$b := \frac{e^{\frac{1}{a}} - 1}{x_{\max} - x_{\min}},$$

$$c := \frac{x_{\max} - x_{\min} * e^{\frac{1}{n}}}{x_{\max} - x_{\min}},$$

$$u(x_{\min}) = 0 \quad ; \quad u(x_{\max}) = 1$$

$x_{\min}$ and $x_{\max}$ give the minimum and maximum values of the QoS dimension, e.g. 1 and 30 for a video's frame rate. The so called *sensitivity parameter* p allows choosing the shape of the function. In [Rog98], this is done by the user. Below 10, the utility function is concave, linear for 10 and convex

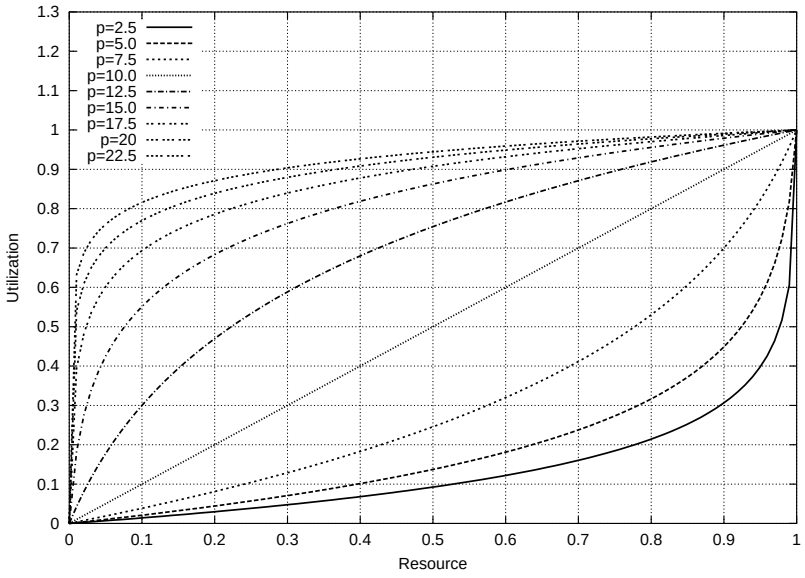


Figure 2.12: The utility function $u(x) = a * \ln(b * x + c)$

above 10. Therefore, the utilization increases faster for higher values of p. The function for different settings of p is plotted in figure 2.12.

A second utility function is introduced in [LS98] and [LLR+99]:

$$u(x) := 1 - e^{a*x+b} \tag{2.6}$$

where the constants a and b are given by the settings x_{50} and x_{95} for utilizations of 50% and 95%:

$$0.5 = 1 - e^{a*x_{50}+b} \quad ; \quad 0.95 = 1 - e^{a*x_{95}+b}$$

$$b = \frac{x_{95} * \ln(-0.05 + 1.00) - x_{50} * \ln(-0.95 + 1.00)}{x_{95} - x_{50}}$$

$$a = \frac{\ln(-0.95 + 1.00) - b}{x_{95}}$$

x_{50} and x_{95} are given by the user in [LS98] and [LLR+99]. Some examples can be found in figure 2.13. As it is shown in the examples, the function's value can go below 0 and is not exactly 1 at the maximum x value! Therefore, values below 0 should be threatened as 0 and $u(x_{\max}) := 1$, that is:

$$u(x) = \begin{cases} 0 & (1 - e^{a*x+b} < 0) \\ 1 & (x \geq x_{\max}) \\ 1 - e^{a*x+b} & \text{else} \end{cases}$$

Now, the dimension-wise description of utilization can be done using each dimension's utility function. But to describe the user satisfaction (total utilization, perceptual quality), it is necessary to compose them.

2.5.2 Composition of Utility Functions

[LS98] simply suggests to use a weighted sum, that is:

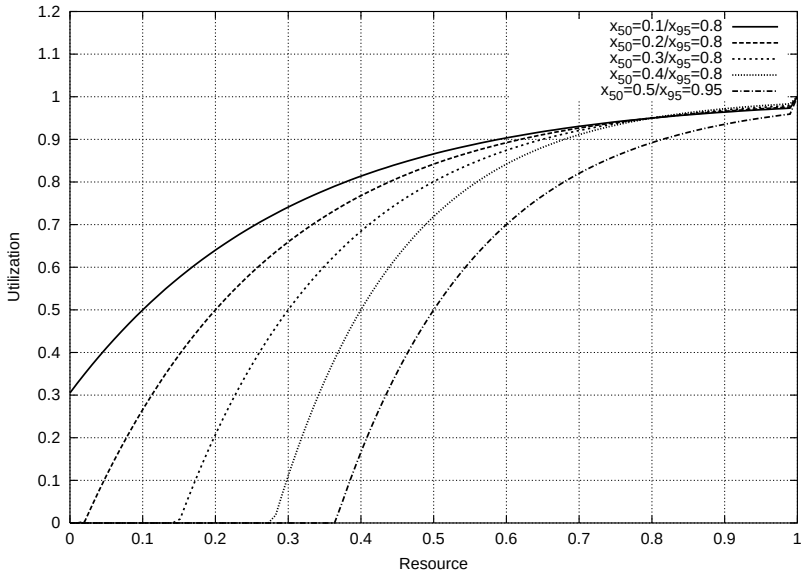


Figure 2.13: The utility function $u(x) = 1 - e^{a*x+b}$

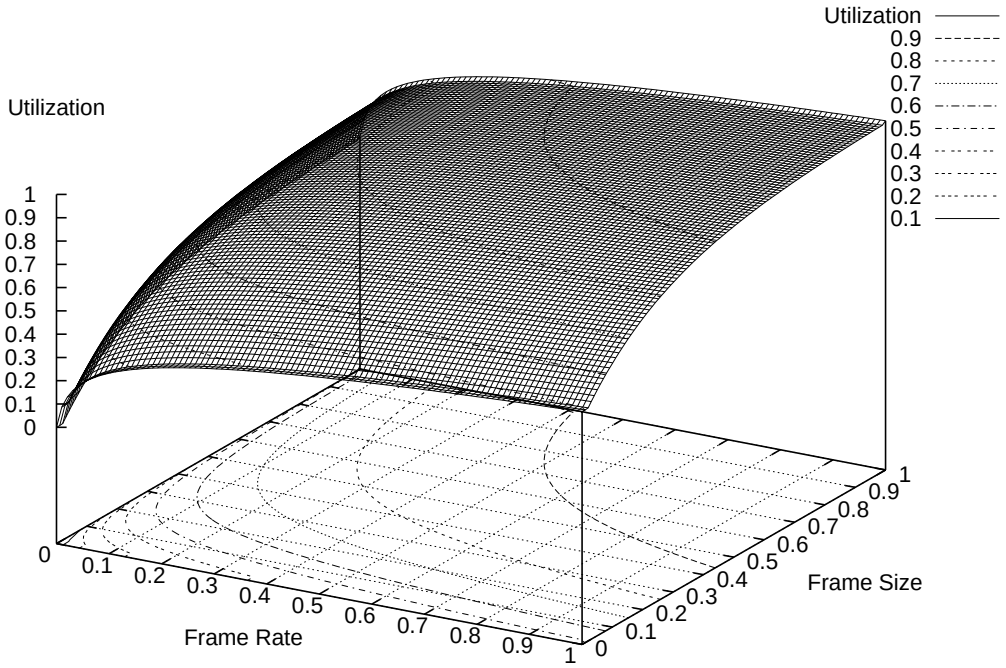


Figure 2.14: The utility function composition $U(x) = \sum_{i=1}^{n} \omega_i * u_i(x_i)$ from [LS98]

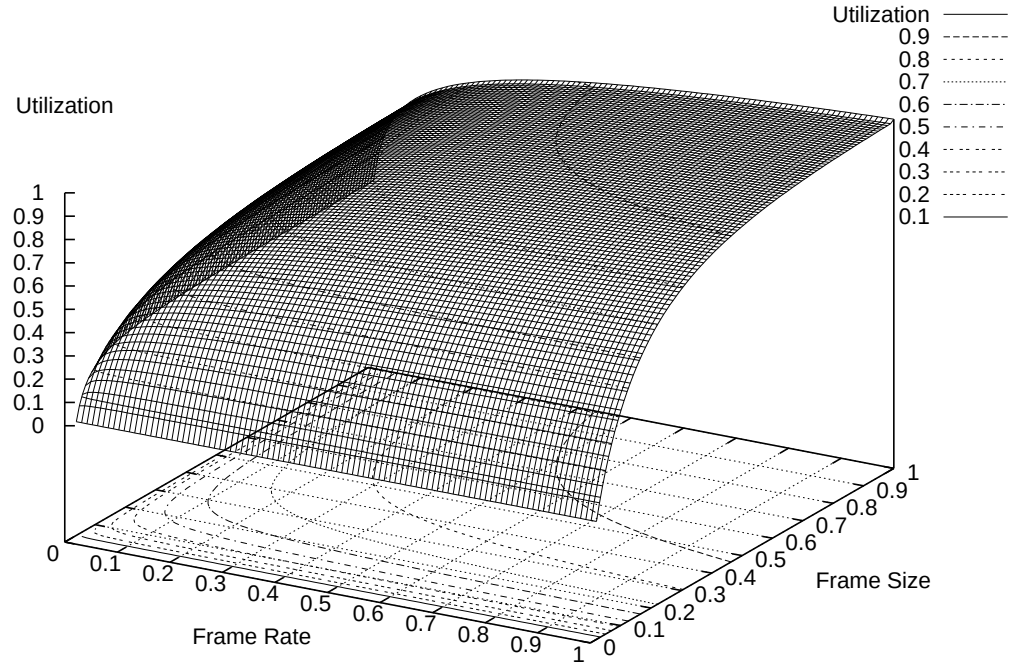


Figure 2.15: The utility function composition $U(x) = \frac{n}{\sum_{i=1}^{n} \frac{1}{u_i(x_i)}}$ from [Rog98]

$$U(\overrightarrow{x}) = \sum_{i=1}^{n} \omega_i * u_i(x_i), \tag{2.7}$$

where ω_i is the weight for the i-th QoS dimension having value x_i and utility function u_i. This makes it possible to emphasize some dimensions. For example, table 1 in [AFK+95] shows that for a comedy video, frame size is more important than frame rate. Therefore, an example utility function may be:

$$U(\overrightarrow{x}) = \underbrace{5 * (1 - e^{a_1 * x_1 + b_1})}_{Video\,Frame\,Rate} + \underbrace{15 * (a_2 * \ln(b_2 * x_2 + c)}_{Video\,Frame\,Size} + \underbrace{1 * (1 - e^{a_3 * x_3 + b})}_{Audio}.$$

An example for two dimensions is shown in figure 2.14 (Frame rate: Forumla 2.5, $p = 15$. Frame size: Forumla 2.6, $q_{50} = 0.2$, $q_{95} = 0.8$).

A second possibility to compose utility functions is presented in [Rog98]:

$$U(\overrightarrow{x}) = f(u_1(x_1), u_2(x_2), ..., u_n(x_n))$$

This function f should have the two following properties:

1. If x_1 is much smaller than each of the $\{x_2, ..., x_n\}$, then $U(\overrightarrow{x})$ must be dominated by x_1. Again an example from table 1 of [AFK+95]: In a sports video, both frame rate and frame size are important. If frame size is very high but frame rate very low (like a slideshow), then the resulting user satisfaction will be low.

2. $f(s, s, ..., s)$:= s. This allows $f(x_1, x_2, ..., x_n)$ to be scaled.

One relationship satisfying (1) and (2) is:

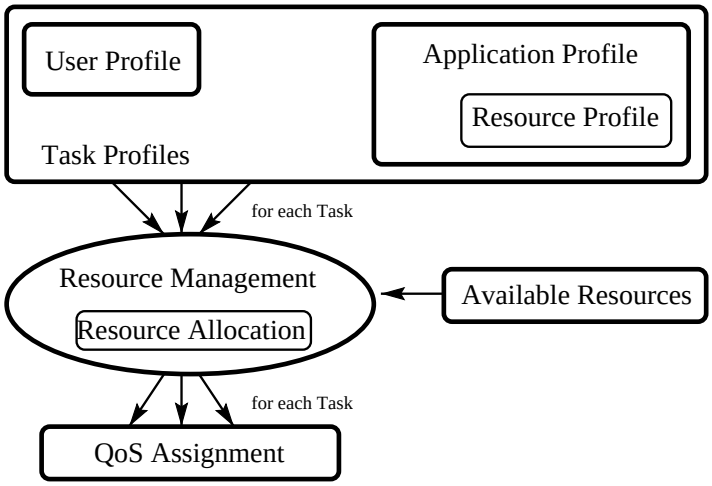


Figure 2.16: The resource management based on [LS98]

$$\frac{1}{U(\overrightarrow{x})} = \frac{1}{n} * \sum_{i=1}^{n} \frac{1}{u_i(x_i)} \qquad \Leftrightarrow \qquad U(\overrightarrow{x}) = \frac{n}{\sum_{i=1}^{n} \frac{1}{u_i(x_i)}}. \tag{2.8}$$

An example for two dimensions is shown in figure 2.15 (Frame rate: Formula 2.5, $p = 15$. Frame size: Forumla 2.6, $q_{50} = 0.2,\ q_{95} = 0.8$).

2.5.3 Resource Management Definitions

Now, the effect to the user satisfaction caused by changing a value of a QoS dimension can be described using the composed utility functions. The next step is to map resources (e.g. bandwidth) to a stream using the user satisfaction calculation to generate a 'useful' mapping. The resource management approach used in this work is based on [LS98] and [LLR+99], a graphical view can be found in figure 2.16. But first, it is necessary to introduce some definitions from [LS98] and [LLR+99]:

- Tasks (e.g. applications): $T_1,\ T_2,\ ...,\ T_n$.

- QoS dimensions of task T_i: $Q_{i1},\ Q_{i2},\ ...,\ Q_{id_i}$.
  Each Q_{ij} is a finite set of values for the j-th QoS dimension of task i. For example, this can be a set of allowed frame rates: {1, 2, ..., 30}. The set of possible quality vectors is therefore defined as $Q_i = Q_{i1} \times Q_{i2} \times ... \times Q_{id_i}$.

- Shared system resources: $R_1,\ R_2,\ ...,\ R_m$.
  Resources can be for example bandwidth, CPU time, memory or harddisk usage. Each R_i is a finite set of non-negative values representing an allocation choice for resource j. The possible set of resource vectors is $R_i = R_1 \times R_2 \times ... \times R_m$. It also includes vector $r^{max} = (r_1^{max},\ r_2^{max},\ ...,\ r_m^{max})$ since all resources are finite.

Each task is associated with a *task profile* consisting of a *user profile* and an *application profile*. The first one comes from the application itself and contains

- a quality space Q_i,

- a *quality index*[12] - a bijective function $f_{ij} : Q_{ij} \rightarrow \{1, 2, ..., \mid Q_{ij} \mid\}$ that preserves the ordering: If q_1 is better than q_2, then $f_{ij}(q_1) > f_{ij}(q_2)$,
- dimension-wise, non-decreasing utility functions $u_{ij} : Q_i \rightarrow \mathbb{R}$ (see section 2.5.1),
- *application utilities* $U_i : Q_i \rightarrow \mathbb{R}$, that is the composition of the utility functions (see section 2.5.2) and
- a *resouce profile*: This is a relation between the resouce set R and the quality set Q_i, where $r \models_i q$ describes a list of possible resource allocations achieving quality q. $\models_i$ has to respect the partial orderings of R and Q_i, that is

$$[r_1 \models_i q_1 \quad \wedge \quad r_2 \models_i q_2 \quad \wedge \quad r_1 > r_2] \quad \Rightarrow \quad q_1 > q_2$$

  Since it is possible to achieve a quality q by more than one resource setting, it is not possible to use a function instead of the relation here! Using this relation, two functions can be defined:

  1. The maximum possible utilization for a given resource setting:

$$g_i : R \rightarrow \mathbb{R} \quad ; \quad g_i(r) = \max\{U_i(q) \mid q \in Q_i \wedge r \models_i q\}. \tag{2.9}$$

     Using this function, the so called *resource/utilization graph* (R-U graph) can be generated for each task. The point $(r, g_i(r))$ is called *resource/utilization point*, a list of such points is a so called *resource/utilization list*.

  2. The set of all qualities possible for a given resource setting:

$$h_i : R \rightarrow \mathcal{P}(Q_i) \quad ; \quad h_i(r) = \{q \in Q_i \mid U_i(q) = g_i(r) \wedge r \models_i q\}. \tag{2.10}$$

The user profile contains a so called *QoS constraint*. This is the specification of the minimum QoS requirements:

$$q_i^{\min} = (q_1^{\min}, q_2^{\min}, ..., q_{\mathrm{id}_i}^{\min}).$$

For example in an audio transmission, this can be used to define the user's lowest acceptable quality at 11,025 Hz sampling rate and 8 bits per sample in stereo.

Finally, the so called *system utility* has to be defined. This is the composition of all tasks' application utility: $U : Q \rightarrow \mathbb{R}$. Two concepts can be used here:

- Maximizing the application utilities using a weighted sum: $U(\overrightarrow{q}) = \sum_{i=1}^{n} \omega_i * u_i(q_i)$, where $\omega_i \geq 0$ is the priority for task i. Assuming that all priorities are the same, this will result in applications having lower resource requirements (e.g. audio streams, 100 KBytes/s for 100% utilization) getting higher quality easier than applications having high resource requirements (e.g. video, 100 KBytes for 5% utilization).
- *Utility fair sharing*: $U(\overrightarrow{q}) = \min_{i=1,...,n}\{u_i(q_i)\}$. The resource mapping only refers to the application utilities. Therefore, it tries to give all applications the same utilization independently of resource requirements!
  But for example sending one high-bandwidth video and 20 low-bandwidth audio streams over a low-speed network, the mapping may result in 10% utilization for all applications instead of giving the video 5% and all audio streams 100%.

Now, the so called *QoS optimization problem* can be defined: Maximize $U(q_1, q_2, ..., q_n)$, where

[12] The quality index is necessary if the QoS dimensions have e.g. non-numerical values. For example, frame rate is numerical and therefore no problem. But a picture format may contain e.g. QCIF, CIF, 4CIF, 16CIF etc.. In this case, a mapping to numerical values is required.

- $q_i \geq q_i^{\min}$ for all i=1, 2, ..., n (QoS Constraints)
- $\sum_{i=1}^{n} r_{ij} \leq r_j^{\max}$ for all j=1, 2, ..., m (Resource Constraints) and
- $r_i \models_i q_i$ for all i=1, 2, ..., n (Resource Profiles).

2.5.4 QoS Optimization Algorithms

In [LS98] and [LLR+99], the QoS optimization problem is grouped into four categories:

- Single Resource Single QoS Dimension (SRSD),
- Single Resource Multiple QoS Dimension (SRMD, superset of SRSD),
- Multiple Resource Single QoS Dimension (MRSD),
- Multiple Resource Multiple QoS Dimension (MRMD, superset of MRSD).

Unfortunately, SRSD, SRMD, MRSD and MRMD are all NP-hard. This is proven in [LS98] by constructing a poly-time reduction from SRSD to the 0-1-Knapsack problem. Since this problem is known to be NP-hard, the QoS optimization problem is also NP-hard. Since the SRSD problem on a NTM will have a polynomial runtime (simply testing all possibilities), the problem is also NP-easy. Therefore, it is NP-complete and if some day somebody develops a poly-time algorithm for this problem, then $P = NP$ is proven.

But under the assumption of $P \neq NP$, it is not possible to get an optimal solution for this problem in poly-time. Fortunately, it is possible to approximate the SRMD and therefore the SRSD problem very efficiently. In [LS98], three algorithms for SRMD are presented and compared by measurements in [LLR+99]:

1. ASRMD1 - A simple but fast approximation having an uncontrollable bound.
2. ASRMD2 - An approximation with an upper bound. But the measurements show, that the runtime is much higher than for ASRMD1.
3. SRMD - An optimal solution for a given resource granularity having inacceptable runtime.

Since only the first one, ASRMD1, has got an acceptable runtime for the system described in this work, only this algorithm is explained here. Although the bound is not limited, the measurements in [LLR+99] show that there is usually no significant difference between the results of ASRMD1 and ASRMD2. See [LS98] and [LLR+99] for details about the other algorithms.

The ASRMD1 algorithm is shown in algorithm 2. For each task T_i, a resource/utilization list C_i, sorted ascending by resource, is given. First, the convex hull of each list is calculated (line 7): $\widetilde{C}_i$. This has the following effects to the resource/utilization list:

- Points having lower utilization than previous points are removed. Therefore:

$$p_1.\text{Resource} < p_2.\text{Resource} \quad \Rightarrow \quad p_1.\text{Utilization} < p_2.\text{Utilization}.$$

- $\widetilde{C}_i$ will be convex: This implies, that points below the line from the $(r_1,\ g_i(r_1))$ to $(r_n,\ g_i(r_n))$ are removed. Such points would have got a 'bad' resource/utilization ratio $\frac{\text{Utilization}}{\text{Resource}}$, causing high cost (resource) at low utilization.

Algorithm 2 The ASRMD1 algorithm from [LS98]

```
01 global resAllocated[n];   // Resource allocated to task T_i
02 global utilization[n];    // Utilization of task T_i
03 global Resource = <Maximum resource, e.g.  bandwidth from SLA>;
04
05 void asrmd1(n, C_1, ..., C_n) {
06    for(i = 1;i ≤ n;i++) {
07       C̃_i = calculateConvexHull(C_i);
08       resAllocated[i] = 0;      // No resources for task T_i
09       utilization[i]  = -1.0;   // No utilization for task T_i
10    }
11
12    list = merge(C̃_1, ..., C̃_n);   // Order is preserved!
13    resAvailable = Resource;     // Resource to be divided up
14
15    for(i = 1;i ≤ |list|;i++) {
16       task   = list[i].Task;      // Allocation trial for P_i, task T_task
17       resNew = list[i].Resource - resAllocated[task];
18       if(resNew ≤ resAvailable) {
19          resAvailable -= resNew;
20          resAllocated[task] = list[i].Resource;
21          utilization[task]  = list[i].Utilization;
22       }
23    }
24 }
```

Next, the lists $\widetilde{C}_i$ are merged to list *list*, preserving the ascending order by resource (line 12). Then, *list* is iterated from the first to the last element. In every iteration, the algorithm tries to allocate resource for the current resource/utilization point: If the difference between the corresponding task's allocation and the point's requirement is less or equal the available resource, the new allocation will be successful. Finally, each task T_i has got its allocation $resAllocated_i$ and using its function h_i, a quality can be set by choosing a vector $q \in h_i(resAllocated_i)$ having utilization $utilization_i$. Note, that always $h_i(0) := \varnothing$ for the case of too few resource available.

A complete example for resource = bandwidth:

$$C_1 = \left\langle \begin{pmatrix} 10 \\ 0.0 \end{pmatrix}, \begin{pmatrix} 30 \\ 0.2 \end{pmatrix}, \begin{pmatrix} 40 \\ 0.9 \end{pmatrix}, \begin{pmatrix} 70 \\ 1.0 \end{pmatrix} \right\rangle,$$

$$C_2 = \left\langle \begin{pmatrix} 15 \\ 0.0 \end{pmatrix}, \begin{pmatrix} 40 \\ 0.5 \end{pmatrix}, \begin{pmatrix} 80 \\ 0.8 \end{pmatrix}, \begin{pmatrix} 100 \\ 1.0 \end{pmatrix} \right\rangle.$$

Here $\begin{pmatrix} b \\ u \end{pmatrix}$ denotes utilization u for b bandwidth units. C_1 and C_2 are plotted in figure 2.17. As it is shown, C_1 is not convex. Therefore, $\begin{pmatrix} 30 \\ 0.2 \end{pmatrix}$ will be removed in the convex hull (see figure 2.17). This results in the following merged list:

$$\text{list} = \left\langle \underbrace{\begin{pmatrix} T_1 \\ 10 \\ 0.0 \end{pmatrix}}_{P_1}, \underbrace{\begin{pmatrix} T_2 \\ 15 \\ 0.0 \end{pmatrix}}_{P_2}, \underbrace{\begin{pmatrix} T_1 \\ 40 \\ 0.9 \end{pmatrix}}_{P_3}, \underbrace{\begin{pmatrix} T_2 \\ 40 \\ 0.5 \end{pmatrix}}_{P_4}, \underbrace{\begin{pmatrix} T_1 \\ 70 \\ 1.0 \end{pmatrix}}_{P_5}, \underbrace{\begin{pmatrix} T_2 \\ 80 \\ 0.8 \end{pmatrix}}_{P_6}, \underbrace{\begin{pmatrix} T_2 \\ 100 \\ 1.0 \end{pmatrix}}_{P_7} \right\rangle.$$

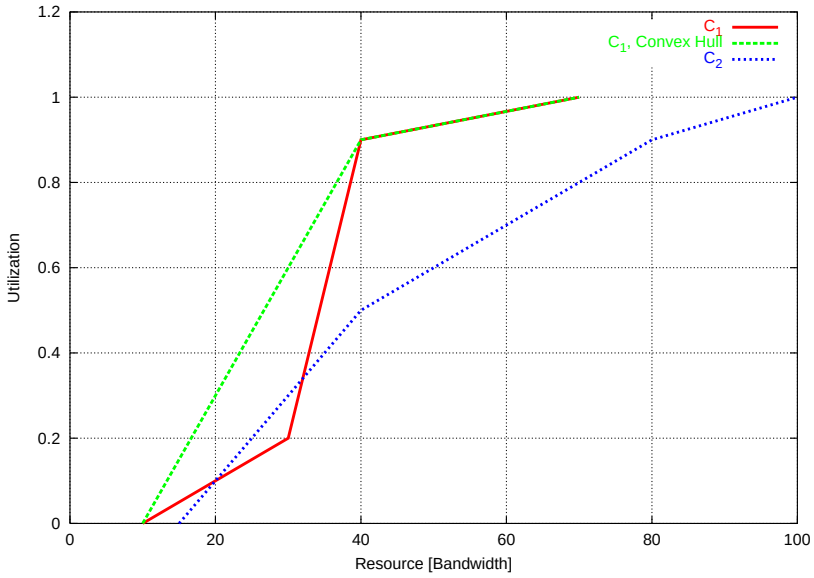


Figure 2.17: An example for the ASRMD1 algorithm

For a maximum bandwidth of 120 units, the allocation will be successful for P_1 to P_5, resulting in 100% utilization for T_1 at bandwidth 70 (P_5) and 50% utilization for T_2 at bandwidth 40 (P_4). 10 bandwidth units remain unallocated.

It is very important to denote here, that this algorithm tries to maximize the application utilities. By sorting the resource/utilization lists by utilization instead of resource, the algorithm will generate a fair sharing, that is trying to give all tasks equal utilization! In this case, *list*'s contents would be as follows:

$$\text{list}_{\text{fair}} = \left\langle \underbrace{\begin{pmatrix} T_1 \\ 10 \\ 0.0 \end{pmatrix}}_{P_1}, \underbrace{\begin{pmatrix} T_2 \\ 15 \\ 0.0 \end{pmatrix}}_{P_2}, \underbrace{\begin{pmatrix} T_2 \\ 40 \\ 0.5 \end{pmatrix}}_{P_3}, \underbrace{\begin{pmatrix} T_2 \\ 80 \\ 0.8 \end{pmatrix}}_{P_4}, \underbrace{\begin{pmatrix} T_1 \\ 40 \\ 0.9 \end{pmatrix}}_{P_5}, \underbrace{\begin{pmatrix} T_1 \\ 70 \\ 1.0 \end{pmatrix}}_{P_6}, \underbrace{\begin{pmatrix} T_2 \\ 100 \\ 1.0 \end{pmatrix}}_{P_7} \right\rangle.$$

In this case, the allocation of 120 bandwidth units results in 90% utilization for T_1 at bandwidth 40 (P_5) and 80% utilization for T_2 at bandwidth 80 (P_4). 0 bandwidth units remain unallocated here.

2.6 Media Types

This section describes the basics of the media types used in this work: MPEG-1/2, H.263 and MP3. It should be denoted here that only a granular summary of the main concepts and properties can be given here, due to the complexity of this subject. For details see the corresponding citations to documentation and papers. All types of this section group their data to logical data units (see [MM00], page 572), so called *frames*. Frames transport a certain part of a media or media objects of a specified duration, e.g. a single picture of a video or a defined amount of samples for audio data. The number of frames per second is the so called *frame rate*.

2.6.1 MPEG-1 Video

In 1990, the Motion Pictures Experts Group (MPEG) defined the MPEG-1 standard for video storage and transmission. It is optimized for bandwidths of about 1.5 MBit/s, used e.g. for video-CDs and video on demand. The properties this standard are as follows([MPEG], [RN98]):

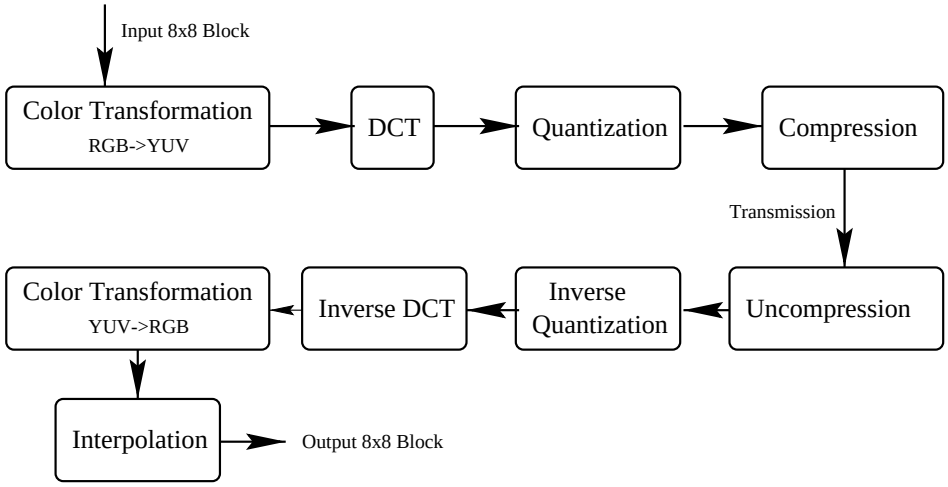


Figure 2.18: The MPEG codec

- Random access,
- fast forward and fast reward are possible,
- reverse play,
- error tolerance (e.g. lost packets) implying no necessity to use reliable transmission,
- jitter tolerance (necessary for packet networks like the Internet without strict guarantees),
- different resolutions and frame rates for supporting various link speeds (e.g. telephone lines, high-speed networks, etc.),
- possibility for real-time encoding and decoding and
- low cost (affordable decoders for end-users).

The basics of the compression are the *discrete cosine transformation* (DCT) and *motion compensation.* MPEG uses temporal redundancy and the limits of the human visual system to achive compression ratios about 1:30.

MPEG Encoding and Decoding

MPEG encoding consists of color transformation, DCT, quantization and compression, see figure 2.18 for a graphical view of the MPEG codec. Since the human eye is more sensitive for luminance than for chrominance, it is not useful to use RGB colors to store the video's pictures. Therefore, the YUV (Y luminance; U, V chrominance - fractions of red and blue), also called YC_rC_b model is used. The transformation between RGB and YUV is done the following way ([MM00]):

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.144 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.0 & 0.0 & 1.402 \\ 1.0 & -0.34414 & -0.71414 \\ 1.0 & 1.772 & 0.0 \end{pmatrix} \begin{pmatrix} Y \\ U \\ V \end{pmatrix}$$

A picture is divided in so called *blocks* consisting of 8×8 pixels. Since luminance is more important than chrominance, the chrominance resolution may be halved horizontally and/or vertically using the average value (*color subsampling*) resulting in Y:U:V ratios of 4:2:2 or 4:1:1 instead of 4:4:4. Four blocks of luminance and four, two or one of chrominance are grouped to a *macroblock* which contains a 16×16 pixel area.
After color transformation, DCT ([RN98]) is applied on each 8×8 block ($s_{x,y}$ is the value for entry x, y):

$$F(u,v) = \frac{1}{4}C(u)C(v) * \sum_{x=0}^{7}\sum_{y=0}^{7}\left[s_{x,y} * \cos\left(\frac{\pi u(2x+1)}{16}\right)\cos\left(\frac{\pi v(2y+1)}{16}\right)\right]$$

$$C(\varepsilon) = \begin{cases} \frac{1}{\sqrt{2}} & (\varepsilon = 0) \\ 1 & (\varepsilon \neq 0) \end{cases}$$

The inverse transformation is F^{-1}:

$$F^{-1}(x,y) = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7}\left[C(u)C(v) * F(u,v) * \cos\left(\frac{\pi u(2x+1)}{16}\right)\cos\left(\frac{\pi v(2y+1)}{16}\right)\right]$$

Note, that due to the limited accurary of the calculation, there will be a small loss of information when applying reverse DCT to a DCT-transformed block, compared to the original one. Quantization of the coefficients is done using an own quantization value q_{uv} for each coefficient ([MM00]):

$$\widetilde{F}(u,v) = \text{round}(\frac{F(u,v)}{q_{uv}}).$$

The inverse calculation is:

$$\widehat{F}(u,v) = \widetilde{F}(u,v) * q_{vu}.$$

But of course, the higher the quantization value q_{uv} the higher the so called *quantization noise*.

The DCT itself results in no reduction of the data (8×8 → 8×8), but since a lot of the quantized coefficients are usually 0, a good compression can be achieved. This is done using Huffman codes and run-level coding. To provide error tolerance, a sequence of macroblocks is grouped to a so called *slice*. Every slice is decodable independently. Missing slices may be interpolated from available slices.

Unless there are scenes changes in the video, the difference between two pictures is usually small. For example, a car moves in front of a landscape. In this case, only storing a vector of the car's movement and the missing parts of the background would result in a huge bandwidth reduction. Since recognition of objects is too complex, the so called *motion compensation* is based on macroblocks. For every macroblock, the motion compensation algorithm searches for a 'best' matching part in the next picture and stores only a *motion vector*. If necessary, an *error picture*, that is the difference to the original picture, is also stored. The quality of the motion compensation is dependent on the algorithm: Algorithms finding better matches will be slower due to more comparisions.

MPEG videos consist of up to four picture types:

I-frames (Intra-coded pictures) contain a picture independent of other ones. Therefore, the compression ratio is low, but random access is possible.

P-frames (Inter-coded pictures) contain a picture using motion compensation. It refers to the previous I- or P-frame. Therefore, it is only decodable in combination with the picture it refers to.

B-frames (Bidirectionaly predictive coded pictures) contain motion compensation references to the previous and/or next I- or P-frame and encode only differences between them. Therefore, this

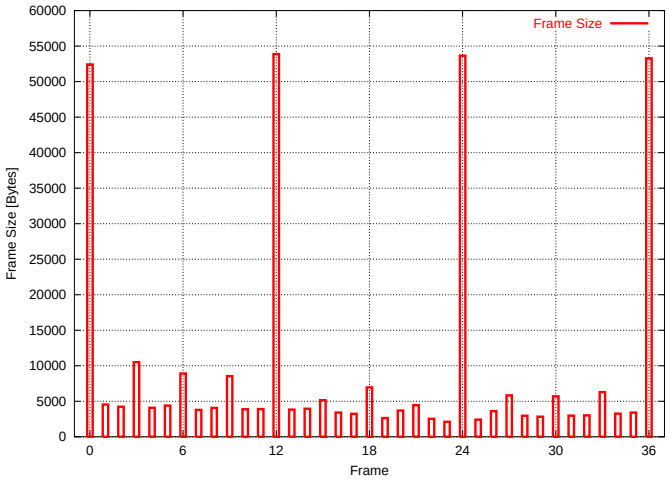


Figure 2.19: An MPEG example ("*The Silence of the Lambs*", GoP: IBBPBBPBBPBB)

type achieves the highest compression ratio. B-frames will never be used as motion compensation references!

D-frames are used for fast forward/fast reward only and contain low-quality intra-coded pictures. Usually, this type is used for stored videos only - it does not make sense to transmit it in real-time applications.

Error Tolerance

The sequence of picture types is given by the periodic *group of pictures* (GoP) pattern: Practical experience shows, that the GoP "IBBPBBPBB IBBPBBPBB ... " results in a good compromise between error reproduction and compression efficiency ([Gum98] on page 20, [RN98]): Since errors in I-frames are reproduced on all P and B pictures refering to it, the acceptable loss rate of this type is the lowest. On the other side, errors in B-frames only concern the B picture itself. Therefore, a higher loss rate may be acceptable for this type. The acceptable loss rate of P pictures may be between the I and B-frames' value - errors remain until the next I-frame is received.

An example for an MPEG stream of the video "*The Silence of the Lambs*"[13] using the GoP "IBBPBBPBBPBB" can be found in figure 2.19. The I:P:B size ratio is about 10:2:1, which is usual for MPEG videos ([RN98]).

Online Scalability

MPEG encoding requires - especially for the motion compensation - much CPU power. Therefore, online scaling by decoding and re-encoding a media would be too inefficient. But some simpler methods for online scaling have been developed:

Frame Dropping reduces the number of frames per second by skipping whole frames. First, B-frames are removed, then P-frames and at last I-frames to achieve a given lower frame rate.

Block Dropping reduces the number of blocks by simple removal ([ZL96]).

Coefficient Elimination skips some of the up to 64 DCT coefficients ([Gum98] on page 88). This can be done by giving a maximum number of coefficients to transmit (scalar breakpoint) or a 64-bit boolean vector containing true/false for every coefficient (vector breakpoint). Finally, it

[13] Trace source: [Wür95], lambs.tar.gz.

is also possible to limit the number of run-level pairs (RLPs) for the compression of the quantization values. In [Gum98] it is shown that vector breakpoints have no advantage compared to scalar breakpoints and the limited number of RLPs results in lower quality.

Requantization repeats quantization ([Gum98]). The disadvantage of this approach is that the new quantization value has to be a multiple of the video's quantization value. Therefore, it is recommended to store the video having the value set to 1. Further, it is difficult to find 'good' quantization values.

Quality Metrics

Some quality metrics have been developed to evaluate the quality of videos by comparing a scaled version to the original:

PSNR (Peak Signal Noise Ratio, [BDT+96]) simply compares pixels:

$$\text{PSNR} = 10 * \lg(\frac{\Psi^2}{\sigma}) \qquad ; \qquad \sigma = \frac{1}{N}\sum_{i=1}^{N}(\sigma_i - \rho_i)^2,$$

where σ_i represents the value of the original image's i-th pixel and ρ_i the scaled image's i-th pixel. Ψ is the peak value for the pixels (e.g. 255 for 8 bits, 65535 for 16 bits etc.). This simple metric is widely used but describes the user's subjective quality very inaccurately.

ITS Quantitative Video Quality Metric is based on user ratings and gives an improvement to the PSNR results ([BDT+96]). But in [BDT+96] it is shown that this metric is inadequate for high-bandwidth video.

MPQM (Moving Pictures Quality Metric, [BDT+96]) is a more complex metric trying to incorporate the human visual system's limits. Therefore, the results are much closer to real users' ratings for a video.

DVQ (Digital Video Quality, [Wat98]) is another quality metric using a simplified model of the human visual system.

2.6.2 MPEG-2 Video

The MPEG-2 standard is an extension of MPEG-1 to provide higher quality at bandwidths from 2 to 15 MBit/s. The main extensions are more picture formats, some features to improve quality, some compression extensions and the *scalable coding extensions*:

The scalable coding extensions provide support to send high-quality and low-quality versions together in one stream (e.g. a standard TV version and a HDTV version): The transport stream is divided up into several layers. The first layer, called *base layer*, contains a low-quality version. The following layers (*enhancement layers*) provide additional data to generate a higher-quality version from the combination of base and enhancement layers. An example is shown in figure 2.20 (low-quality picture) and figure 2.21 (original picture[14]).

Four layering methods have been defined:

Temporal Scalability transmits a frame-rate reduced version in the base layer. The enhancement layers are used for additional frames resulting in a higher frame rate.

Spatial Scalability uses the base layer for low-resolution pictures and contains the missing data for higher resolutions in the extension layers.

[14]Picture source: `http://www.usrailroad.com`

Figure 2.20: Base layer only

Figure 2.21: Base and enhancement layer

SNR Scalability uses higher quantization for the base layer. The combination with the extension layers results in a lower quantization.

Data Partitioning transmits a subset of the DCT coefficients in the base layer and the remaining ones in the enhancement layers.

In combination with DiffServ, the base layer may be transmitted over high-quality but expensive classes and the enhancement layers over lower-quality but cheap ones (e.g. even best effort). In this case, the a low quality is guaranteed even if all enhancement data is lost.

2.6.3 H.263 Video

The H.263 standard defined in [H.263] is based on MPEG technology. Its main applications are video telephony and video conferences. Therefore, it is optimized for real-time compression and decompression and the usage of low-bandwidth links. Like MPEG, it uses DCT and motion compensation and the frame types I, P and B. The additional frame type PB contains a P-frame and a B-frame; the combined storage saves some bandwidth. H.263 also contains the scalability extensions explained for MPEG-2 (see section 2.6.2). Metrics and online scalability are the same as for MPEG-1 (see section 2.6.1). Since the H.263 standard is optimized for real-time encoding, scaling can also be done by re-encoding.

An important difference to MPEG is, that H.263 does not require a constant GoP. Picture types may be used as necessary, resulting in a lower bandwidth requirement. This especially affects the usage of I-frames: H.263 is mainly used for real-time video conferences. Therefore, fast forward and reward are impossible or unnecessary. In this case, it is possible to use an I-frame only for the first picture of a stream. As long as it is ensured that every part of the screen is transmitted **without** using prediction after some seconds within P or PB frames, there is no necessity to send I-frames. For example, P-frame #1 contains unpredicted blocks for the left upper part, P-frame #2 the same for the right lower part etc.. The result will be a smaller bandwidth requirement and a lower burstiness.

2.6.4 MP3 Audio

The MPEG-1/2 Layer-3 audio compression - better known as MP3 - has been developed to provide efficient compression of high-quality audio ([Bra99]). The decoded stream should be as close as possible to the original, at least for human listeners. The base for this is to use a perceptual model which incorporates the human ear's limits for removing unnecessary and redundant parts. Compression ratios of about 1:12 are usual for high-quality output.

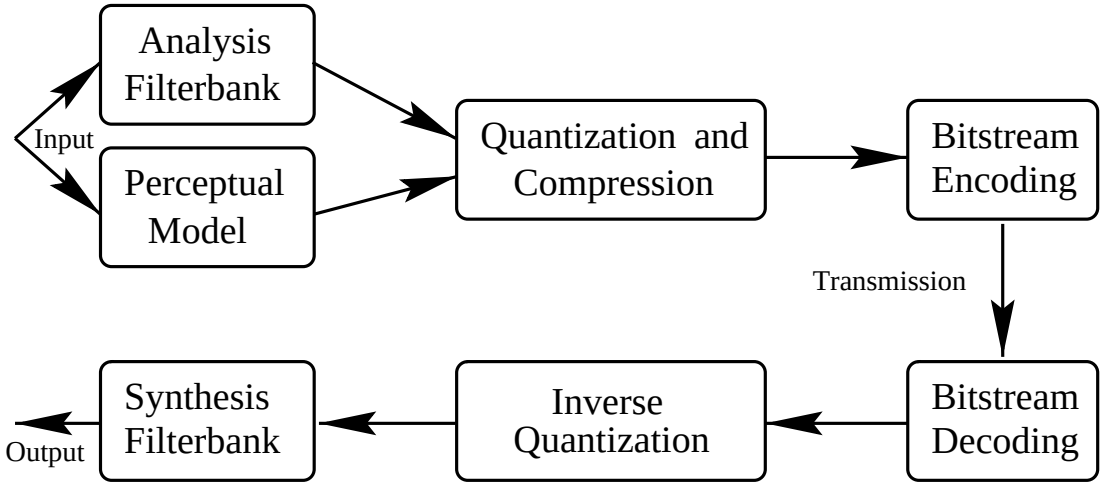


Figure 2.22: The MP3 codec

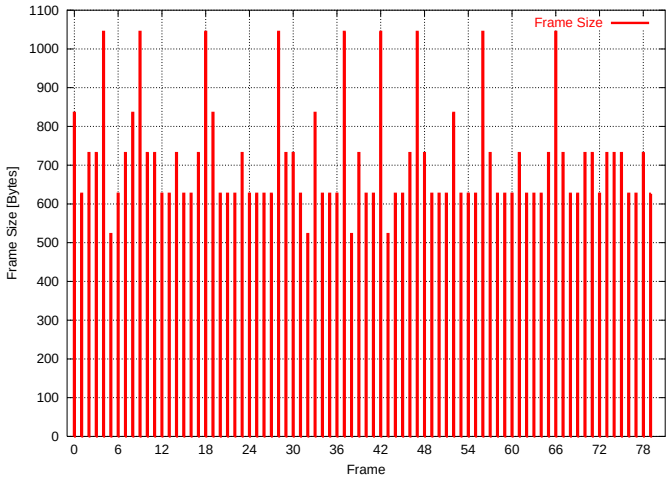


Figure 2.23: An MP3 Example (Go West)

The MP3 codec is shown in figure 2.22. First, it splits the input signal into its spectral components. Based on the perceptual model, the quantization is applied using the 'best' quantization value for each component. The quantized data is then compressed using Huffman codes.

To enhance compression ratios of stereo signals (two channels: left and right), the optional *joint stereo* mode uses a combined coding of both channels. The MP3 standard allows bitrate changes for every of the 38 frames per second. Possible bitrates range from 8 KBit/s to 320 KBit/s. But at the moment, most MP3 encoders only generate constant bitrate MP3 (CBR-MP3), that is a constant rate is used for the whole file. Since this may reduce quality for some parts and waste bandwidth for others, modern encoders like the Open Source encoder LAME (see [LAME]) can also generate variable bitrate MP3 (VBR-MP3). In this case, the different bitrates' quality for a certain frame has to be evaluated using the perceptual model, resulting in usage of the most efficient choice for encoding. For more details, see [LAME] and [TTB+98]. Figure 2.23 shows a part of the VBR-MP3 "*Go West*", generated using LAME.

Using a 500MHz Pentium III processor, it is possible to encode about three high-quality MP3 streams using the LAME encoder simultaneously. Therefore, online decoding and re-encoding to fit a given scaling is possible. Another way would be to drop some spectral parts of the stream.

Very detailed descriptions of audio quality metrics can be found in [TTB+98]. Well-known models described in this paper are NMR (Noise-to-Mask Ratio), Perceptual Audio Quality Measure (PAQM), Perceptual Evaluation (PERCEVAL), Perceptual Objective Measure (POM), Disturbance

Index (DIX), Objective Audio Signal Evaluation (OASE), Toolbox and PEAQ (Perceptual Evaluation of Audio Quality). The last one is based on the first six models and will become the future ITU standard for objective measurement of perceptual audio quality.

2.7 Summary

This chapter has given an introduction into the basics of this work. First, the network fundamentals are explained: The OSI and TCP/IP network reference models, the Internet protocols IPv4 and IPv6, the unreliable, connection-less datagram protocol UDP, the reliable, connection-oriented transmission protocol TCP, the control message protocols ICMPv4 and ICMPv6 and the real-time transport protocol framework RTP. This has been followed by the basics of Quality of Service (QoS) and its realizations: IntServ using per-flow reservations and DiffServ providing different classes, especially expedited forwarding (EF) and assured forwarding (AF). Further, traffic shaping has been explained. The next section has introduced the efficient description of variable bitrate traffic using approximations of the so called empirical envelope: The D-BIND and $(\vec{\sigma}, \vec{\rho})$ traffic model. Further, an algorithm for the cost-optimal, a priori calculation of bandwidth remapping intervals has been presented, based on traffic descriptions. In the following section, so called utility functions have been introduced for the evaluation of quality changes to the user satisfaction. This has been followed by some resource management definitions, especially the so called resource/utilization points and lists containing user satisfactions for given resource settings. Finally, the approximative algorithm ASRMD1 for mapping a resource (usually bandwidth) to different concurrent streams, based on resource/utilization lists for each stream, has been presented. The chapter has closed with a description of some standard video and audio formats: MPEG-1, MPEG-2 and H.263 for video and MP3 for audio. This has included encoding, decoding, scalability and quality metrics.

Chapter 3

The CORAL Project and the Goals of This Work

This chapter gives an introduction to the CORAL project's concept. Further, a short overview of an implementation example is shown: The RTP AUDIO system. The necessary enhancements to this system for efficient support of variable bitrate streams finally leads to the goals of this work.

3.1 The CORAL Concept

CORAL is the abbrevation for COmmunication Protocols for Real-time Access to digital Libraries ([AKM+00]). The goal of the CORAL project is the development of real-time protocols to transmit scalable streams of different multimedia standards for audio and video (e.g. MPEG-1/2, H.263, MP3, ...) over a network.

The concept can be found in figure 3.1. Since real-time multimedia transmission requires QoS guarantees, it is necessary to provide reservation mechanisms. This is done in the *Reservation Module*. The transport module can mark each packet to belong to a reserved flow (IntServ, see section 2.2.2) or class (DiffServ, see section 2.2.3) using e.g. the IPv6 flow label or the *Type of Service* or *Traffic Class* field of the IP header.

Since multimedia transmissions usually require transmission of different media types to a single user (e.g. video and audio, hence the name multimedia), several streams to the same client are grouped to a so called *session*. It consists of one or more *application streams*, all having their own *stream priority*. An example of a session for a virtual shopping mall can be found in table 3.1.

To provide efficient usage of reservation, CORAL uses the concept of so called *layered transmission*: A stream (e.g. a video) can be splitted into several substreams called *layers* by the *Layering Control*. Each layer has got its own QoS requirements. For example using MPEG-2, the base layer can be transmitted via a DiffServ class like EF while the enhancement layers use best effort or MPEG-

Stream	Usage	Type	Priority
Stream #1	Interactive 3D scenario	3D	High
Stream #2	Background music	MP3	Low
Stream #3	Commercial video (background)	H.263	Lowest
Stream #4	Commercial video (foreground)	MPEG-2	Normal
Stream #5	Sound of the commercial video	MP3	Normal
Stream #6	Information video for selected product	MPEG	Highest
Stream #7	Sound for the product's video	MP3	High

Table 3.1: An example session for a virtual shopping mall

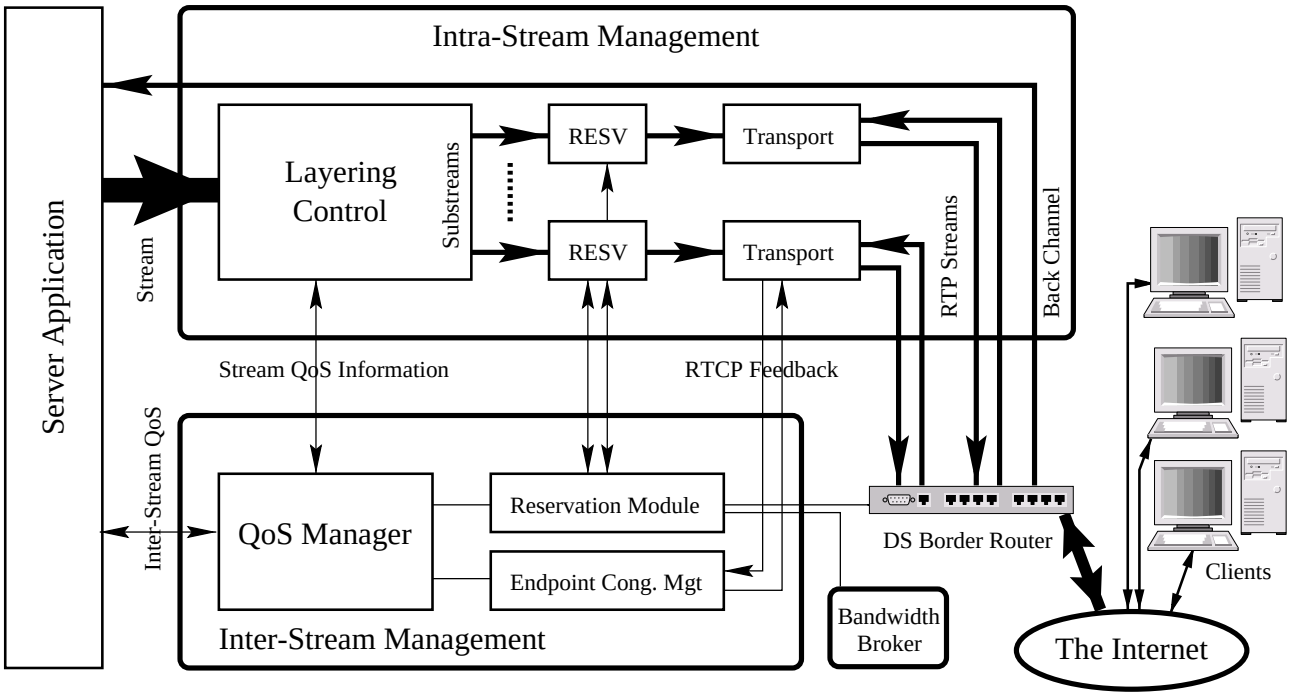


Figure 3.1: The CORAL concept

1 video streams can transmit I-frames over a low-loss but expensive and P- and B-frames over cheap but more lossy DiffServ classes. The complete stream hierarchy of the CORAL concept can be found in figure 3.2.

The available bandwidth of the *Reservation Module*'s SLA is managed by the *QoS Manager*. It maps the bandwidth to the different streams, accordingly to their stream priority and session's priority. Each session has got a constraint for minimum and maximum bandwidth used by the streams belonging to the session. This ensures that at least a minimum quality is possible and that the total bandwidth (e.g. only 1 MBit/s link speed) and cost is limited. Further, the mapping has to fit the streams' QoS requirements for maximum delay, loss rate and jitter (see section 2.2.1). The server's SLA may be dynamic, that is it can be changed. For example, the DiffServ link is shared between several servers. If one server has got lots of clients but a second one only a few, the first server's SLA may be increased while the second one's may be decreased. These changes are managed by the *Bandwidth Broker* (BB). For more details, see [Sel01]. The CORAL bandwidth pricing concept is

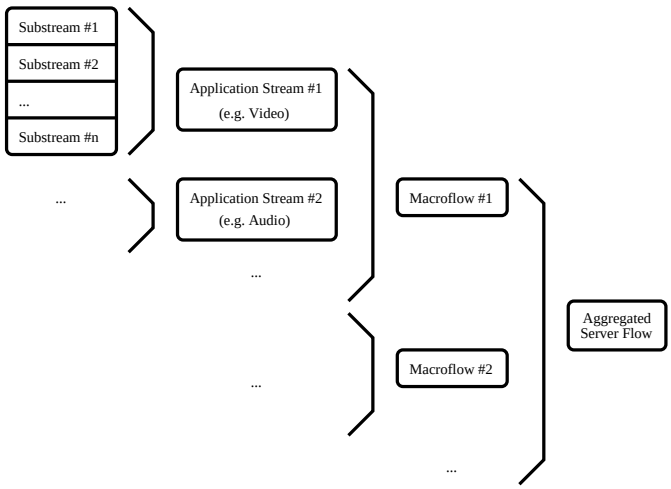


Figure 3.2: The stream hierarchy of the CORAL concept

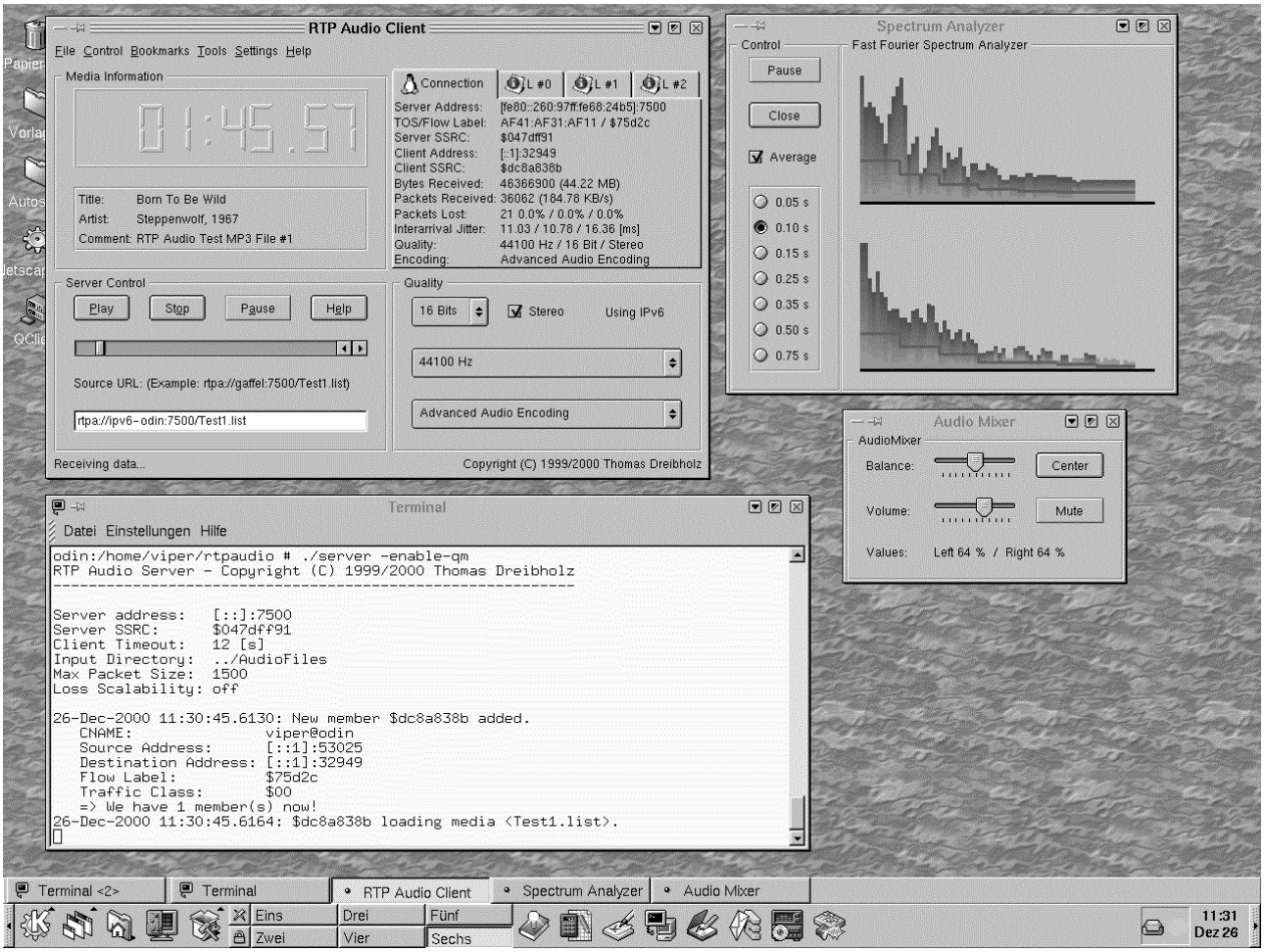


Figure 3.3: The RTP AUDIO system

described in [Rad01], it shows three types of charges:

1. Holding Charge: The price for reserving a given bandwidth of DiffServ class n (but not for its usage!).
2. Usage Charge: The price for sending a given volume via DiffServ class n.
3. Bandwidth Change Charge: The charge for renegotiating the DiffServ class n's SLA via the bandwidth broker.

The *Endpoint Congestion Management* (ECM) is necessary to provide *TCP-friendly* behavior of streams sent over best effort service. That is, the streams should behave like TCP streams (see section 2.1.3) and adapt bandwidth to the network's congestion. More details about the ECM can be found in [Kar01].

3.2 RTP AUDIO - A CORAL Implementation Example

One example implementation of the CORAL concept is the RTP AUDIO system ([DSV00], [AKM+00] and [RTP Audio]). Figure 3.3 shows the RTP AUDIO server and a client. This system transmits uncompressed audio using RTP (see section 2.1.5) based on UDP (see section 2.1.3) over IPv4 and IPv6 (see section 2.1.2) using DiffServ (see section 2.2.3). The following audio qualities are supported:

- Sampling rates from 4410 Hz to 44,100 Hz in steps of 2205 Hz,

- 4, 8, 12 or 16 bits per sample and
- mono or stereo.

23 constant levels have been chosen from these 152 possible settings for the QoS manager to select the scaling. This uncompressed transmission results in **constant bitrates** from about 4 KBytes/s to about 185 KBytes/s. The stream itself is divided into up to three layers having ascending maximum acceptable loss rates:

1. The upper 8 bits of the left channel,
2. the upper 8 bits of the right channel (in stereo mode only) and
3. the lower bits of both channels (in 12/16 bit mode only).

In case of packet losses, the left or right channels' bits can be replaced by the corresponding bits of the other channel resulting in mono quality. Lost packets of lower bits result in 8-bit quality - these bits are simply set to 0.

Since DiffServ only guarantees bandwidths and not maximum delay, loss rate or jitter, the QoS manager checks these values by analyzing the RTCP receiver reports (see section 2.1.5) and doing round trip time measurements for each class. If necessary, layers are mapped to other classes or streams are scaled down. The round trip time measurement uses ICMP echo requests and replies (see section 2.1.4) sent over every available DiffServ class to the destination host. This is necessary since the round trip times of **all** possible classes are required. Echo replies are sent back via BE. Therefore, the transfer delay of Class_x is not simply $\frac{\text{RTT}_{\text{Class}_x}}{2}$. Instead, the following formula is used:

$$\text{Delay}_{\text{Class}_x} := \text{RTT}_{\text{Class}_x} - \frac{\text{RTT}_{\text{BestEffort}}}{2}.$$

Since round trip time, jitter and transfer delay are slightly varying, it is useful to smooth their values, that is incorporating the former value into the result. Therefore:

$$\text{Value}_{\text{New}} := \alpha * \text{Value}_{\text{Old}} + (1 - \alpha) * \text{Measurement}.$$

A useful value for the constant α is $\frac{7}{8}$. See also [DSV00] for details.

3.3 The Goals of This Work

The measurements in [DSV00] show that the RTP AUDIO system works quite well for constant bitrate streams. But since many compressed multimedia standards like MPEG (see section 2.6.1 and 2.6.2), H.263 (see section 2.6.3) and MP3 (see section 2.6.4) use or may use variable bitrates, some extensions are necessary: Simply using the peak rate as bandwidth is far too inefficient (see section 2.3). A more detailed traffic description and bandwidth remapping is therefore necessary.

The CORAL project's subject are digital libraries. Therefore, most medias like video and audio files are already completely stored. Therefore, it is obvious to use a traffic model from section 2.3.3 and the algorithm described in section 2.4 to calculate efficient remapping intervals and traffic descriptions a priori. But since the described algorithm does not support layered transmission and scalable medias, some extensions are necessary. This will be described in chapter 4. Of course, for real-time video and audio conferences for example, an a priori analyzation is not possible. In this case, an online analyzation is necessary. But this is not a goal of this work. For details about this, see [BCC+99] and [Vey01].

Since the medias are scalable, it is necessary to evaluate the effects of scaling to the user satisfaction. This can be done using the resource/utilization lists described in section 2.5.3. Based on

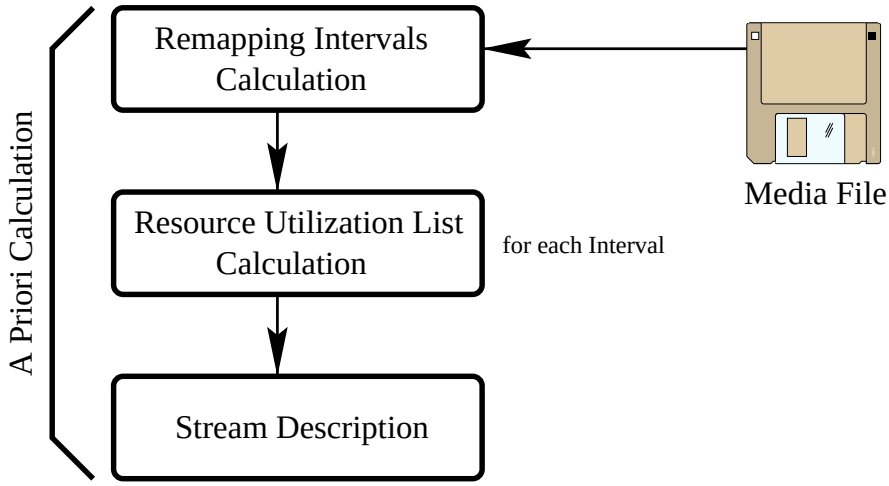


Figure 3.4: The offline part

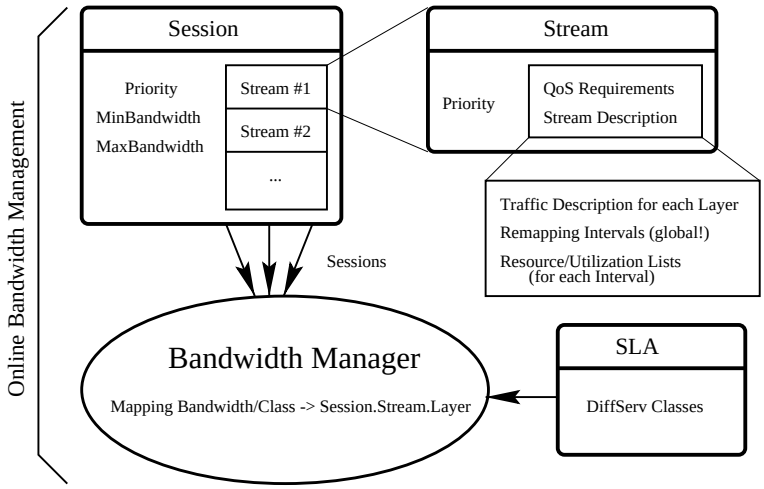


Figure 3.5: The online part

the a priori calculated remapping intervals and traffic descriptions, it is now necessary to develop an algorithm to calculate 'useful' resource/utilization lists. This is possible a priori, too. Details can be found in chapter 5.

Finally, in chapter 6, the online bandwidth management has to be developed: The SLA's bandwidth has to be mapped online to different streams, based on the a priori calculated information. As it is shown in section 2.3.1, buffering may result in a huge bandwidth gain. Since the complete traffic description is known from the a priori calculation, it is useful to optimize the buffer delay here: The user has got a certain maximum transfer delay limit, each class's current delay is known from the ICMP measurements as shown in section 3.2. Now, it is possible to check the resulting cost for every class and use the cheapest transport possibility. For example, the example of section 2.3.1 requires 100% bandwidth for a buffer delay of 1 frame but only 40% for a buffer delay of 6 frames. Therefore, if class #1 permits only a buffer delay of 1 frame and class #2 a buffer delay of 6 frames (due to a lower transfer delay) but at double cost, it is still cheaper to use the expensive class #2 - since it requires only 40% of the original bandwidth to be reserved.

Further, the bandwidth management has to support sessions as described in section 3.1. Within a session, the bandwidth mapping should be utility-fair (see section 2.5.3): A user having two equal-prioritized streams in the same session wants both having the same user satisfaction - not one stream in best and the other in lowest quality. But globally, it is useful for the provider to have as much sessions as possible getting a high quality. For example, it is not useful if user #1 has got a large

session of e.g. 80 MBit/s at 25% utilization and users #2 to #21 have got small sessions of 1 MBit/s at 25%, too. Instead, it is recommended to use an unfair sharing here and give user #1 e.g. 50 MBit/s at 10% utilization and users #2 to #21 e.g. 95% at 2.5 MBit/s.

As described, this work consists of two parts:

1. The a priori calculations of remapping intervals, traffic descriptions and resource/utilization lists and
2. the online bandwidth management.

A graphical view of the a priori part can be found in figure 3.4, the online part is shown in figure 3.5.

3.4 Summary

This chapter has given an introduction to the CORAL project's concept. Further, a short overview of an implementation example has been shown: The RTP AUDIO system. The necessary enhancements to this system for efficient support of variable bitrate streams has finally leaded to the goals of this work. These can be summarized as follows:

1. The offline part:
   (a) Extensions of the a priori remapping interval calculation algorithm by support for layered transmission and scalable medias.
   (b) Development of an algorithm to calculate 'useful' resource/utilization lists, also a priori.
2. The online part - the bandwidth management:
   - Minimization of bandwidth cost by the usage of cost-optimized buffering.
   - Support for sessions and fair bandwidth distribution within the sessions.
   - Maximization of the global utilization.

Chapter 4

The A Priori Remapping Interval Calculation

In this chapter, the a priori remapping interval calculation algorithm, presented in section 2.4, is extended to support layered transmission as described in section 3.1. Further, an efficient support for scalability is required. Therefore, additions to provide frame rate scalability and a method to allow frame size scalability of the traffic description are developed. The chapter closes with optimizations for runtime and storage space.

4.1 The Remapping Interval Algorithm Basics

First of all, a traffic model has to be chosen to store the traffic descriptions (see section 2.3.3). Since the traffic shaper, which is necessary for the buffering, is implemented in software (see also system description in section 7.3), the D-BIND model has been chosen. The reasons for this decision are:

- D-BIND provides a more accurate description, due to its non-convexity.
- It is not necessary to limit the leaky bucket's size. This is only required for traffic shaping using hardware (e.g. a network card or router) having a built-in leaky bucket of fixed size. Instead, the server's main memory is used for buffering. This is available at a sufficient amount[1].

Next, cost functions for a given D-BIND traffic description and the remapping have to be defined. The cost for the bandwidth has been defined as follows:

$$\text{cost}_{\text{Bandwidth}}(\overrightarrow{t}) := \frac{\sum_{i=1}^{n} t_i.\text{Bandwidth}}{n}.$$

The bandwidths of every D-BIND pair i are added and divided by the total number of pairs. This cost function is therefore the average bandwidth for all delays. It has got the following properties:

- If the bandwidth requirements are constant for all buffer delays ($b(t) = \text{const}$, see section 2.3.1), then the function represents the cost of this constant bandwidth.
- On the other side, for $b(t)$ decreasing with increasing delay the cost-advantage of buffering is also included: It will be the average bandwidth cost for the given delays.

Another cost function would be weighting some of the delays and therefore emphasizing lower or higher delays. But since the buffer delay results from the difference between the network's varying transfer delay and the user's maximum delay requirements, it would be difficult to give useful delay weights here. For this reason, this approach has not been used.

[1]The number of streams and the buffer delay is limited. Care has been taken that even in the worst-case buffering scenario, the server's main memory of 256 MBytes will not be exceeded.

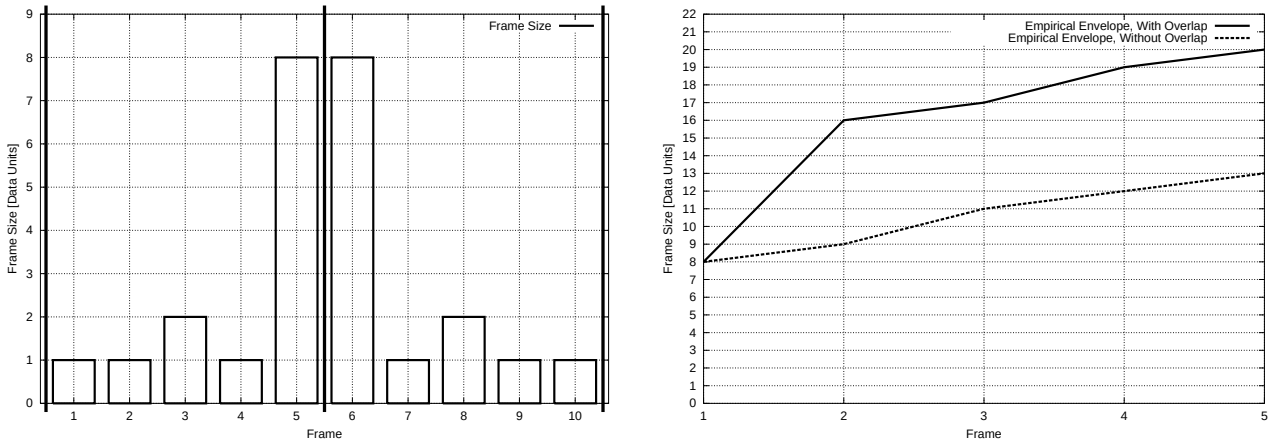


Figure 4.1: The comparision of overlapping and non-overlapping intervals

Since remapping using DiffServ only consists of setting the *Type of Service*/*Traffic Class* field of the IP header to another value (see section 2.2.3), the only work to do is the computation of a new remapping. Therefore, a constant value is used as remapping cost. This is also recommended in [Gum98] on page 67.

It should be denoted here, that the traffic description has to be calculated using overlapping intervals: For a delay of $n \geq 2$, it is necessary to include the next interval's $n-1$ frames into the traffic description calculation, too.

An example stream consisting of two intervals is shown on the left side of figure 4.1. Interval I1 contains frames #1 to #5 and interval I2 frames #6 to #10. Using each interval's own frames only, the calculated empirical envelope results in the lower graph on the right side of figure 4.1. The upper graph shows the overlapping calculation's result.

Now, using a delay of 3 frames, the non-overlapping calculation requires a reservation of $\frac{11}{3} = 3.67$ bandwidth units per frame. But sending frames #5, #6 and #7 in a sequence results in exceeding the delay constraint for the leaky bucket (see section 2.2.4): After buffering frame #5, the buffer contains 8 units. 3.67 are sent until frame #6 is buffered. Then, 12.33 units are in the buffer which would require a time of $\frac{12.33}{3.67} = 3.36$ frames to be sent; this is more than the delay limit of 3 frames. Using the overlapping calculation, the reserved bandwidth $\frac{17}{3} = 5.67$ ensures this contraint: $\frac{(8-5.67)+8}{5.67} = 1.83$ frames.

4.2 Layering of the Media Types

Now, the remapping interval calculation has to be extended to support layered transmission. But first, the layering has to be described for the media types of section 2.6. In this work, only traces of the different media types are used. Therefore, it is not possible to examine e.g. application QoS dimensions like picture size, colors or audio sampling rate. Instead, the generic dimensions *frame rate* and *frame size* are used. In a real media transport scenario, an additional mapping between these two values and the media-specific dimensions is required.

For MPEG-1/2, it is recommended to use an own layer for each frame type since each one has got its own QoS requirement for the maximum acceptable loss rate (see section 2.6.1). Therefore, MPEG-1 streams have got three layers: One for I-, P- and B-frames.

An example for the first 25 frames of the MPEG-1 video "*Terminator II*"[2] can be found in table 4.1. It uses 25 frames per second and the GoP "IBBPBBPBBPBB", each table entry contains the

[2]Trace source: [Wür95], terminator.tar.gz.

Frame	Layer #0 (I)	Layer #1 (P)	Layer #2 (B)
#00	33712	0	0
#01	0	0	4224
#02	0	0	4104
#03	0	11688	0
#04	0	0	4424
#05	0	0	4256
#06	0	9896	0
#07	0	0	4184
#08	0	0	4432
#09	0	11168	0
#10	0	0	4680
#11	0	0	4808
#12	36752	0	0
#13	0	0	4704
#14	0	0	6376
#15	0	24384	0
#16	0	0	3096
#17	0	0	3424
#18	0	6800	0
#19	0	0	3096
#20	0	0	3376
#21	0	7736	0
#22	0	0	4016
#23	0	0	7072
#24	30160	0	0

Table 4.1: 25 frames of the MPEG-1 video "*Terminator II*"

frame size. I-frames are sent in layer #1 every 12th frame, P-frames are sent in layer #2 and B-frames are sent in layer #3. It is important to denote here, that as it is shown in the table, a lot of entries are set to zero. That is, no transmission is done in the corresponding layer for the current frame. For I-frames, there is a gap of 11 and for P-frames a gap of at least 2 frames between every transmission. Therefore, a buffer delay of two or more frames will result in much lower bandwidth requirements!

Using MPEG-2, additional layers for every frame type of the MPEG-2 enhancement layers can be added: For example using an MPEG-2 stream having one base and one enhancement layer, six layers can be used: Three for I-, P- and B-frames of the base layer and three for the same types of the enhancement layer. The same scheme can also be applied to H.263. But in this case, up to four layers may be necessary: Three for I-, P- and B- frames and the fourth for PB-frames.

MP3 only consists of rather small frame sizes, usually about 600 to 700 bytes for high-quality audio. Therefore, it does not make much sense to use layering here because each packet requires transport headers: IP (40 bytes for IPv6) + UDP (8 bytes) + RTP (at least 12 bytes) + finally a header for the media itself (e.g. 20 bytes). In this example, each packet contains 80 bytes overhead which results in 3040 bytes/s for 38 frames per second (MP3 frame rate). Even for high-quality MP3 transport at about 100 KBit/s = 12500 bytes/s, there would be no significant bandwidth gain due to this header overhead.

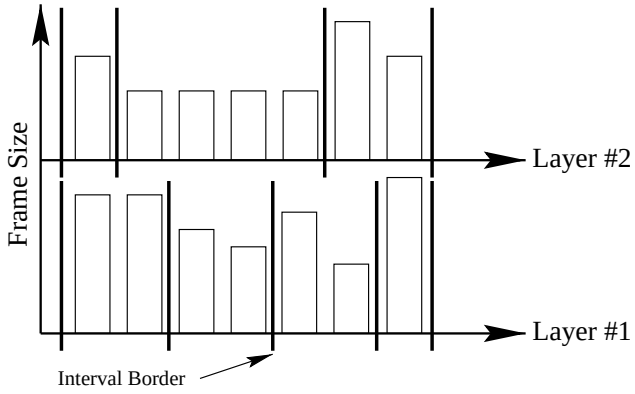


Figure 4.2: Per-layer intervals

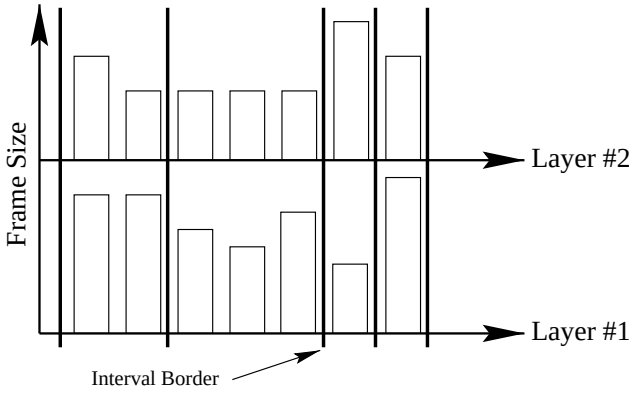


Figure 4.3: Per-stream intervals

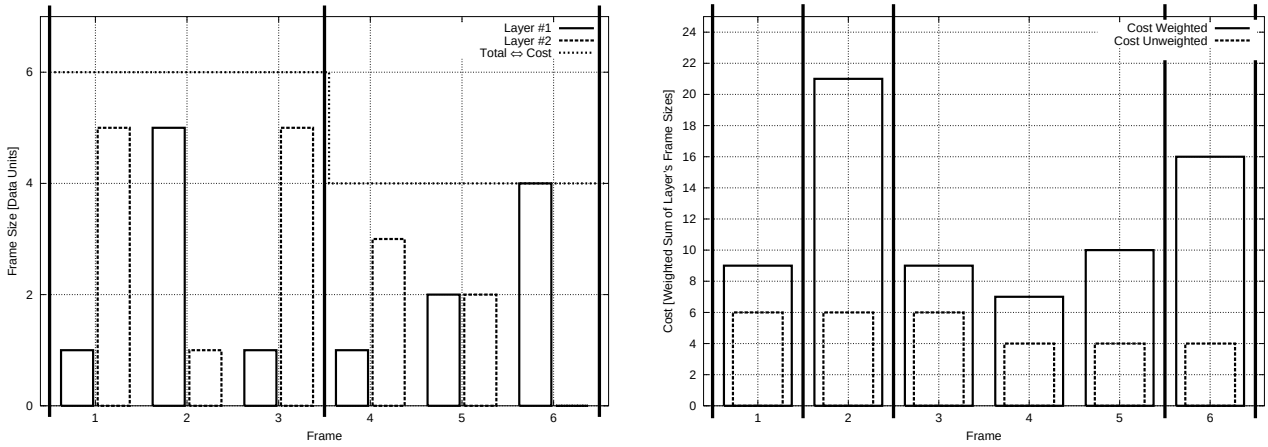


Figure 4.4: Remapping intervals calculated using unweighted and weighted cost

4.3 Extension for Layered Transmission

Using layered transmission, the independent traffic descriptions have to be calculated for each layer. Now, it would be possible to do an independent remapping interval calculation for each layer, too. But for streams containing many layers (e.g. 9 for MPEG-2 having 3 MPEG layers and one transport layer for each frame type), this would be very inefficient, since each layer would require remappings independently. Therefore, instead of using *per-layer intervals*, *per-stream intervals* are used: The interval borders are the same for all layers, which implies remappings for all layers simultaneously. An illustration of both methods is shown in figure 4.2 (per-layer intervals) and figure 4.3 (per-stream intervals).

It is very important to denote here, that at time of the a priori remapping interval calculation, the layers' later online mapping to DiffServ classes is unknown! But every layer has got its own known QoS requirements and burstiness (a higher buffering gain may be reached by faster network transmission and a higher possible buffer delay), that is some layers will usually require more expensive DiffServ classes than others. Therefore, the cost calculation should use a weighting for each layer: A bandwidth change in an 'expensive' layer should affect the cost function more than a change in a 'cheap' one. This can be achieved by a weighted sum:

$$\text{cost}_{\text{Bandwidth}}(\overrightarrow{t}^{L_1}, \ldots, \overrightarrow{t}^{L_m}) := \sum_{i=1}^{m} \left[\omega_i * \text{cost}_{\text{Bandwidth}}(\overrightarrow{t}^{L_i})\right],$$

where $\text{cost}_{\text{Bandwidth}}(t)$ denotes the cost function introduced in section 4.1, $\overrightarrow{t}^{L_i}$ the traffic description

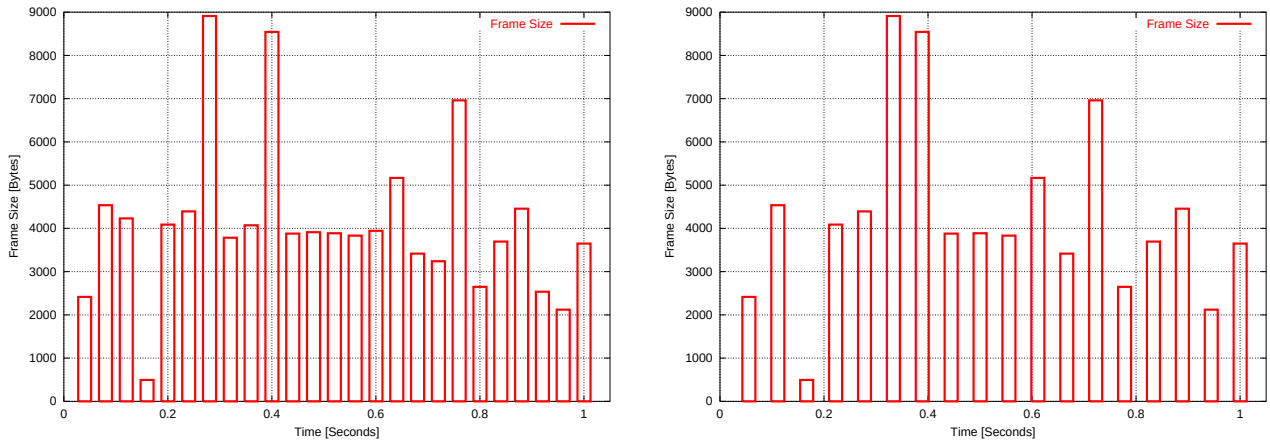


Figure 4.5: A frame rate scalability example

of layer i for the current interval and ω_i the weight of layer i.

A 2-layered example stream can be found on the left side of figure 4.4. It also shows the sum of both layers - this corresponds to the cost in the unweighted case. Now for simplification, only a buffer delay of one frame is used in this example. Since the cost is 6 cost units/frame for frame #1 to #3 and 4 cost units/frame for frame #4 to #6, the unweighted algorithm creates only two intervals: Frame #1 to #3 and frame #4 to #6.

But if layer #1 is expensive and layer #2 is cheap refering to their QoS requirements, the generated intervals are very inefficient: For example, only for frame #2, layer #1 requires a bandwidth of 5 bandwidth units per frame; for frame #1 and #3 it is only 1 bandwidth unit per frame. This over-provisioning can be avoided using for example a weight of 4 for layer #1 and 1 for layer #2. This calculation results in the right side of figure 4.4. Here, the bars show the weighted and unweighted cost per frame for comparision. Note, that due to the buffer delay setting of only one frame, the peak frame cost has to be used for the third interval (see traffic description basics in section 2.3.1 for details). Now, layer #1 dominates the cost. This results in four intervals, giving frame #2 its own interval. Therefore, bandwidth will saved at the cost of more remappings.

A method for finding 'good' weights would be to guess test values based on the layers' QoS requirements and burstiness and calculate a sequence of test intervals. Using this test version, some remappings using a real DiffServ SLA can be simulated. Then, the weights can be adapted to the simulation results. An example simulation with a detailed description can be found in section 8.1.2.

4.4 Extension for Scalable Media Types

Now, the remapping interval calculation has to be extended to support scalable media types. The basis for scaling is to reduce the frame rate and/or frame size of the original media to a lower-quality version. First, frame rate scalability will be examined.

4.4.1 Frame Rate Scalability

Scaling of the frame rate consists of dropping some of the frames (e.g. video: 30 frames/s -> 20 frames/s) or partitioning of a quality-reduced version (see section 4.4.2) into fewer frames. Since the traffic description for frame rate n does not contain any information about frame rate $n-1$, the remapping interval calculation described in section 4.1 and 4.3 has to be computed for every frame rate to be supported.

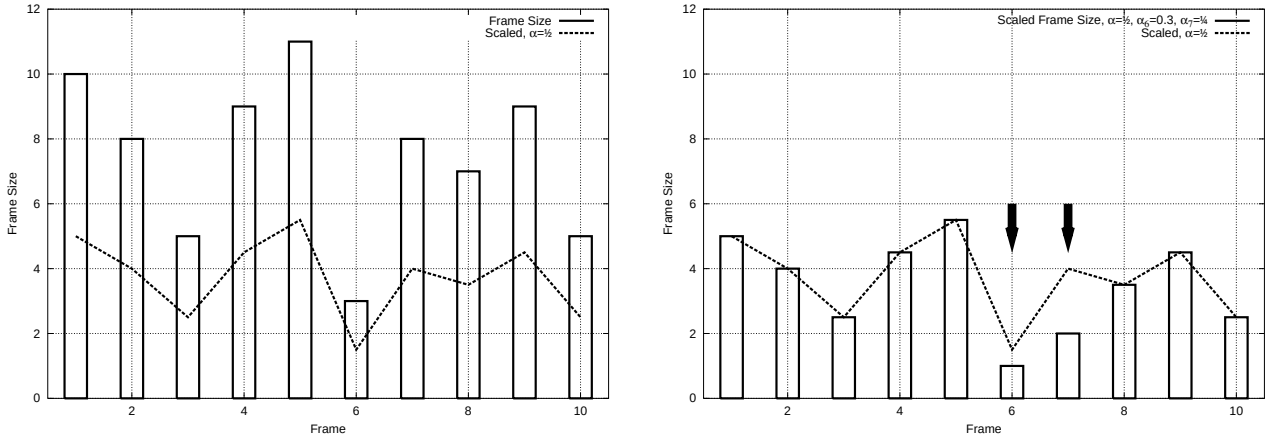


Figure 4.6: A frame size scalability example

Using MPEG-1/2 video, the frame rate can be reduced by first removing a given number of B-frames per second since this type is not referenced by any other frames. If there are no more B-frames, P-frames have to be removed first. In this case, care has to be taken not to remove P-frames referenced by other P-frames. Finally, if there are no more P-frames, I-frames can be removed. The frames can be dropped either randomly or selective using e.g. one of the quality metrics described in section 2.6.1 to minimize the quality reduction. This scheme can also be applied to H.263 handling PB-frames equal to P-frames.

MP3 medias have got a constant frame rate of 38 frames/s. Therefore, frame rate scalability by dropping frames is not supported here. Further, joining frames which would reduce the header overhead would result in much worse quality in case of packet losses: At 38 frames per second, a missing packet of $\frac{1}{38}$ second would hardly be perceptible. But joining frames to e.g. generate maximum-length FDDI packets of 4500 bytes for a 100 KBit/s = 12,500 Bytes/s high-quality MP3 stream would result in a frame rate of 3 frames per second. A single lost packet would therefore cause a significant gap of $\frac{1}{3}$ second.

An example stream having a frame rate of 25 frames per second can be found on the left side of figure 4.5. The same stream scaled to 18 frames per second by dropping frames 3, 8, 9, 12, 16, 18 and 23 is shown on the right side.

4.4.2 Frame Size Scalability

The second scaling method is to reduce the frame size. For MPEG-1/2 and H.263 video, this can for example be done by block dropping, coefficient elimination, etc. as described in section 2.6.1. If all frames are scaled by a constant factor α, then the empirical envelope and therefore the traffic descriptions (see section 2.3.1) are scalable, too:

$$\text{FrameSize}_{\text{New}} = \alpha * \text{FrameSize}_{\text{Old}},$$

$$E^*(t) = \sup_{\tau>0} [\alpha * A[\tau, \tau + t]] = \alpha * [\sup_{\tau>0} A[\tau, \tau + t]] \qquad \forall t > 0,$$

where t denotes the buffer delay. It should be denoted here, that it is not necessary to scale all frames using exactly factor α: Some frames may also be scaled using factors α_i as long as $\alpha_i \leq \alpha$ for all i. In this case,

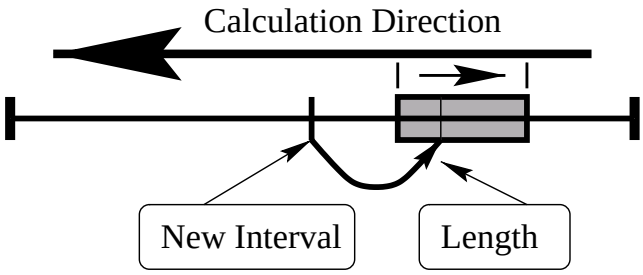


Figure 4.7: The runtime-optimized remapping interval calculation

$$\text{FrameSize}_{\text{New}} \leq \alpha * \text{FrameSize}_{\text{Old}}$$

and therefore

$$E^*(t) \geq \sup_{\tau>0} A_{\text{Scaled}}[\tau, \tau + t] \qquad \forall t > 0.$$

The scaled empirical envelope remains to be a valid traffic constraint but the higher the difference the higher the over-provisioning due to the introduced inaccuracy. If some frame types have equal QoS requirements and burstiness but different scale factors (e.g. due to different scaling algorithms), own layers may be useful for such types to reduce this inaccuracy.
An example stream can be found on the left side of figure 4.6. The stream's frame sizes have to be scaled by scale factor $\alpha = \frac{1}{2}$, the new frame sizes are marked by the line. The right side shows the result but having used $\alpha_6 = \frac{1}{3}$ and $\alpha_7 = \frac{1}{4}$ to scale the frames #6 and #7 (see arrows). Again, the line shows the sizes for a scale factor $\alpha = 0.5$ for comparision.

Since the traffic description is scalable, only the original version has to be stored. Frame size scaled versions can simply be calculated online.

4.5 Runtime Optimization

The remapping interval algorithm from section 2.4 has been extended to support layered transmission and scalability. Now, it is necessary to do some optimizations.

First, its runtime of $O(n^3)$ should be improved. As it will be shown in chapter 6, a regular remapping is required in an interval of some seconds anyway. Therefore, large intervals of e.g. several minutes result in no significant improvement. But limiting the interval to a constant range of e.g. 2 seconds to 30 seconds greatly improves the runtime for the interval calculation: $O(n^2)$ instead of $O(n^3)$. A graphical representation can be found in figure 4.7. Only the range marked by the grey box has to be checked for the cheapest length. See also figure 2.9 for comparision to the original algorithm.

4.6 Space Optimization

Next, the algorithm with its extensions requires enormous disk space to store the calculated intervals and its traffic descriptions. For example in an MPEG-2 video of 90 minutes at 30 frames per second, there are 30 different frame rates (1 frame/s to 30 frame/s in steps of 1 frame/s) and a total of six transport layers: I, P and B for the base MPEG-layer and I, P and B for the enhancement MPEG-layer. 90 minutes at 30 frames per second is equal to 162,000 frames. The interval calculation is done for each of the 30 frame rates. Each calculation generates the interval length for each frame

Frame	Interval Length	Traffic Description	Interval
#00	**3**	**D-BIND for each Layer**	**I01**
#01	2	D-BIND for each Layer	*
#02	7	D-BIND for each Layer	*
#03	**2**	**D-BIND for each Layer**	**I02**
#04	5	D-BIND for each Layer	*
#05	**9**	**D-BIND for each Layer**	**I03**
#06	8	D-BIND for each Layer	*
...	...	...	...

Table 4.2: Example of the space optimization

Interval	First Frame	Last Frame	Traffic Description
I01	#00	#02	D-BIND for each Layer
I02	#03	#04	D-BIND for each Layer
I03	#05	#14	D-BIND for each Layer
...	...	...	...

Table 4.3: The result of the space optimization

and therefore a traffic description for each of the 6 layers. The total number of traffic descriptions is therefore:

$$6 * \sum_{i=1}^{30} \left[(1 - \frac{(i-1)}{30}) * 162,000 \right] = 15,066,000$$

Assuming 12 D-BIND points for each description with 2 bytes for length and 4 bytes for bandwidth, the required storage space is 15,066,000*12*6 Bytes $\approx$ 1 GByte! Finally, some more space is required to store the resource/utilization lists - one for each interval start, therefore 162.000 (see chapter 5 for details) - and management information like indices. Under the assumpton of up to 32 resource/utilization points per list, this makes about 200 additional MBytes and 1.2 GBytes total. For comparision, an average bitrate of 5 MBit/s results in a size of about 3.143 GBytes for the complete 90-minutes video. This is only about 2.5 times more than the required management information.

Obviously, this space requirement is far too high. The problem of the interval calculation is that the next interval and therefore its traffic description is stored for **every** frame. This has the advantage that moving to any frame of the media results in starting with the optimal[3] interval.

A simple but effective optimization is to store only an *interval path* from frame 0 to the end. This is illustrated in figure 4.8. The storage starts at frame 0 and consists of the length and traffic description for the first interval. If e.g. the length is 250 frames, the next interval to store starts at frame 250. If this interval's length is e.g. 300, the next one will be 550 and so on. An example can be found in table 4.2. The bold-printed rows show the descriptions to be stored, all other rows are skipped. Table 4.3 contains the resulting data to store. Practical experience shows, that the space reduction is usually about two orders of magnitude, depending on the average interval length.

Movements within the media now result in jumping into an interval instead of starting at its begin (see frame #01 in table 4.3: Interval I01). But it is important to denote here, that this disadvantage remains only for the first interval after a movement. At the begin of the following interval, playing will be again on the *optimal path* (see frame #03 in table 4.3: Interval I02 starts here). A normal interval length is usually about 3 to 30 seconds. Lower values cause too many remappings and higher

[3] *Optimal* refers to the property of the algorithm presented in section 2.4: The intervals are optimal refering to the given cost for remapping and bandwidth.

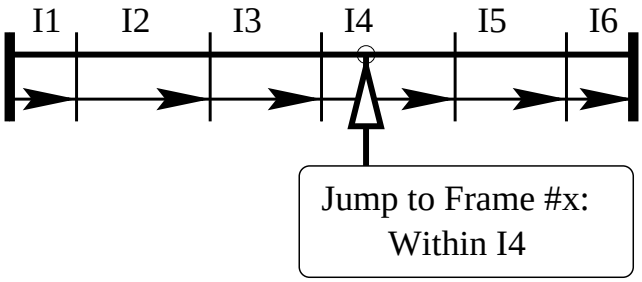


Figure 4.8: The space-optimized remapping interval calculation

values are not necessary since a regular remapping is necessary anyway (details about this will be shown later in chaper 6). Therefore, the non-optimal traffic description lasts for a few seconds only.

4.7 Summary

In this chapter, the a priori algorithm of section 2.4 has been extended to support layered transmission using an own D-BIND traffic description for each layer and a weighted sum for the total cost. This weighting has improved the cost function to be more affected by bandwidth requirement changes of more expensive layers, resulting in more cost-efficient bandwidth remappings. Further, scalability support has been added:

- Different frame rates require their own remapping interval lists, since the traffic description can usually not be derived from another frame rate scaling's description.
- Using a constant scale factor α for frame size scaling, the traffic description is also scalable. Therefore, here is no need for additional traffic descriptions.

Next, the algorithm's runtime has been optimized using a constant interval range, resulting in an improvement to $O(n^2)$ instead of $O(n^3)$. Finally, an efficient storage optimization has been developed: A reduction of usually about two orders of magnitude can be achieved by storing only the so called *interval path* instead of the complete traffic description beginning at every frame.

Chapter 5

The A Priori Resource/Utilization List Calculation

As described in section 2.5.4, an ASRMD1-based algorithm requires so called resource/utilization lists for each stream to calculate a bandwidth distribution. Therefore, an a priori algorithm for the calculation of 'useful' resource/utilization lists, based on the a priori calculated remapping intervals and traffic descriptions, is developed in this chapter.

5.1 Resource/Utilization Basics

First of all, it is necessary to examine the remapping intervals: As described in section 4.4.1, there is an independent list for each supported frame rate. Now, a resource/utilization list for every interval start is required. Since each list contains points of different frame rates, it is not useful to calculate a resource/utilization list for every remapping interval of any frame rate's list. This would cause redundancy. Instead, a global enumeration for the frames can be used, which is valid for **all** frames rates: This will be called *position* and may for example be a timestamp in microseconds. Frame numbers (called *frame positions* in this context) and positions can simply be translated using the following formulas:

$$\text{FramePosition}_{\text{FrameRate}} = \text{round}\left(\frac{\text{Position} * \text{FrameRate}}{\text{PositionStepsPerSecond}}\right), \tag{5.1}$$

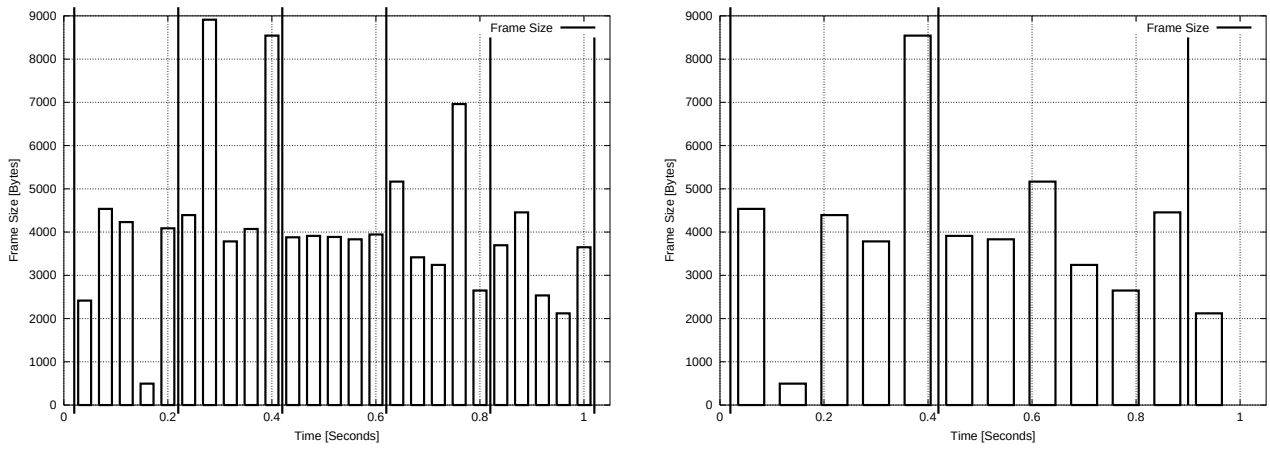


Figure 5.1: Remapping intervals for different frame rates

$$\text{Position} = \text{round}\left(\frac{\text{FramePosition}_{\text{FrameRate}} * \text{PositionStepsPerSecond}}{\text{FrameRate}}\right). \tag{5.2}$$

PositionStepsPerSecond denotes the number of position steps within one second (e.g. 1,000,000 for one step per microsecond) and *FrameRate* the corresponding remapping interval list's frame rate. For example, the position 15,000,000 is frame position 450 for 30 frames/s, 300 for 20 frames/s and 105 for 15 frames/s.

Now, a resource/utilization list has to be calculated for every position having an interval start in one of the frame rates' remapping interval lists. For example, the left side of figure 5.1 shows a stream having a frame rate of 25 frames/s. A frame rate scaled version to 12 frames/s can be found on the right side. The interval borders are marked by horizontal lines. The 25 frames/s version contains interval borders at position 0.02 s, 0.22s, 0.42s, 0,62s, 0.82s and 1.02s. For these positions, a resource/utilization list has to be calculated and stored. But since the second version contains interval borders at 0.02s, 0.42s and 0.90s and resource/utilization lists for the first two positions have already been stored, only an additional storage for position 0.90s will be necessary.

Possible utility functions are described in 2.5.1. Since QoS dimensions like frame rate and frame size are finite and have got minimum and maximum values, the following simplifications can be applied to the utilization calculation: First, instead of always recalculating the utility function's constants to fit the varying QoS dimension's range (e.g. 30 to 100 KBytes frame size for the first interval and 50 to 150 for the next one), a scale factor $\vartheta \in [0,1] \subset \mathbb{R}$ can be used. $\vartheta = 0$ should correspond to the dimension's minimum value and respectively $\vartheta = 1$ to its maximum one. Therefore:

$$\vartheta := \frac{\text{RealValue} - \text{MinValue}}{\text{MaxValue} - \text{MinValue}}. \tag{5.3}$$

For example for MinValue=100 and MaxValue=1000, a value of 865 corresponds to $\vartheta = 0.85 = 85\%$.

Further, the utilizations can be normalized: Its value should also be out of $[0,1] \subset \mathbb{R}$ - if this is not already ensured by the utility function itself. Similarly, this can also be applied to the utility function compositions (application utility, see section 2.5.2 and 2.5.3) and finally the system utility (see section 2.5.3). The reason for doing this is to simplify comparision of utilizations between different utility functions or their compositions.

Since the ASRMD1 algorithm described in section 2.5.4 is a solution to the SRMD QoS optimization problem (only a single resource), an extension to support layered transmission (several bandwidths for different layers) is required: The resource to be divided up is the total bandwidth, that is the sum of all DiffServ classes' bandwidths. An explaination of this media-specific total bandwidth to layers' bandwidths mapping will be given in section 5.2. As described in section 4.2, the used QoS dimensions are frame rate and frame size. In this context, frame size denotes the *reserved frame size* which is associated with the reserved bandwidth as follows:

$$\text{FrameSize} = \left\lfloor \frac{\text{Bandwidth}}{\text{FrameRate}} \right\rfloor, \tag{5.4}$$

$$\text{Bandwidth} = \lceil \text{FrameRate} * \text{FrameSize} \rceil. \tag{5.5}$$

This reserved frame size can be viewed as the maximum output of the traffic shaper (see section 2.2.4) during one frame time. Since each layer has got its own frame size, the quality space for stream i (see section 2.5.3) is defined as follows:

$$Q_i := \text{FrameRates}_i \times \text{FrameSizes}_{i1} \times \text{FrameSizes}_{i2} \times \ldots \times \text{FrameSizes}_{il_i},$$

where FrameRates_i denotes the set of possible frame rates and FrameSizes_{ij} the set of possible frame

Resource/Utilization Point
Utilization
Total Bandwidth (Resource)
Frame Rate
Bandwidth of Layer #0
Bandwidth of Layer #1
...
Bandwidth of Layer #n

Table 5.1: A resource/utilization point

sizes for layer $j \in \{1, ..., l_i\}$. Therefore, the utility function is:

$$U_i(\overrightarrow{\vartheta}) := f_i(u_{\text{FrameRate},i}(\vartheta_0),\ u_{\text{FrameSize}_1,i}(\vartheta_1),\ ...,\ u_{\text{FrameSize}_{l_i},i}(\vartheta_{l_i})) \in [0,1] \subset \mathbb{R}, \tag{5.6}$$

where f_i denotes the function used to compose (see section 2.5.2 for examples) the dimensions' corresponding utility functions $u_{\text{FrameRate},i}$ and $u_{FrameSize_j,i}$ (see section 2.5.1 for examples) and $\overrightarrow{\vartheta} \in [0,1]^{l_i+1} \subset \mathbb{R}^{l+1}$ a vector of scale factors for each dimension.

5.2 Resource/Utilization Points

As it is shown in section 2.5.4, the ASRMD1 algorithm calculates a resource (= bandwidth) allocation $resAllocated_i$ for each stream i. Finally, this allocation can be mapped to a quality setting $q \in Q_i$ of the media by choosing **one** out of a set given by the function h_i: $q \in h_i(resAllocated_i)$. To simplify the later mapping, a resource/utilization point will be a priori associated with one 'useful' choice. Therefore, fields for frame rate and each layer's bandwidth[1] for this choice have to be added. The result is shown in table 5.1. Here, *Total Bandwidth* denotes the resource of this point and is simply the sum of all layer bandwidths. From now on, the notion *resource/utilization point* refers to this extended definition. It is important to denote here, that it is only necessary to store the bandwidth for a buffer delay of one frame (see section 2.3.1 and 2.2.4). The translation to other buffer delays will be explained in section 5.4.

Now, such a resource/utilization point has to be calculated for a given upper bandwidth limit. Formally, this is a function

$$\psi : \underbrace{\mathbb{N}}_{\text{Upper bw. limit}} \rightarrow \underbrace{[0,1] \times \text{TotalBandwidths} \times \text{FrameRates} \times \text{Bandwidths}_1 \times ... \times \text{Bandwidths}_n}_{\text{Resource/Utilization point}}.$$

This calculation can be splitted into a generic media-independent and a media-dependent part. The media-independent part is shown in algorithm 3. For every frame rate, the upper bandwidth limit is divided up to the media's layers using the media-dependent algorithm (line 6 to 8, example follows below). Especially, this media-dependent part also contains the choice of a 'useful' quality setting $q \in h_i(resAllocated_i)$. In line 10 to 11, the total bandwidth and frame sizes for each layer of this setting q are used to calculate the setting's utilization using formula 5.6. Finally, the settings resulting in the highest utilization are returned as resource/utilization point of the given upper bandwidth limit (line 12 to 18).

Having a first look at the algorithm, it seems obvious to stop if at frame rate n it is not even possible to allocate the minimum bandwidth requirements to the layers. But this is wrong due to the independent remapping intervals for every frame rate, as the following example shows: Figure 5.2 shows a stream having a frame rate of 25 frames/s (left side) and a frame rate scaled version to

[1]Note that reserved bandwidth is directly associated with reserved frame size by the formulas 5.4 and 5.5.

Algorithm 3 Calculation of a given bandwidth's resource/utilization point

```
01 calculateMaximumUtilizedPoint(upperBandwidthLimit) {
02    ResourceUtilizationPoint = <empty>;
03    for(frameRate = getMinFrameRate();
04       frameRate ≤ getMaxFrameRate();
05       frameRate = getNextFrameRate(frameRate)) {
06       // Divide up bandwidth to layers (media-dependent part):
07       bandwidthToFrameSizes(upperBandwidthLimit,frameRate);
08          => frameSize_i, i ∈ {1, ..., layers}
09       totalBandwidth = Σ_{i=1}^{layers} ⌈frameSize_i * frameRate⌉;
10       utilization    = calculateUtilizationForSetting(
11                           frameRate,frameSize_1, ..., frameSize_layers);
12       if(ResourceUtilizationPoint.Utilization < utilization) {
13          ResourceUtilizationPoint = {
14             utilization, totalBandwidth,
15             frameRate,
16             ⌈frameSize_1 * frameRate⌉, ..., ⌈frameSize_n * frameRate⌉
17          };
18       }
19    return(ResourceUtilizationPoint);
20 }
```

18 frames/s (right side). For this media type, frame size scalability may not be allowed. The frame marked by an arrow is within the first interval for 25 frames/s and the second interval for 18 frames/s. Now, starting to send at position 0.2s, the first frame to send will be the marked one in both cases. Assuming a buffer delay of one frame, it is necessary to reserve 4536 bytes/frame = 113400 bytes/s in the first case and 8912 bytes/frame = 160416 bytes/s in the second one. Further assuming that the available bandwidth is only 128 KBytes/s, 18 frames/s will be impossible. Therefore, stopping trials at 18 frames/s would result in a low frame rate (less than 18) and quality, although high quality at 25 frames/s would be achievable at 128 KBytes/s.

As mentioned above, the algorithm has got a media-dependent part. An example for MPEG-1/2 is shown in algorithm 4. For each MPEG layer (base layer, enhancement layer(s)), the algorithm tries to map bandwidth to each transport layer's frame size: First, the minimum allocation is tried. If this is successful, the maximum one will be tried. In case of a failure here, the remaining bandwidth will be mapped to the transport layers according to the ratio of I-, P- and B-frames of the corresponding

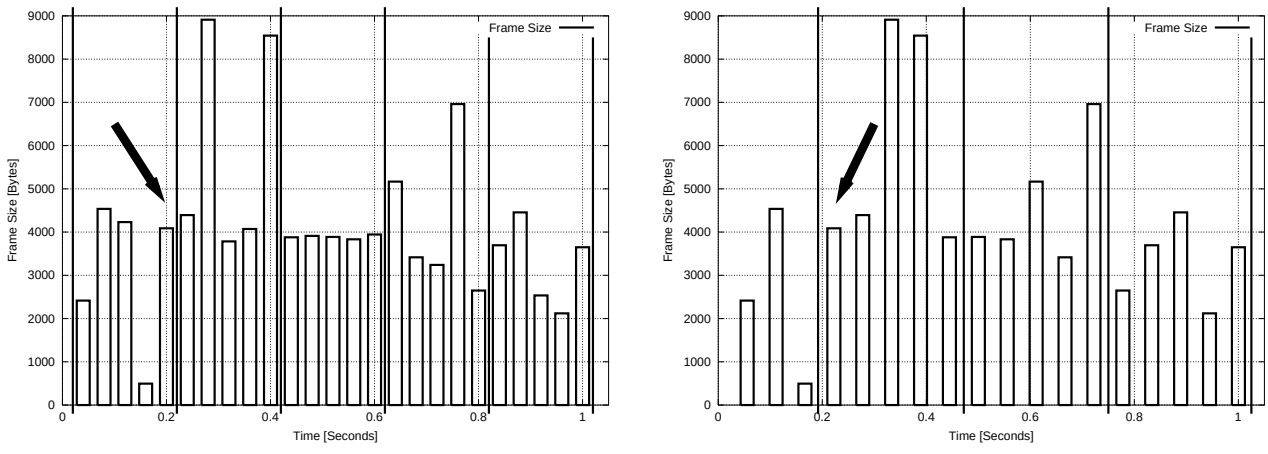


Figure 5.2: Example for lower frame rate but higher bandwidth requirement

Algorithm 4 The media-dependent resource/utilization list calculation part for MPEG

```
01 bool bandwidthToFrameSizes(bandwidth,frameRate) {
02    allSuccessful = true;
03    Set all layers' frame sizes to 0.
04    for(i = 0;i < MPEGLayers;i++) {
05       Try to get minimum frame size allocation for each layer.
06       if(success) {
07          Try maximum frame size allocation for each layer.
08          if(!success) {
09             Allocate remaining bandwidth to each layer
10                using ratio I:P:B.
11          }
12       }
13       else if(i == 0) {
14          allSuccessful = false;
15       }
16    }
17    return(allSuccessful);
18 }
```

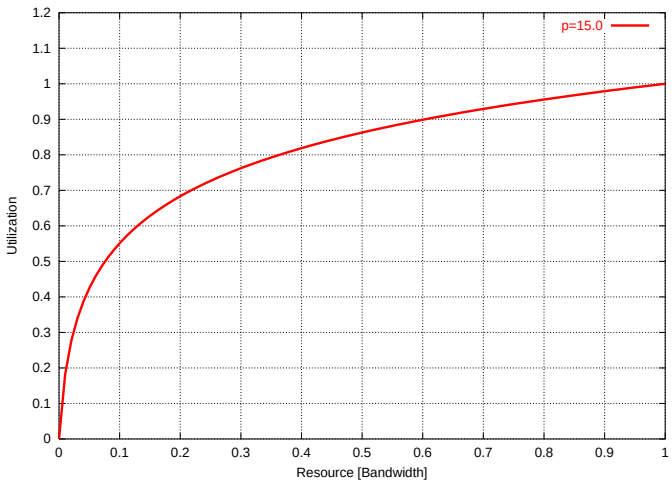


Figure 5.3: An example utility function for the resource/utilization list calculation

MPEG layer (see section 4.2 for details about the MPEG layering). Note, that the same utility function is assumed for all transport layers of the same MPEG-layer.

This may also be applied to H.263 by adding support for PB-frames. For MP3, the algorithm is trivial since there is only one layer (see section 4.2): It is therefore simply the application of formula 5.4.

5.3 Resource/Utilization Lists

Now, sorted lists of resource/utilization points - the so called resource/utilization lists - have to be calculated for the bandwidth range given by a stream's minimum and maximum bandwidth requirement. A trivial implementation would be to pick n bandwidth settings of equal difference out of the range and calculate the resource/utilization points for them:

$$\text{step}_i = \text{MinBandwidth} + \frac{i-1}{n-1} * (\text{MaxBandwidth} - \text{MinBandwidth}) \quad \forall i \in \{1, ..., n\}.$$

Resource	Utilization
0 %	0 %
10 %	55 %
20 %	68 %
30 %	76 %
40 %	82 %
50 %	86 %
60 %	90 %
70 %	93 %
80 %	95 %
90 %	98 %
100 %	100 %

Table 5.2: Using the trival calculation

Resource	Utilization
0 %	0 %
1.5625 %	24 %
3.125 %	34 %
6.25 %	46 %
9.125 %	53 %
12.5 %	60 %
25 %	72 %
37.5 %	80 %
50 %	86 %
75 %	94 %
100 %	100 %

Table 5.3: Using the recursive algorithm

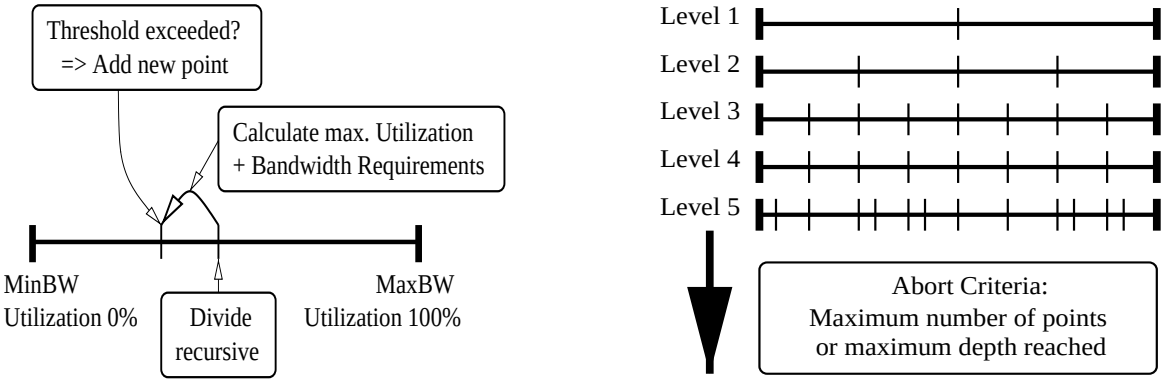


Figure 5.4: The recursive resource/utilization list calculation algorithm

An example utility function can be found in figure 5.3 (using formula 2.5, $p = 15.0$ from section 2.5.1), the trival algorithm's result for $n = 11$ is shown in table 5.2. The stream reaches high quality at a quite low bandwidth (resource) requirement, e.g. 76% at 30% bandwidth or 82% at 40% bandwidth. Unfortunately, the utilization distribution results in only a few points at lower and a lot of points at higher utilizations (e.g. 8 of 11 points at utilizations greater than 75%). A more sophisticated algorithm would therefore be desirable.

A graphical view of an advanced algorithm can be found in figure 5.4, its pseudocode is shown in algorithm 5. It recursively divides the bandwidth range and calculates the resource/utilization point. As mentioned above, the point's resource is then the sum of all layers' bandwidths (e.g. 1000 KBytes/s result in a usage of 800 KBytes/s only, since the next possible step would be 1100 KBytes/s). If a point's resource and utilization difference to its left and right neighbor point is greater than or equal a given *bandwidth threshold* or *utilization threshold*, then the new point is added:

$$\begin{aligned}
(&|p_{\text{Left}}.\text{Resource} - p.\text{Resource}| \geq \text{BandwidthThreshold} \quad \wedge \\
&|p_{\text{Right}}.\text{Resource} - p.\text{Resource}| \geq \text{BandwidthThreshold}) \quad \vee \\
(&|p_{\text{Left}}.\text{Utilization} - p.\text{Utilization}| \geq \text{UtilizationThreshold} \quad \wedge \\
&|p_{\text{Right}}.\text{Utilization} - p.\text{Utilization}| \geq \text{UtilizationThreshold}) \quad \Rightarrow \text{Add point.}
\end{aligned}$$

An exception are the points at minimum and maximum bandwidth (0% and 100% utilization): These points are always added, therefore a resource/utilization list contains at least these two elements.

Algorithm 5 Recursive resource/utilization list calculation

```
01 const bandwidthThreshold;
02 const utilizationThreshold;
03
04 void recursionStep(minBandwidth,maxBandwidth,list[],level,maxLevel) {
05    newBandwidth = (minBandwidth + maxBandwidth) / 2;
06    if(level == maxLevel) {
07       Calculate maximum utilization RU point
08          => utilization, realBandwidth.
09       Add point to list, if utilization and bandwidth
10          difference between next lower and next higher
11          point in list is greater than utilization and
12          bandwidth thresholds.
13       Always add point for minimum and maximum bandwidth.
14    }
15    else {
16       recursionStep(minBandwidth, newBandwidth, list,
17                         level + 1, maxLevel);
18       recursionStep(newBandwidth, maxBandwidth, list,
19                         level + 1, maxLevel);
20    }
21 }
22
23 void calculateList(minBandwidth,maxBandwidth,list[],maxLevel) {
24    for(i = 0;i ≤ maxLevel;i++) {
25       recursionStep(minBandwidth,maxBandwidth,list,0,i);
26    }
27 }
```

The abort criteria of the algorithm are a reached maximum number of calculated points (e.g. 32) or a maximum reached recursion depth. Since there may be recursion levels having points not satisfying the formula above, the minimum depth for at least the given number of points has to be increased:

$$\text{Depth} = \lceil \log_2(n) \rceil + \text{Additional Depth}. \tag{5.7}$$

Practical experience shows, that an additional depth of 2 or 3 is a good compromise between runtime and output quality. It is important to denote here that the algorithm first completes calculation of all possible points of a given recursion depth (see right side of figure 5.4) before increasing it!

An example using the utility function from figure 5.3, a utilization threshold of 5% and a resource threshold of 0% can be found in table 5.3. As it is shown, the utilization values are quite regularly distributed over the range from 0% to 100%. See also table 5.2 for comparision to the trivial calculation. Table 5.4 shows the recursion levels of this example. A recursion depth of 4 is necessary for the limit of 11 points, two levels are used additionally. Note that without these two levels, the points for 1.5625%, 3.125% and 9.125% bandwidth (24%, 34% and 53% utilization) would not be included!

The runtime of the recursive algorithm is $O(n)$ and therefore equal to the trivial one. Assuming the resource/utilization lists to be calculated online, it is necessary to have a look at the constant factors. The trival algorithm only requires $n * |\text{FrameRates}|$ calls of the media-dependent bandwidth to layer's frame sizes mapping. But for the recursive one, it is:

$$\text{Calls} \leq \underbrace{|\text{FrameRates}|}_{\text{Number of Frame Rates}} * \underbrace{2^{(\lceil \log_2(n) \rceil + \text{Additional Depth})}}_{\text{Resource/Utilization Point Calculations}},$$

Recursion Level	Bandwidth Steps	Added Points	Skipped Points
Initialization	0 %, 100 %	0 %, 100 %	-
1	50 %	50 %	-
2	25 %, 75 %	25 %, 75 %	-
3	12.5 %, 37.5 %, 62.5 %, 87.5 %	12.5 %, 37.5 %	62.5 %, 87.5 %
4	6.25 %, 18,75 %, . . .	6,25 %	18.75 %, . . .
5	3.125 %, 9.125 %, 15,375 % . . .	3.125 %, 9.125 %	15,375 %, . . .
6	1.5625 %, 4,6875, . . .	1.5625 %	4,6875, . . .

Table 5.4: Recursion levels for the resource/utilization list example

Point	**Utilization**	**Resource**	Layer #1	Layer #2	Layer #3	Frame Rate
#01	0 %	28,272	28,272	0	0	1
#02	8.3 %	70,949	70,949	0	0	2
#03	17.9 %	156,302	99,088	66,214	0	3
#04	29.4 %	241,656	139,283	102,373	0	4
#05	41.9 %	327,010	188,478	138,852	0	6
#06	51.3 %	412,364	237,673	174,691	0	8
#07	55.4 %	497,718	262,555	192,978	42,185	9
#08	59.5 %	583,072	310,247	228,032	44,793	9
#09	65.6 %	711,103	381,785	280,613	48,075	9
#10	73.9 %	881,811	477,169	350,721	53,923	9
#11	77.9 %	1,052,518	522,104	394,980	135,434	10
#12	83.0 %	1,214,224	604,296	457,160	152,768	11
#13	86.3 %	1,393,934	692,363	523,784	177,787	13
#14	90.4 %	1,564,642	805,231	508,651	250,760	24
#15	94.8 %	1,735,351	901,515	569,472	264,364	24
#16	100 %	2,759,600	1,373,400	1,039,000	347,200	25

Table 5.5: An example resource/utilization list of an MPEG-1 video

that is $2^{\text{Additional Depth}}$ times more (e.g. usually 4 or 8 for an additional depth of 2 or 3). Assuming further a scenario of many streams and therefore e.g. 50 reached remapping intervals per second, 64 resource/utilization points per list and an additional depth of 3, this would result in $50 * 30 * 2^{\log_2(64)+3} = 768,000$ calls compared to $96,000$ in the trivial case. Due to this high CPU requirement, it is useful to calculate the resource/utilization lists like the remapping intervals a priori.

An example for a complete resource/utilization list from an MPEG-1 video having 16 points is shown in table 5.5. The bandwidths are given in bytes per second. As shown in section 4.2, layer #1 contains I-frames, layer #2 P-frames and layer #3 B-frames.

5.4 Buffer Delay Translation of Bandwidths

As mentioned in section 5.2, only bandwidths for a buffer delay of one frame (see section 2.3.1) are stored in the resource/utilization points. Since the minimum and maximum bandwidths for all delays are known from the remapping interval's traffic description (see 2.3), it is possible to translate the points' fields:

For given bandwidth$_m$ for delay m (e.g. $m = 1$), traffic description of the corresponding frame rate's interval and delay n to convert the given bandwidth to, the translation is computed as follows:

1. Calculate the frame size scale factor α (see section 4.4.2):
$$\alpha = \frac{\text{Bandwidth}_m - \text{MinBandwidth}_\text{m}}{\text{MaxBandwidth}_\text{m} - \text{MinBandwidth}_\text{m}}.$$
2. Calculate the new bandwidth:
$$\text{Bandwidth}_n = \text{MinBandwidth}_\text{n} + \lceil \alpha * (\text{MaxBandwidth}_\text{n} - \text{MinBandwidth}_\text{n}) \rceil .$$

For example, $\text{MinBandwidth}_1 = 60$, $\text{MaxBandwidth}_1 = 120$ and $\text{Bandwidth}_1 = 100$ have to be translated to a delay of 6 frames. Then,
$$\alpha = \frac{100 - 60}{120 - 60} = \frac{2}{3}.$$
Now, $\text{MinBandwidth}_6 = 30$, $\text{MaxBandwidth}_6 = 60$. Therefore,
$$\text{Bandwidth}_6 = 30 + \left\lceil \frac{2}{3} * (60 - 30) \right\rceil = 50.$$
Due to its simplicity, such buffer delay translations can be computed online. Therefore, its not necessary to store bandwidths of other buffer delays than one. This results in a lower storage space requirement for the lists.

5.5 Summary

Since an ASRMD1-based algorithm (see section 2.5.4) requires resource/utilization lists for each stream, it had been necessary to calculate such lists for the remapping intervals and traffic descriptions of chapter 4. Therefore in this chapter, an efficient algorithm for the calculation of resource/utilization lists has been developed: It generates resource/utilization lists containing a limited number of resource/utilization points. Further, all points satisfy the constraint of having at least a certain configurable distance for utilization and/or bandwidth, e.g. all points' utilization has to differ by at least 3%. This results in a homogeneous distribution of the points over the whole utilization range from 0% to 100%.

To simplify the usage of utility functions, the scale factor ϑ has been introduced. That is, the utilization is not directly calculated from a resource setting but from its scale factor $\vartheta \in [0, 1] \subset \mathbb{R}$, where $\vartheta = 0$ corresponds to the resource's minimum and $\vartheta = 1$ to its maximum setting.

The chapter has closed with the demonstration, that it is only necessary to store a resource/utilization point's bandwidth settings for a buffer delay of 1 frame. Using the a priori calculated traffic description, this value can be translated for any other buffer delay, too. This results in lower storage space requirements for the calculated lists.

Chapter 6

The Bandwidth Management

The online bandwidth management for multimedia streams of different media types is developed in this chapter by extending the ASRMD1 algorithm described in section 2.5.4. This bandwidth management is based on each stream's a priori calculated remapping intervals and traffic descriptions (see chapter 4), resource/utilization lists (see chapter 5) and each layers' QoS requirements for maximum acceptable transfer delay, loss rate and jitter (see section 2.2).

6.1 Bandwidth Management Basics

As mentioned in the description of the CORAL concept in section 3.1, each stream belongs to a certain session. The properties of a session are shown in table 6.2. Minimum and maximum bandwidth of a session may be given[1]. A minimum bandwidth can be used to ensure a minimum quality. The bandwidth sum for all streams of the session may not exceed the upper limit (e.g. the user is connected via a low-bandwidth link at only 1 MBit/s). This will also be a limit for the cost (cost factor of the most expensive usable class multiplied by the maximum bandwidth, see explaination below). It is important to note that a cost limit would not be a bandwidth limit, since it may be possible that a cheaper class than expected may be usable. In this case, more data than expected may be sent, exceeding a link's speed limit.

The properties of a stream are shown in table 6.1. Each stream contains a priority, each layer's QoS requirements (maximum delay, acceptable loss rate and jitter, see section 2.2), the remapping intervals

[1]It is possible to give no minimum and/or maximum bandwidth limit(s). In this case, the minimum is simply zero and respectively the maximum is the sum of all streams' maximum requirement.

Property	Description
Priority	Priority of the stream
Layer1QoSRequirements	Maximum delay, loss rate and jitter of layer #1
...	...
Layer*n*QoSRequirements	Maximum delay, loss rate and jitter of layer #*n*
RemappingIntervalList	List of remapping intervals for each frame rate
Layer1TDescription	Traffic description of layer #1 for each interval
...	...
Layer*n*TDescription	Traffic description of layer #*n* for each interval
ResourceUtilizationLists	Set of resource/utilization lists
PossibleDSClassMappings	see section 6.1

Table 6.1: A stream description

Property	Description
Priority	Priority of the session
MinBandwidth	Minimum bandwidth
MaxBandwidth	Maximum bandwidth
Stream1	Description of stream #1
...	
Stream*m*	Description of stream #*m*
MultiPointList	see section 6.2

Table 6.2: A session description

for all frame rates (see chapter 4), each layer's traffic description for every interval and frame rate (see section 2.3.3) and finally the resource/utilization lists for every interval start (see chapter 5).

The SLA (see section 2.2.3) contains the available bandwidth and a *cost factor* of each class. The bandwidth pricing is done as follows: To simplify the cost calculation, only the reserved bandwidth is charged. That is the product *Bandwidth*CostFactor*. This will be sufficient since the reservation will never be exceeded due to the usage of the a priori calculated traffic descriptions (see chapter 4) and the traffic shaper (see section 2.2.4). Therefore, this cost factor can be viewed as the combination of holding charge and usage charge, described in section 3.1. See also section 6.5 for some comments on the charging scheme.

Now, the goal of the bandwidth management is to calculate a 'good' mapping of the classes' available bandwidth to the streams' layers. Within a session, the user usually requires a fair bandwidth distribution for his streams - depending on each stream's priority. For example, a video stream should have the same user satisfaction like its audio stream. But from a global view, it might be useful to provide as high user satisfaction as possible to as many streams as possible. To cope with this problem, resource/utilization points of each session's streams will first be combined to so called multipoints using e.g. a fair distribution (see section 6.3). Then, an algorithm based on the ASRMD1 algorithm described in section 2.5.4 will be applied to these multipoints to calculate e.g. an utilization-maximizing bandwidth mapping (see section 6.4). But first, some preparations are necessary.

6.2 Stream Description Initialization

Before doing the bandwidth mapping, it is first necessary to incorporate the packet headers into the bandwidths given by the traffic descriptions and resource/utilization lists (see section 6.2.1) and then calculate possible layer to DiffServ class mappings (see section 6.2.2).

6.2.1 Packet Headers and the Payload⇔Raw Translation

As shown in table 6.1, the *stream description* contains the traffic descriptions of each layer for each interval and frame rate plus the resource/utilization list for every interval start. But these bandwidth descriptions only refer to the so called *payload data*, that is the number of bytes without any packet headers. Since the media stream has to be packaged in order to be sent over a network, this necessary packet overhead has to be added to the traffic descriptions. It is important to denote here, that this is not possible a priori, since the header length and maximum packet size[2] are first known during transmission, e.g. using IPv4 (at least 20 bytes header, see section 2.1.2) or IPv6 (40 bytes header,

[2]The maximum packet size denotes the limit of the underlying network, e.g. Ethernet, ATM or FDDI. To avoid fragmentation of IP packets (see section 2.1.2), it is recommended not to exceed this size.

see section 2.1.2) over Ethernet (maximum packet size: 1500 bytes) or FDDI (maximum packet size: 4500 bytes).

Algorithm 6 Conversion from payload to raw bandwidth

```
01 PktMaxSize    = 1500;          // e.g.  Ethernet
02 PktHeaderSize = 40+12+8+16;    // IPv6+UDP+RTP+X
03
04 cardinal payloadToRaw(frameRate,payload,bufferDelay) {
05     maxPktPayload = PktMaxSize - PktHeaderSize;
06     payloadPkts   = ⌈payload / PktMaxSize⌉;
07     maxFrameCount = getMaxFrameCountForDelay(frameRate,bufferDelay);
08     maxFramePkts  = ⌈maxFrameCount / bufferDelay * frameRate⌉ - 1;
09     return(payload + ⌈(maxFramePkts + payloadPkts) * PktHeaderSize⌉);
10 }
```

Algorithm 6 computes the raw bandwidth from the payload bandwidth. This algorithm can be explained best using an example: A stream has got a frame rate of 60 frames/s and a bandwidth of 250 KBytes/s = 256,000 bytes/s for a buffer delay of 10 frames (=$\frac{1}{6}$ second). The stream should be transported via Ethernet (maximum packet size is 1500 bytes) using IPv6 (40 bytes header), UDP (8 bytes header), RTP (12 bytes header in this example) and a media-specific protocol (16 bytes header in this example). Therefore, the total header size 40+8+12+16=76. This results in a maximum packet payload of 1500-76=1424 bytes, calculated in line 5. The number of packets necessary to transport 256,000 bytes per second is $\lceil \frac{256{,}000}{1424} \rceil = 180$ (line 6).

A packet belongs to only one certain frame, that is the last packet of frame $n-1$ will not already contain the first bytes of frame n. Since the calculation above does not incorporate this behavior, it is therefore necessary to add additional packet headers for these "frame starts". The trivial approximation for the number of packets to add is of course $\lceil \text{FrameRate} \rceil - 1$. One packet may be subtracted, since the calculation of line 5 to 6 already includes the first packet. But in some cases, this is very inefficient as it will be shown below. A better approximation would be the maximum frame count per frame, converted to the maximum frame count per second. For example, during the buffer delay of 10 frames, a maximum of only 4 frames will be really sent. All other frames have got a size of 0 (see also the frame sizes of the MPEG layering example in table 4.1). In this case, only $\lceil \frac{4}{10} * 60 \rceil - 1 = 23$ (line 8) packet headers have to be added instead of $60 - 1 = 59$. Now, the raw bandwidth can be calculated using the given number of packets (line 9):

$$\underbrace{256,000}_{\text{Payload Bandwidth}} + \underbrace{(180+23)*76}_{\text{Header Overhead}} = \underbrace{271,428}_{\text{Raw Bandwidth}} .$$

In this example, the gain using the maximum frame count approximation is only 36 packets/s or 2736 bytes/s. Compared to the total bandwidth, this is a quite few amount. But for streams having small bandwidth and frame count but high frames rates and buffer delays, the yield is much higher: For example, using 100 frames/s, 150 frames buffer delay (=1.5 seconds), a maximum frame count of 2 frames for the delay of 150 frames and a bandwidth of 10 KBytes/s (=10,240 bytes/s), the number of additional packets is only $\lceil \frac{2}{150} * 100 \rceil - 1 = 1$, compared to 100-1=99 using the frame rate approximation. In bandwidths, this is 76 bytes/s versus 7524 bytes/s. This is a quite large amount, compared to $10,240 + 8 * 76 = 10,848$ for the payload transport.

Since such traffic as described in the example above is realistic (e.g. an audio transmission of 100 frames/s, sending e.g. additional text or picture information in an enhancement layer every 0.75 seconds), it is useful to apply the frame count approximation and therefore to include a so called *frame count empirical envelope* approximation to the a priori traffic descriptions. That is, instead of

counting the maximum number of bytes for a given delay as for the bandwidth empirical envelope approximation (see section 2.3.3), simply the maximum number of non-zero frame sizes is counted.

Algorithm 7 Conversion from raw to payload bandwidth

```
01 PktMaxSize    = 1500;          // e.g.  Ethernet
02 PktHeaderSize = 40+12+8+16;    // IPv6+UDP+RTP+X
03
04 cardinal rawToPayload(frameRate,raw,bufferDelay) {
05     maxFrameCount = getMaxFrameCountForDelay(frameRate,bufferDelay);
06     maxFramePkts  = ⌈ maxFrameCount / bufferDelay * frameRate ⌉ - 1;
07     payloadPkts   = ⌈ (raw-(maxFramePkts*PktHeaderSize)) / PktMaxSize ⌉;
08     return(raw - ⌈(maxFramePkts + payloadPkts) * PktHeaderSize⌉);
09 }
```

The reverse calculation from raw to payload frame size can be found in algorithm 7. Again, the settings of the 250 KBytes/s stream example above are used. First, the number of 'frame start' packets is calculated (line 4-5), therefore again 23. Then, the number of packets to transport the payload is calculated from the raw bandwidth minus the additional 'frame start' headers, again 180. Finally, the total number of packets is known and subtracting the packet headers from the raw bandwidth results in the payload bandwidth (line 8):

$$\underbrace{271,428}_{\text{Raw Bandwidth}} - \underbrace{(23+180)*76}_{\text{Header Overhead}} = \underbrace{256,000}_{\text{Payload Bandwidth}} .$$

6.2.2 Layer to DiffServ Class Mappings

Now, it is possible to check, which of the SLA's classes are sufficient for the layers' QoS requirements: Bandwidth, maximum transfer delay, acceptable loss rate and jitter. The current transfer delay of each class is measured using ICMP echo requests and replies as described in section 3.2, the current loss rate and jitter are calculated from RTCP reports (see section 2.1.5).

The bandwidth is given by the traffic description, jitter and loss rate limits are properties of the application. The maximum transfer delay is a limit given by the user, e.g. 750ms for video on demand (VoD) or 150ms for a video conference. If a class's transfer delay is lower, e.g. 100ms, much bandwidth can be saved using buffering (see section 2.3.1). Therefore, a buffer delay of up to 650ms (750ms-100ms) may be used for the VoD example and still up to 50ms (150ms-100ms) for the video conference. Since the transfer delay is usually slightly varying, a small tolerance would be useful. This variability depends on the properties of the class: EF has got a very low variability due to small queues. On the other side, the variability of BE may be very high due to congestion. It is therefore recommended to use a variability given as a fraction ($\in [0,1] \subset \mathbb{R}$) of the class's measured bandwidth. Then, it also incorporates the fact that more hops (= higher delay) may cause more variabiliy. Finally, a system tolerance may be necessary. Due to inaccurate process scheduling, data may be buffered too early or too late (see section 7.3 for more details). Therefore, the maximum possible buffer delay for class n is:

$$\text{BufferDelay}_n := \text{MaxTransferDelay} - \tag{6.1}$$

$$\underbrace{(1.0 + \text{DelayTolerance}_n) * \text{MeasuredDelay}_n}_{\text{Measured delay of class } n \text{ plus tolerance}} - \underbrace{\text{SystemTolerance}}_{\text{Scheduling tolerance}} . \tag{6.2}$$

An example traffic description for a stream consisting of two layers is given in table 6.3 and table 6.4. The points of the empirical envelope $E(t)$, the traffic constraint function $b(t)$ and the resulting

Delay t [Frames]	$E(t)$ [KBytes]	$b(t)$ [KBytes/Frame]	Bandwidth [KBytes/s]
1	150	150	1500
2	200	100	1000
3	231	77	770
4	240	60	600
5	245	49	490

Table 6.3: Example traffic constraint for layer #1

Delay t [Frames]	$E(t)$ [KBytes]	$b(t)$ [KBytes/Frame]	Bandwidth [KBytes/s]
1	60	60	600
2	100	50	500
3	144	48	480
4	188	47	470
5	230	46	460

Table 6.4: Example traffic constraint for layer #2

bandwidth per second for a frame rate of 10 frames/s are given for a buffer delay of 1 to 5 frames. See also figure 6.1 for the graphical representation of the empirical envelope $E(t)$ (left side) and the bandwidth $b(t)$ to be reserved (right side). As it is shown, the first layer's buffering gain is much higher compared to the second one: For a buffer delay of 5 frames (=$\frac{1}{2}$ second), the required bandwidth reduces to about 33% of the original value for the first layer but only to about 77% for the second one.

Now, the user's delay requirement may be 320ms. Transfer delays for each DiffServ class to the destination host are shown in table 6.5. A system delay tolerance of 20ms is used. The highest possible buffer delay is also shown in the table. In this case, the best effort service is unusable, since its delay of 500ms is far too high for the user's delay limit of 320ms. But the other classes are possibilities for the transport. For simplicity, this example does not contain a maximum acceptable loss rate and jitter. These QoS requirements are handled like the transfer delay.

Using the maximum achievable buffer delay of each class, it is now possible to calculate the really required bandwidth for each class using the buffer delay translation described in section 5.4. For the example above, the bandwidth of 1500 KBytes/s in layer #1 (see table 6.3) and 600 KBytes/s in layer #2 (see table 6.4) for a buffer delay of one frame shrinks to the values given by table 6.6 for layer #1

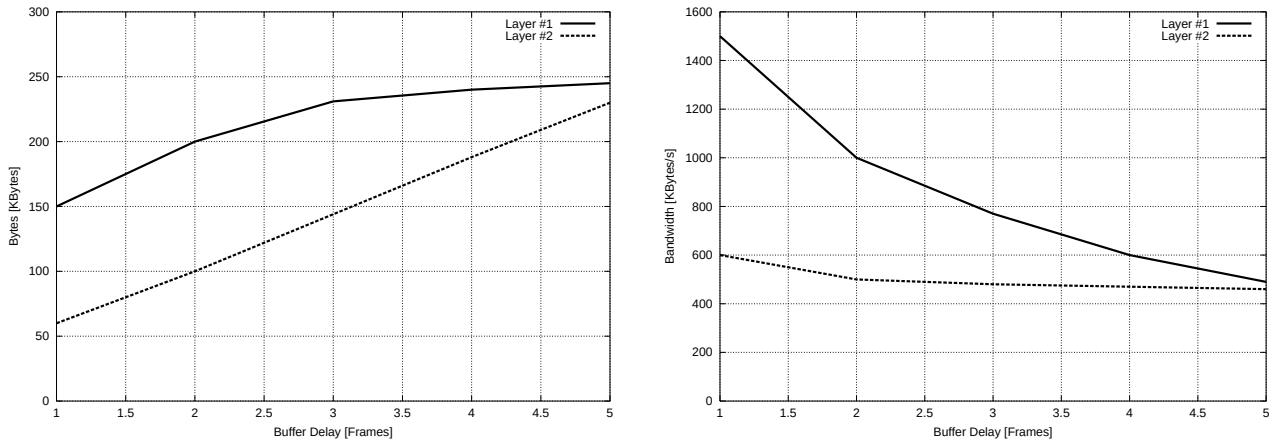


Figure 6.1: Empirical envelopes and bandwidth to reserve for layer #1 and #2

Class	Cost factor [$]	Variability	Measured class delay	Possible buffer delay
EF	4.0	5 %	100ms	200ms ≅ 5 frames
AF31	3.0	10 %	140ms	160ms ≅ 4 frames
AF21	2.5	10 %	180ms	120ms ≅ 3 frames
AF11	2.0	10 %	280ms	0ms ≅ 1 frame
BE	1.0	50 %	500ms	-

Table 6.5: DiffServ class delays for the layer to class mapping example

Class	Cost factor [$]	Delay [Frames]	Bandwidth [KBytes/s]	Cost [Bw*$]
EF	4.0	4	490	**1960**
AF31	3.0	3	770	2310
AF21	2.5	2	1000	2500
AF11	2.0	1	1500	3000

Table 6.6: Cost for layer #1 for an original bandwidth of 1500 KBytes/s

Class	Cost factor [$]	Delay [Frames]	Bandwidth [KBytes/s]	Cost [Bw*$]
EF	4.0	4	460	1840
AF31	3.0	3	480	1440
AF21	2.5	2	500	1250
AF11	2.0	1	600	**1200**

Table 6.7: Cost for layer #2 for an original bandwidth of 600 KBytes/s

and table 6.7 for layer #2. These tables also contain the total cost, that is the bandwidth multiplied by the corresponding DiffServ class's cost factor. As it is shown, the first layer's cheapest class is EF - although EF has got the highest cost factor! This is a result of the layer's high buffering gain. On the other side, the second layer's cheapest class is AF11. Since this layer's buffering gain is quite low, a faster but more expensive class would only increase the cost.

Using the described calculation, the resource/utilization list can now be extended by a list of possible layer to DiffServ class mappings for each point. Each of these mapping lists consists of class, bandwidth and buffer delay for each possible setting and is sorted ascending by cost. Therefore, the highest-quality resource/utilization point of the example above is:

$$\left(\begin{array}{c} 1090\,\mathrm{KB/s} \\ 100\,\% \\ 10\,\mathrm{Frames/s} \end{array} \left[\begin{array}{lcccc} L_1: & \left\langle \begin{array}{c} \mathrm{EF} \\ 490\,\mathrm{KB/s} \\ 5\,\mathrm{Frames} \end{array} \right. , & \begin{array}{c} \mathrm{AF31} \\ 770\,\mathrm{KB/s} \\ 4\,\mathrm{Frames} \end{array} , & \begin{array}{c} \mathrm{AF21} \\ 1000\,\mathrm{KB/s} \\ 3\,\mathrm{Frames} \end{array} , & \begin{array}{c} \mathrm{AF11} \\ 1500\,\mathrm{KB/s} \\ 1\,\mathrm{Frame} \end{array} \left. \right\rangle \\ L_2: & \left\langle \begin{array}{c} \mathrm{AF11} \\ 600\,\mathrm{KB/s} \\ 1\,\mathrm{Frame} \end{array} \right. , & \begin{array}{c} \mathrm{AF21} \\ 500\,\mathrm{KB/s} \\ 3\,\mathrm{Frames} \end{array} , & \begin{array}{c} \mathrm{AF31} \\ 480\,\mathrm{KB/s} \\ 4\,\mathrm{Frames} \end{array} , & \begin{array}{c} \mathrm{EF} \\ 460\,\mathrm{KB/s} \\ 5\,\mathrm{Frames} \end{array} \left. \right\rangle \end{array} \right] \right) .$$

Note, that the point's resource setting is the bandwidth sum of the best (=first) mapping possibilities of each layer ($490\,\mathrm{KBytes/s} + 600\,\mathrm{KBytes/s} = 1090\,\mathrm{KBytes/s}$ in the example above), since this is the cheapest transport possibility.

6.2.3 Buffering and the Resource/Utilization Lists

Now, it is necessary to remove some 'bad' points, that are points which have got a higher resource requirement and lower utilization than following points. For example, the a priori calculated resource/utilization list contains the following two points:

Point	Utilization	Resource	Layer #1	Frame rate
...	...	...	...	...
#*n*	55%	300 KBytes/s	300 KBytes/s	4
#*m*	90%	750 KBytes/s	750 KBytes/s	10
...	...	...	...	...

For a frame rate of 10 frames/s, the traffic description of the example in section 6.2.2 is used (see table 6.3 and figure 6.1). Then, point #*m* may be the frame size scaled version using $\vartheta = \frac{1}{4}$ for $\alpha = \frac{1}{3}$ (see section 4.4.2). The traffic description for a frame rate of 4 frames/s may be a constant size of 75 KBytes for each frame, which results in a bandwidth of 300 KBytes/s. This may be a result of frame rate scalability from the 10 frames/s version to the 4 frames/s version (see section 4.4.1), leaving only 75 KBytes peaks due to frame size scaling of $\vartheta = \frac{1}{4}$. Since each frame has got the same size here, buffering will result in no gain ($b(t) = \text{const}$).

Now, applying the buffer delay translation for a buffer delay of 5 frames as described in section 5.4, 750 KBytes/s shrink to only 245 KBytes/s for point #*m*. But for point #*n*, there are no changes! Therefore:

#*n*	55%	300 KBytes/s	300 KBytes/s	4
#*m*	90%	245 KBytes/s	245 KBytes/s	10

Since it does not make sense to allocate 300 KBytes for 55% utilization instead of only 245 KBytes/s for 90%, point #*n* has to be removed.

6.2.4 The Sorting Value

As described in section 2.5.4, the ASRMD1 algorithm would generate a fair sharing if the resource/utilization points are sorted by utilization and a utilization-maximizing distribution (unfair) for sorting by resource. To make this fairness behavior configurable and also incorporate a stream priority, it is useful to introduce a so called *sorting value*:

$$\text{SortingValue} := \text{PriorityFactor} * \underbrace{\left(\varphi(\text{Resource}) * (1 - \omega_{\text{Fairness}}) + \text{Utilization} * \omega_{\text{Fairness}}\right)}_{\text{Unprioritized Sorting Value}}. \quad (6.3)$$

In this formula, $\omega_{\text{Fairness}} \in [0, 1] \subset \mathbb{R}$ sets the fairness from none (=0, the sorting value only depends on resource) to maximum (=1, the sorting value only depends on utilization). Therefore, the result is more or less dominated by the value of resource or utilization. More details about this fairness setting can be found in section 6.5, since it is necessary to explain more details about the bandwidth mapping first.

Since *Resource* is given in bandwidth units and *Utilization* $\in [0, 1] \subset \mathbb{R}$, it is not possible to compare these two values (e.g. 5000 KBytes/s resource and 0.95=95% utilization). Therefore, a function φ is necessary to convert the resource setting to a better comparable value. For example, simply:

$$\varphi(r) := \frac{r}{\text{SLA's Total Bandwidth}}. \quad (6.4)$$

This represents the bandwidth fraction of the resource/utilization point. It is important to denote here, that $\varphi(r) \in]0, 1] \subset \mathbb{R}$ for all allocatable resource/utilization points, since $\varphi(r) > 1$ implies a resource requirement r higher than the SLA's total bandwidth.

To incorporate a stream priority, the *unprioritized sorting value* is multiplied by a so called *priority factor* $\in]0, 1] \subset \mathbb{R}$. The result is called *sorting value* and can be viewed as a fraction of the unprioritized sorting value. So, for a factor of e.g. 20%, only 20% of the original value are 'counted' for the sorting. One possible formula for the priority factor is:

$\begin{pmatrix} 10000 \\ 50\% \end{pmatrix}$	$\omega_{\text{Fairness}} = 0.0$	$\omega_{\text{Fairness}} = 0.5$	$\omega_{\text{Fairness}} = 1.0$
Priority -100	0.17812500	0.44531250	0.71250000
Priority -50	0.13906250	0.34765625	0.55625000
Priority 0	0.10000000	0.25000000	0.40000000
Priority 50	0.06093750	0.15234375	0.24375000
Priority 100	0.02187500	0.05468750	0.08750000

Table 6.8: Sorting values for different priorities and fairness settings

$$\text{PriorityFactor} := \frac{1}{256}(256 - (\text{Priority} + 128)), \tag{6.5}$$

where *Priority* denotes a stream priority out of [-128,127].

Some examples can be found in table 6.8. It shows the sorting value of the resource/utilization point $\begin{pmatrix} 10000 \\ 50\% \end{pmatrix}$, calculated for different priorities and settings of ω_{Fairness}. Formula 6.4 with a total bandwidth of 50000 units has been used to do the resource conversion. Therefore, $\varphi(10000) = 0.2$.

Now, such a sorting value can be added to all points of the resource/utilization lists. Of course, ω_{Fairness} has to be constant for **all** points of **all** lists. It is important to denote here, that the sorting value will is not able to change a list's sorting by utilization and resource itself. The order of any two points of the same list will always be preserved. But comparing two streams of the same media, the higher prioritized stream will have lower sorting values for its points compared to the lower-prioritized one.

6.2.5 Parallelization

It is important to denote here, that the stream description initialization for a stream is independent of all other streams. Therefore, these computations can be done very efficiently using parallelization on multiprocessor systems. As measurements in section 8.5 will show, these stream description initializations consume most CPU time during bandwidth remapping. Therefore, parallelization will allow scenarios using hundreds or even thousands of streams, depending on number of processors and CPU power.

6.3 Session Description Initialization

In section 6.2, the initialization of a stream description has been shown. The next step is to have a look at the sessions: A session has got a session priority and contains one or more streams, each stream has got its own stream description (see section 6.2). To use an ASRMD1-based algorithm, it is now necessary to map the streams' resource/utilization points to so called *resource/utilization multipoints*, containing one resource/utilization point of each stream:

$$\left(\begin{array}{c} r_{\text{Total}} = \sum_{i=1}^{n} r_{id_i} \\ u_{\text{Total}} = \frac{\sum_{i=1}^{n} u_{id_i}}{n} \\ s_{\text{Multipoint}} \\ \left(\begin{array}{c} r_{1p_1} = \sum_{i=1}^{l_n} r_{1p_1}^{1i} \\ u_{1p_1} \\ s_{1p_1} \\ f_{1p_1} \end{array} \quad \left[\begin{array}{ll} L_1: & \left\langle \begin{array}{ccccc} C_{1p_1}^{11} & & C_{1p_1}^{21} & & C_{1p_1}^{c_1 1} \\ r_{1p_1}^{11} & , & r_{1p_1}^{21} & , \ldots, & r_{1p_1}^{c_1 1} \\ d_{1p_1}^{11} & & d_{1p_1}^{21} & & d_{1p_1}^{c_1 1} \end{array} \right\rangle \\ & \ldots \\ L_{l_1} & \left\langle \begin{array}{ccccc} C_{1p_1}^{11} & & C_{1p_1}^{21} & & C_{1p_1}^{c_1 1} \\ r_{1p_1}^{11} & , & r_{1p_1}^{21} & , \ldots, & r_{1p_1}^{c_1 1} \\ d_{1p_1}^{11} & & d_{1p_1}^{21} & & d_{1p_1}^{c_1 1} \end{array} \right\rangle \end{array}\right] \right) \\ \ldots \\ \left(\begin{array}{c} r_{np_n} = \sum_{i=1}^{l_n} r_{np_n}^{1i} \\ u_{np_1} \\ s_{np_1} \\ f_{np_1} \end{array} \quad \underbrace{\left[\begin{array}{ll} L_1: & \left\langle \begin{array}{ccccc} C_{np_n}^{11} & & C_{np_n}^{21} & & C_{np_n}^{c_n 1} \\ r_{np_n}^{11} & , & r_{np_n}^{21} & , \ldots, & r_{np_n}^{c_n 1} \\ d_{np_n}^{11} & & d_{np_n}^{21} & & d_{np_n}^{c_n 1} \end{array} \right\rangle \\ & \ldots \\ L_{l_n} & \left\langle \begin{array}{ccccc} C_{np_n}^{1l_n} & & C_{np_n}^{2l_n} & & C_{np_n}^{c_n l_n} \\ r_{np_n}^{1l_n} & , & r_{np_n}^{2l_n} & , \ldots, & r_{np_n}^{c_n l_n} \\ d_{np_n}^{1l_n} & & d_{np_n}^{2l_n} & & d_{np_n}^{c_n l_n} \end{array} \right\rangle \end{array}\right]}_{\text{Layer to DiffServ class mappings}} \right) \end{array}\right) \quad . \tag{6.6}$$

In this formula, $\left(\begin{array}{c} r_{ip_i} \\ u_{ip_i} \\ s_{ip_i} \\ f_{ip_i} \end{array} \quad \left[\begin{array}{c} \ldots \\ \ldots \\ \ldots \\ \ldots \\ \ldots \end{array}\right]\right)$ denotes the i-th stream's p_i-th resource/utilization point, having resource r_{ip_i}, utilization u_{ip_i}, sorting value s_{ip_i} and frame rate f_{ip_i}. In the brackets, the possible layer to DiffServ class mappings are given as described in section 6.2.2: $r_{ip_i}^{jk}$ denotes the bandwidth required using DiffServ class $C_{ip_i}^{jk}$ with a buffer delay of $d_{ip_i}^{jk}$ for layer *#k*. Note, that these entries are sorted ascending by the resulting cost! Finally, r_{Total} is the total resource which is simply the sum of all points' first (= best) mapping's resource and u_{Total} is the total utilization which is the average of all points' utilization. The multipoint's sorting value $s_{\text{Multipoint}}$ will be described later.

Now, it is neccessary to calculate a list of multipoints for the session's streams. An algorithm for this problem is shown in algorithm 8. It can be explained best by an example: A session contains two streams S_1 and S_2, which both have got priority -128 for simplicity. The priority factor is therefore $\frac{256}{256} = 1$. Let $\omega_{\text{Fairness}} = 1$ (maximum fairness). The streams have got the following resource/utilization lists:

$$S_1: \left\langle \left(\begin{array}{c} 100\,\text{KB/s} \\ 0\,\% \\ 0.0 \end{array}\right), \left(\begin{array}{c} 200\,\text{KB/s} \\ 50\,\% \\ 0.5 \end{array}\right), \left(\begin{array}{c} 250\,\text{KB/s} \\ 70\,\% \\ 0.7 \end{array}\right), \left(\begin{array}{c} 400\,\text{KB/s} \\ 100\,\% \\ 1.0 \end{array}\right) \right\rangle ,$$

$$S_2: \left\langle \left(\begin{array}{c} 50\,\text{KB/s} \\ 0\,\% \\ 0.0 \end{array}\right), \left(\begin{array}{c} 150\,\text{KB/s} \\ 60\,\% \\ 0.6 \end{array}\right), \left(\begin{array}{c} 175\,\text{KB/s} \\ 70\,\% \\ 0.7 \end{array}\right), \left(\begin{array}{c} 300\,\text{KB/s} \\ 100\,\% \\ 1.0 \end{array}\right) \right\rangle .$$

First, a set of all possible sorting values is calculated and sorted ascending (line 2 to 9): *SortingValueSet*={0.0, 0.5, 0.6, 0.7, 1.0}. Next, for every value *value* out of this set, a resource/utilization point having highest possible sorting value less or equal *value* is searched from every stream's list. Then, the found points of every stream are joined to a multipoint (line 10 to 14). For example, for the

Algorithm 8 Calculation of session's resource/utilization multipoints.

```
01 calculateSessionMultiPoints(session) {
02    MultiPointList         = <emtpy>;
03    SessionSortingValueSet = <empty>; // no duplicates allowed!
04    for each(stream in session.StreamSet) {
05       for each(point in stream.ResourceUtilizationList) {
06          Add point.SortingValue to SessionSortingValueSet.
07          // This includes duplicate elimination!
08       }
09    }
10    Sort SessionSortingValueSet ascending.
11    for each(value in SessionSortingValueSet) {
12       for each(stream in session.StreamSet) {
13          Find point having highest sorting value less or equal value.
14       }
15       Join points found to multiPoint.
16       if(new multipoint and previous computed one differ) {
17          Calculate multipoint's global sorting value.
18          Append multiPoint to MultiPointList.
19       }
20    }
21    return(MultiPointList);
22 }
```

sorting value 0.6, the points found are $\begin{pmatrix} 200\,\text{KBytes/s} \\ 50\,\% \\ 0.5 \end{pmatrix}$ from stream S_1 and $\begin{pmatrix} 150\,\text{KBytes/s} \\ 60\,\% \\ 0.6 \end{pmatrix}$ from stream S_2. If the last added multipoint and the newly created one differ, the new multipoint's sorting value is calculated and the new multipoint is appended to the multipoint list. The multipoint's sorting value is the same like the value for points, except that the new constant $\omega_{\text{SessionFairness}}$ is used instead of ω_{Fairness}. This will be the global distribution fairness. It is calculated from the multipoint's resource and utilization setting as described in formula 6.6. See section 6.5 for more details about this fairness setting.

Therefore, the example's resulting multipoint list using $\omega_{\text{SessionFairness}} = 0$ for an utilization-maximizing global sharing and $\varphi(r) := \frac{r}{1000\,\mathbf{KBytes/s}}$ is shown in figure 6.2.

It is important to denote here, that the initialization of a session's multipoint list is independent of all other sessions. Therefore, its calculation can simply be parallelized, too.

6.4 The Bandwidth Mapping

Finally, the last step before the bandwidth remapping is to join the resource/utilization multipoint lists of each session and sorting the resulting global list by its multipoint sorting value. Again, this sorting does not change the order of each session's resource/utilization multipoints. Only the position compared to other sessions may change, depending on the session's priority. Now, the bandwidth remapping can be applied on this multipoint list. This is called *complete remapping* and will be described in the next subsection. An optimization which does a partial remapping only for a single stream, therefore called *partial remapping*, will be described in subsection 6.4.2.

$$\left\langle \begin{pmatrix} 150\,\mathrm{KB/s} \\ 0\,\% \\ 0.075 \\ \begin{pmatrix} 100\,\mathrm{KB/s} \\ 0\,\% \\ 0.0 \end{pmatrix} \\ \begin{pmatrix} 50\,\mathrm{KB/s} \\ 0\,\% \\ 0.0 \end{pmatrix} \end{pmatrix}, \begin{pmatrix} 250\,\mathrm{KB/s} \\ 25\,\% \\ 0.125 \\ \begin{pmatrix} 200\,\mathrm{KB/s} \\ 50\,\% \\ 0.5 \end{pmatrix} \\ \begin{pmatrix} 50\,\mathrm{KB/s} \\ 0\,\% \\ 0.0 \end{pmatrix} \end{pmatrix}, \right.$$
$$\left. \begin{pmatrix} 350\,\mathrm{KB/s} \\ 55\,\% \\ 0.175 \\ \begin{pmatrix} 200\,\mathrm{KB/s} \\ 50\,\% \\ 0.5 \end{pmatrix} \\ \begin{pmatrix} 150\,\mathrm{KB/s} \\ 60\,\% \\ 0.6 \end{pmatrix} \end{pmatrix}, \begin{pmatrix} 375\,\mathrm{KBytes/s} \\ 70\,\% \\ 0.1875 \\ \begin{pmatrix} 250\,\mathrm{KB/s} \\ 70\,\% \\ 0.7 \end{pmatrix} \\ \begin{pmatrix} 175\,\mathrm{KB/s} \\ 70\,\% \\ 0.7 \end{pmatrix}, \end{pmatrix}, \begin{pmatrix} 700\,\mathrm{KB/s} \\ 100\,\% \\ 0.35 \\ \begin{pmatrix} 400\,\mathrm{KB/s} \\ 100\,\% \\ 1.0 \end{pmatrix} \\ \begin{pmatrix} 300\,\mathrm{KB/s} \\ 100\,\% \\ 1.0 \end{pmatrix} \end{pmatrix} \right\rangle .$$

Figure 6.2: The example's resulting multipoint list

Algorithm 9 The complete remapping

```
01 void completeRemapping() {
02    Get available bandwidth for each DiffServ class from SLA.
03    Subtract bandwidths reserved for partial remappings.
04    MultiPointList = <empty>;
05    for each(session in SessionSet) {
06       sessionPoints = calculateSessionMultiPoints(session);
07       MultiPointList ∪= sessionPoints;
08       for each(multipoint in sessionPoints) {
09          if(!tryAllocation(multipoint,session.MinBandwidth)) {
10             break;
11          }
12          else multipoint.AlreadyUsed = true;
13       }
14    }
15    Sort ResouceUtilizationMultiPointList by global sorting value.
16    for each(multipoint in MultiPointList) {
17       if((!multipoint.AlreadyUsed) &&
18          (!multipoint.session.NoMoreTrials)) {
19          if(!tryAllocation(multipoint,session.MaxBandwidth)) {
20             multipoint.session.NoMoreTrials = true;
21          }
22       }
23    }
24    Add bandwidths reserved for partial remappings
25       to available bandwidths.
26 }
```

6.4.1 The Complete Remapping

The algorithm for the complete remapping is shown in algorithm 9. First, the available bandwidths are fetched from the SLA (line 2). Line 3, 24 to 25 are necessary for the partial remapping and will be described later in subsection 6.4.2. At the moment, they can be ignored. The next step is to create a global multipoint list, using the calculation described in section 6.2 to get the multipoints of each session (line 6, see also algorithm 8), joining them (line 7) and finally sorting the complete list (line 15). In line 8 to 13, it is tried to allocate points up to the session's minimum bandwidth, used points are marked. The allocation of the remaining bandwidth up to the session's maximum bandwidth is finally done in lines 16 to 23; points marked during the minimum bandwidth allocation will be skipped.

Algorithm 10 An allocation trial for a resource/utilization multipoint

```
01 bool tryAllocationForMultiPoint(multipoint,limit) {
02    allSuccessful = true;
03    for each(point in multipoint) {
04       if(!tryAllocationForPoint(point,limit))
05          allSuccessful = false;
06       }
07    }
08    return(allSuccessful);
09 }
```

The pseudocode of the allocation trial for a resource/utilization multipoint can be found in algorithm 10. For each point within the multipoint, a resource/utilization point allocation trial will be done (line 4). Note, that the point allocation may fail for some of the streams (e.g. high-bandwidth video streams) while the allocation of other streams may be successful (e.g. low-bandwidth audio streams). This is called *partial multipoint allocation*. In this case, the session is marked for doing no more allocation trials (line 20 of algorithm 9). This is done because for example in the case of a high bandwidth video stream and low bandwidth audio stream, the user does not want to receive and **pay** for a film having high-quality audio but only very poor picture! The parameter *limit* denotes a bandwidth limit for the total bandwidth of the multipoint's session. It is passed to the point allocation trial.

The algorithm for the resource/utilization point allocation trial is shown in algorithm 11. Here, bandwidth is allocated to the layers of the stream to which the point belongs to. In this case, a trial is only successful, if **all** layers get their allocation. This is comparable to a database transaction, where several changes are made in the database. In case of a failure, a so called *rollback* has to be done. After the rollback, the database's contents are the same as before the transaction's start. In line 2, the complete allocation state is saved. Now, the stream's old allocation is released in line 3. In line 4 to 20, it is tried to allocate each layer to the cheapest possible DiffServ class (see descriptions and examples in section 6.2.2): For each layer, all possible layer to DiffServ class mappings are checked (line 6). If the class has got enough free bandwidth (line 6) and the allocation will not exceed the session's bandwidth limit (line 7), the allocation will be made (line 9 to 11). Since the possibilities are sorted ascending by cost, no more checks are necessary after a successful allocation (line 13). If one of the layers can not get an allocation, a complete rollback of all allocations is done (line 16 to 19). Then, the stream's allocation will be the same as before the allocation trial.

Now, it is the time to give an example. One session containing two streams is given. The first stream has got one layer, the second one two. The global multipoint list contains the following two

Algorithm 11 An allocation trial for a resource/utilization point

```
01 bool tryAllocationForPoint(point,limit) {
02    Set savepoint.
03    Release old allocation.
04    for each(layer in point) {
05       success = false;
06       for each(dsClass in layer.PossibleDSClassMappings) {
07          if((dsClass.Available ≥ layer.Bandwidth[dsClass]) &&
08             (resulting total bandwidth of session below limit)) {
09             layer.DiffServClass = dsClass;
10             layer.Allocated     = layer.Bandwidth[dsClass];
11             dsClass.Available  -= layer.Bandwidth[dsClass];
12             success = true;
13             break;
14          }
15       }
16       if(!success) {
17          Do rollback.
18          return(false);
19       }
20    }
21    return(true);
22 }
```

Class	Cost factor	Bandwidth
EF	4.0	500 KBytes/s
AF31	3.0	2200 KBytes/s

Class	Cost factor	Bandwidth
EF	4.0	500 KBytes/s
AF31	3.0	3500 KBytes/s

Table 6.9: SLAs for the bandwidth management example

multipoints (For simplification, the fields for frame rate, sorting value and buffer delay are not shown):

$$\left(\begin{array}{l} \left(\begin{array}{c} 100\,\mathrm{KB/s} \\ 50\,\% \end{array} \left[\left\langle \begin{array}{cc} \mathrm{EF} & \mathrm{AF31} \\ 100\,\mathrm{KB/s} & 200\,\mathrm{KB/s} \end{array} \right\rangle \right] \right) \\ \left(\begin{array}{c} 900\,\mathrm{KB/s} \\ 80\,\% \end{array} \left[\begin{array}{l} \left\langle \begin{array}{cc} \mathrm{EF} & \mathrm{AF31} \\ 300\,\mathrm{KB/s} & 700\,\mathrm{KB/s} \end{array} \right\rangle \\ \left\langle \begin{array}{cc} \mathrm{AF31} & \mathrm{EF} \\ 600\,\mathrm{KB/s} & 500\,\mathrm{KB/s} \end{array} \right\rangle \end{array} \right] \right) \end{array} \right), \tag{6.7}$$

$$\left(\begin{array}{l} \left(\begin{array}{c} 200\,\mathrm{KB/s} \\ 60\,\% \end{array} \left[\left\langle \begin{array}{cc} \mathrm{EF} & \mathrm{AF31} \\ 200\,\mathrm{KB/s} & 400\,\mathrm{KB/s} \end{array} \right\rangle \right] \right) \\ \left(\begin{array}{c} 1300\,\mathrm{KB/s} \\ 95\,\% \end{array} \left[\begin{array}{l} \left\langle \begin{array}{cc} \mathrm{EF} & \mathrm{AF31} \\ 600\,\mathrm{KB/s} & 1500\,\mathrm{KB/s} \end{array} \right\rangle \\ \left\langle \begin{array}{cc} \mathrm{EF} & \mathrm{AF31} \\ 700\,\mathrm{KB/s} & 1800\,\mathrm{KB/s} \end{array} \right\rangle \end{array} \right] \right) \end{array} \right). \tag{6.8}$$

Table 6.9 shows two example SLAs. First, the left one is used. For the first multipoint (formula 6.7), the allocation consists of allocating 100 KBytes/s EF to stream #1, layer #1 and to stream #2 300 KBytes/s EF to layer #1 and 600 KBytes/s AF31 to layer #2. Now, 100 KBytes/s EF and 900 KBytes/s AF31 are remaining. For the next multipoint (formula 6.8), additional 100 KBytes/s EF (200 KBytes/s - 100 KBytes/s are already allocated) are allocated to the only layer of stream #1. Now - since there is no remaining EF bandwidth - it is not possible to allocate additional 300 KBytes/s

(600 KBytes/s total) to layer #1 of stream #2. Instead, 1500 KBytes/s of AF31 can be used. The previously allocated 300 KBytes/s EF are freed. Layer #2 requires 700 KBytes/s of EF or additional 1200 KBytes/s AF31 (600 KBytes/s are already allocated to this layer). Since both is not available, this layer's allocation fails here. As mentioned in the description of the resource/utilization point allocation, a rollback has now to be done for all layers of this stream. Note, that stream #1's allocation was successful, therefore this allocation will be kept. However, since the session has got one failed point allocation for this multipoint, no more allocation trails will be made for following multipoints of this session. The resulting bandwidth mapping is 200 KBytes/s for the only layer of stream #1 and 300 KBytes/s EF (layer #1)/600 KBytes/s AF31 for stream #2.

Now, using the right SLA of table 6.9, the resulting bandwidth mapping is 200 KBytes/s EF for the only layer of stream #1 and 1500 KBytes/s AF31 (layer #1)/1800 KBytes/s AF31 (layer #2) for stream #2. The total cost will therefore be:

$$\underbrace{200 * 4.0}_{\text{Stream } \#1} + \underbrace{1500 * 3.0 + 1800 * 3.0}_{\text{Stream } \#2} = 10,700.$$

Assuming an SLA that contains enough bandwidth to allocate all layers to the first (= best) layer to DiffServ mapping possibility, the resulting mapping would be 200 KBytes/s EF for stream #1 and 600 KBytes/s EF (layer #1)/700 KBytes/s EF (layer #2) for stream #2. Then, the total cost would only be:

$$\underbrace{200 * 4.0}_{\text{Stream } \#1} + \underbrace{600 * 4.0 + 700 * 4.0}_{\text{Stream } \#2} = 6,000.$$

To solve the problem of such inefficient SLAs, the CORAL concept described in section 3.1 contains a dynamic SLA, managed by the bandwidth broker. In case of a 'bad' bandwidth distribution of the classes, the bandwidth manager may request the bandwidth broker to e.g. increase EF by 1000 KBytes/s and decrease AF31 by 3500 KBytes/s in the example above. This can be done e.g. by automatically renegotiating the SLA with an ISP or changing the SLAs of other managed servers. Such change requests can simply be calculated using each stream's resource/utilization point finally allocated and sum up the difference between the real allocation and the best possible choice. The bandwidth broker itself is not part of this work. Therefore, see [Sel01] for more details.

6.4.2 The Partial Remapping

Everytime a stream reaches a new remapping interval, it is necessary do a remapping for this stream. Although measurements in section 8.5 will show, that the complete remapping is quite efficient, it is desirable to optimize it for usage in scenarios using hundreds or even thousands of streams. A simple but effective optimization for the remapping in case of reached interval borders is to do a remapping only for the stream itself. This will be called *partial remapping*.

Algorithm 12 The partial remapping

```
01 void partialRemapping(stream) {
02    // Get resource/utilization point nearest to utilization
03    // of last complete remapping and below session's bw.  limit.
04    point = getNerestForLastUtilization(stream);
05    if(point != NULL) {
06       if(!tryAllocationForPoint(point,point.Session.MaxBandwidth)) {
07          doCompleteRemapping();
08       }
09    } else {
10       doCompleteRemapping();
11    }
12 }
```

The pseudocode of such a partial remapping is shown in algorithm 12. As described in section 5.1, the resource/utilization list changes for every new remapping interval. In this case, the payload to raw translation (see section 6.2.1) and calculation of possible layer to DiffServ class mappings (see section 6.2.2) have to be calculated for the resource/utilization point having nearest utilization to the stream's utilization resulting from the last complete remapping. To avoid extreme quality changes, the point's utilization may also not differ from the last complete remapping's utilization by more than 5%. Further, it may of course not exceed the session's maximum bandwidth limit. This point's calculation is done in line 4. Next, an allocation for this point is tried as described in section 6.4.1 (line 6). If this allocation trial fails, a complete remapping will be invoked (line 7). The same happens, if no usable point has been found (line 10).

To make a partial remapping possible, it is useful not to allocate the SLA's whole bandwidth at the complete remapping. Instead, a certain fraction has to be reserved for partial remappings. This is done in lines 3 and 24 to 25 of algorithm 9. Due to the varying bandwidth requirements of variable bitrate streams, some streams may allocate some additional bandwidth while other ones free some. Therefore, the fraction necessary to reserve for partial remappings during a complete remapping depends on the traffic properties of the scenario's streams.

Using partial remappings, the bandwidth distribution possibly differs from a complete remapping's calculation: The partial remapping only tries to keep the stream's utilization at almost the same level and ensures the session's bandwidth limit. If for example in a video, a small-bandwidth scene is followed by a high-bandwidth action scene, a large amount of additional bandwidth is required to keep the former utilization. To cope with this problem, it is recommended to force a complete remapping regularly, e.g. at least every 5 to 10 seconds. This will result in keeping the difference between the current mapping and a complete remapping's result small.

Of course, the settings for the maximum time between two complete remappings and the bandwidth fraction to be reserved exclusively for partial remappings are strongly dependent on the scenario. For example, the maximum time between two complete remappings may be higher and the reserved bandwidth fraction lower for scenarios containing only streams of low bandwidth variance (or even CBR) and vice versa. If sufficient bandwidth is available for all streams of a scenario, there is no difference of quality, cost and bandwidth between complete remappings and partial remappings. In this case, the partial remapping would always select the 100% utilized resource/utilization point. This is the same as a complete remapping finally would do. But if bandwidth is scarce, partial remappings may introduce inefficiency, since they only take care for keeping the invoking stream's utilization level, but not for bandwidth distribution among other streams. Therefore in section 8.4, examinations of parameter changes are made on a large and realistic video/audio on demand scenario.

6.5 Comments on the Bandwidth Remapping

Fairness for sessions and streams

As described in section 6.2.4, 6.2 and 6.3, the fairness is controlled using the constant $\omega_{\text{Fairness}} \in [0,1] \subset \mathbb{R}$ for streams and $\omega_{\text{SessionFairness}} \in [0,1] \subset \mathbb{R}$ for sessions. Since the user probably wants a fair distribution for the streams within his session (e.g. a video to have the same[3] utilization as its sound), it is therefore recommended to use a high value for ω_{Fairness}, e.g. example $\omega_{\text{Fairness}} = 1$. But globally, it is useful for the provider to have as much sessions as possible getting a high quality. For example, it is not useful if user #1 has got a large session of e.g. 80 MBit/s at 25% utilization and users #2 to #21 have got small sessions of 1 MBit/s at 25%, too. Instead, it is recommended to use an unfair sharing here and give user #1 e.g. 50 MBit/s at 10% utilization and users #2 to #21 e.g. 95% at 2.5 MBit/s. $\omega_{\text{SessionFairness}}$ should therefore be low, e.g. $\omega_{\text{SessionFairness}} = 0$.

[3] Of course, only if both streams have got equal stream priorities.

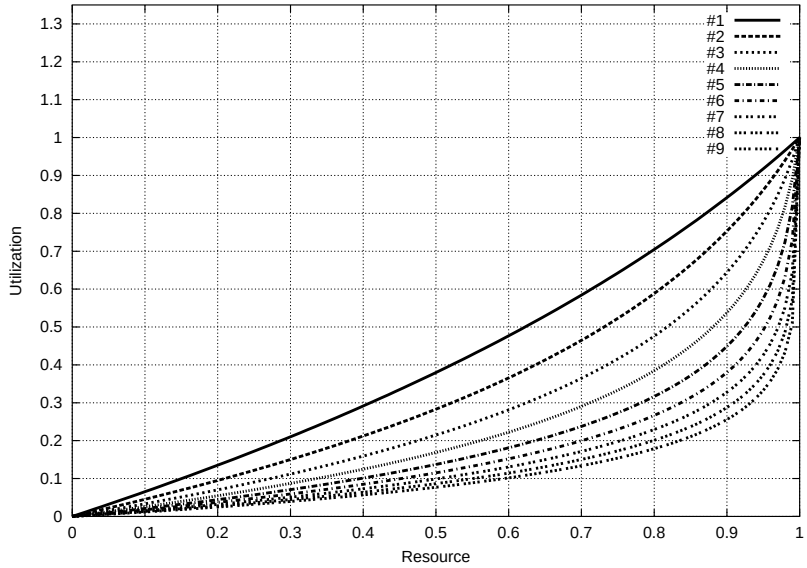


Figure 6.3: Some concave functions

Bandwidth pricing

As described in section 3.1, the bandwidth pricing concept for CORAL introduces three charges: The holding charge for reserving the bandwidth, the usage charge for sending data and a charge for changing the reservation. Since the reserved bandwidth will never be exceeded, holding charge and usage charge may simply be joined. Changing the reservation is not recommended to be charged since changes may be necessary due to adding new streams, removing streams, new remapping intervals, partial remapping not possible, loss scaling, changing bandwidth of other streams, etc.. Further, reservations using DiffServ do not require to send control information over the network. Therefore, charging bandwidth changes would only make the pricing complicated and confusing.

ASRMD1's convex hull and the bandwidth mapping

As described in section 2.5.4, the ASRMD1 algorithm uses the convex hull of the resource/utilization list to remove 'bad' points. This may also be applied to the resource/utilization list of each stream and the resource/utilization multipoint list of each session. But in the case of concave or partially concave utility functions, this will be very inefficient. For example, figure 6.3 shows some concave functions[4]. The convex hull of all these functions only contains two points: (0.0, 0.0) and (1.0, 1.0) - these are the minimum and maximum points. Since such a quality granularity is absolutely inacceptable and there is no evidence for the assumption of only non-concave utility functions, the convex hull calculation has been skipped. The cost of this decision is possibly a slightly higher cost, since points of high resource but low utilization remain in the lists.

Some implementation recommendations

As it is shown in section 6.2 and 6.3, the stream and session description initilizations are parallelizable. But even if a single-CPU system is used, the calculation can be optimized by calculating and storing the stream description initialization directly after the stream has reached a new resource/utilization list. The result is, that it is already available when the complete remapping is executed. Further, the CPU usage distributes homogeneously. And since it is stored and valid until the next resource/utilization list is reached, it may be used for more than one complete remapping. This small optimization

[4]These functions use formula 2.5, parameter $p = 9.0$ to $p = 1.0$.

will save a lot of CPU power, since the stream description initialization is the most CPU-consuming part of the bandwidth management (see section 8.5).

6.6 Summary

In this chapter, the online bandwidth management for multimedia streams of different media types has been developed by extending the ASRMD1 algorithm of section 2.5.4. This has been based on each stream's a priori calculated remapping intervals and traffic descriptions from chapter 4, resource/utilization lists from chapter 5 and each layers' QoS requirements for maximum acceptable transfer delay, loss rate and jitter. First, some online preparations for the resource/utilization lists have been necessary; this has been called stream description initialization:

- Since the size of necessary packet headers for transport via a network is usually not known a priori, a payload⇔raw translation has been developed, first. That is, the necessary number of packets is calculated and the header sizes are added to the resource/utilization lists' bandwidth settings.
- The next step has been the optimization of buffer delay and cost: Based on formula 6.2, a cost-sorted list of all possible layer to DiffServ class mappings is calculated. This leads to the usage of the most efficient class.

Such a stream description initialization may be computed as soon as possible, leading to a homogeneous distribution of CPU usage and a reduced duration of the later remapping itself. Further, efficient parallelization is possible.

Finally, the ASRMD1 algorithm has been extended by

- support for sessions by the usage of so called resource/utilization multipoints,
- independently configurable fairness for streams and sessions,
- priorities for streams and sessions and
- usage of several DiffServ classes instead of only one resource (= bandwidth).

To save CPU power, a reduced version of this complete bandwidth remapping has also been developed: The so called partial remapping only does the remapping for a single stream. Its efficiency is dependent on two parameters: The maximum time between two complete remappings and the bandwidth fraction exclusively reserved for partial remappings.

The chapter has closed with some comments on the developed bandwidth remapping algorithm, concerning fairness, bandwidth pricing and implementation issues.

Chapter 7

The Traces and the System

First, this chapter describes the used MPEG-1, MPEG-2, H.263 and MP3 traces using some statistics. Then, the remapping interval calculation parameters are explained. Next, the choice of utility functions and compositions is presented. Finally, a short overview of important implementation details and the real network scenario is given, since this is necessary to understand the simulations and measurements in chapter 8.

7.1 The Traces

The system is evaluated using traces of MPEG-1, MPEG-2, H.263 and MP3 medias. For MPEG-1, the traces of the University of Würzburg ([Wür95]) are used. H.263 traces have been found at the Technical University of Berlin ([Ber00]) and the University of Bonn (part of an MPEG-4 project). Since there were no traces of VBR-MP3s available, such traces have been generated from VBR-MP3s, encoded from CD tracks using the Open Source encoder LAME (see [LAME]) and downloads via NAPSTER (see [Napster]), as part of this work.

7.1.1 MPEG-1 and MPEG-2

The MPEG-1 (see section 2.6.1) traces are layered as explained in section 4.2: One layer for I-, P- and B-frames. Statistics of the used traces can be found in table A.1. All traces have got a length of 25,000 frames at a frame rate of 25 frames/s and therefore a playtime of 1,000 seconds. Their GoP is "IBBPBBPBBPBB". The frame rate may be scaled in steps of one from 1 frame/s to 25 frames/s (see section 4.5 for details about frame rate scalability). The frame size may be scaled at a maximum scale factor α=0.5 (see section 4.6 for details about frame size scalability). Using methods like coefficient elimination and block dropping (see section 2.6.1), this is a reasonable value.

Since there were no layered MPEG-2 medias or traces available, MPEG-2 (see section 2.6.2) traces are calculated from the MPEG-1 traces by adding two MPEG enhancement layers. Therefore, the number of transport layers is 9 (see section 4.2): Each MPEG layer (one base + two enhancement

I_B	P_B	B_B	I_{E1}	P_{E1}	B_{E1}	I_{E2}	P_{E2}	B_{E2}
10000	0	0	20000	0	0	30000	0	0
0	0	2500	0	0	5000	0	0	7500
0	5000	0	0	10000	0	0	15000	0
Base			1st Enhancement			2nd Enhancement		

Table 7.1: An MPEG-2 trace example with frame sizes in bytes

layers) consists of three transport layers for each frame type: I-, P- and B-frames. The first enhancement layer is calculated from the base layer (= original MPEG-1 trace) multiplying the frame sizes by 2. The second one multiplies these sizes by 3. An example is shown in table 7.1: The base layer's I-frame size of 10,000 bytes results in 20,000 bytes and 30,000 bytes for the first and second enhancement layer. Factors of 2 and 3 seem to be useful, since the resulting partitions have got a fraction of about 17% for the base layer, 33% for the first and 50% for the second enhancement layer. This is achievable using SNR scalability, data partitioning and spatial scalability as explained in section 2.6.2. Statistics for the resulting MPEG-2 traces can be found in table A.2.

7.1.2 H.263

The H.263 (see section 2.6.3) traces are layered as explained in section 4.2: One layer for I-, P-, PB- and B-frames. Statistics of the used traces can be found in table A.3, their playtime is also given there. All traces use a maximum frame rate of 30 frames/s, it may be scaled in steps of one from 1 frame/s to 30 frames/s. The frame size may be scaled at a maximum scale factor α=0.5. Again, see section 4.5, section 4.4.2 and section 2.6.1 for details about scalability.

7.1.3 MP3

The MP3 (see section 2.6.4) traces contain only one layer. As it is described in section 4.2, layering would be too inefficient for MP3 due to the header overhead. The used traces are all generated from VBR-MP3 files. These are encoded from CD tracks using the Open Source MP3 encoder LAME with highest quality setting[1] at a maximum rate of 320 KBit/s or downloaded via NAPSTER. Statistics of the traces can be found in table A.4. Their playtime is also given there. All traces have got the constant frame rate of 38 frames/s. As explained in section 2.6.4, MP3 does not provide frame rate scalability. The frame size may be scaled at a maximum scale factor α=0.25. This will reduce the bandwidth requirement by a maximum of 75% and therefore results in an average frame size of about 150 to 180 bytes, which is an achievable value. See also section 4.4.2 and section 2.6.4 for details about scalability.

7.1.4 The Remapping Intervals

From every MPEG-1, MPEG-2 and H.263 trace, two remapping interval versions have been generated: One version without layer weighting and one weighted version. See section 4.3 for details. The used weights for the corresponding layers are:

- MPEG-1: $\underbrace{4}_{\text{I}} / \underbrace{3}_{\text{P}} / \underbrace{2}_{\text{B}}$
- MPEG-2: $\underbrace{4 / 3 / 2}_{\text{Base layer}} / \underbrace{1.5 / 1.3 / 1.2}_{\text{1st Enhancement}} / \underbrace{1.1 / 1.0 / 1.0}_{\text{2nd Enhancement}}$,
- H.263: $\underbrace{4}_{\text{I}} / \underbrace{3}_{\text{P}} / \underbrace{3}_{\text{PB}} / \underbrace{2}_{\text{B}}$ and
- MP3: 1.

These settings are made under the assumption that I-frames usually require more expensive bandwidth than P-frames, P-frames more than B-frames and MPEG-2 enhancement layers may use BE

[1] LAME parameters: -V 0.

	Low frame size	High frame size
Low frame rate	Talk, Test pattern	Comedy, Snooker
High frame rate	Advertisements, Station break	Music, Sports

Table 7.2: The results of [AFK+95]

	Low frame size	High frame size
Low frame rate	Talk	Comedy, Movie, News
High frame rate	Cartoon	Action, Music, Sports

Table 7.3: Used genre mappings for the traces

bandwidth. PB-frames of H.263 are handled like P-frames. Since MP3 has got only one layer, weighting is senseless here.

The remapping cost (see section 2.4) has been set to 500,000 for the video traces and 50,000 for the MP3 traces[2]. The maximum buffer delay has been set to 1000ms for the MPEG traces and 500ms for the H.263 and MP3 traces. To limit the calculation time (see section 4.5), a minimum remapping interval length of one second and a maximum one of 20 seconds have been used. As explained in section 6.4.2, there is a complete remapping regularly anyway, due to the partial remappings. Therefore, longer intervals would not be useful. Using these settings, the average calculation time for the remapping intervals of one trace are about 3 days for MPEG-1, 4.5 days for H.263 and 6.5 days for MPEG-2 on a 300 MHz Pentium-II system. The trace statistics of table A.1 (MPEG-1), table A.2 (MPEG-2), table A.3 (H.263) and table A.4 (MP3) show the number of remapping intervals for the highest frame rate in the $N_{FrameRate}$ column.

7.2 The Resource/Utilization Lists

Finally, resource/utilization lists are calculated for the traces using the algorithm described in chapter 5. Therefore, it is necessary to choose utility functions for the frame rate and frame size of each layer. Since only traces are used, it is not possible to use quality metrics to calculate such functions as described in section 2.5.1. Instead, the following approximation is made:

In [AFK+95], user ratings for the importance of frame rate (called temporal component) and frame size (called visual component) using video sequences of different genres (e.g. sports, comedy, talk etc.) have been made. The results of these so called *video watchability* experiments are shown in table 7.2: Based on these results, the MPEG-1/2 and H.263 traces' genres are mapped to the different frame rate and frame size requirements as shown in table 7.3. 'Action' may be compared to 'Sports', 'News' and 'Movie' do not have such a high frame rate requirement as sports and music, but its visual component is important. 'Cartoons' (here: "*Simpsons*" and "*Asterix*") may be compared to 'Advertisements', having a high temporal component but a low visual one.

Next, utility functions for each genre had to be chosen for frame rate and frame size. For both dimensions, the utility function from [Rog98], shown in formula 2.5 and described in section 2.5.1, is used, due to its properties:

- $u(0.0) = 0.0$, $u(1.0) = 1.0$ and
- $u(\vartheta)$ may be convex, linear or concave, depending on the sensitivity parameter p.

[2] In case of 500,000 for the MP3 traces, all intervals would have the maximum length of 20 seconds - due to the small size of MP3 files. Therefore, 50,000 is used here.

As described in section 2.5.1, the utility function from [LS98] and [LLR+99] (see formula 2.6) has got the disadvantages of being unsteady at $x = 0.0$ and $x = 1.0$ and always being convex. Therefore, this function is not used here.

For simplification, each layer's frame size has got the same utility function. To emphasize the different importances of each layer, a weighting is useful here. Therefore, it is reasonable to use the weighted sum (see formula 2.7) from [LS98] and [LLR+99] to compose the frame size utility functions of all layers. Formula 2.8 presents the composition function from [Rog98]. Its advantage is, that settings like 100% frame size utilization and 0% frame rate utilization result in a total utiliztion of 0%, which is much more realistic than e.g. 50% as for a weighted sum. For example, a viewer usually does not accept to view a sports video at best picture quality but only 1 frame/s (like a slideshow). Therefore, formula 2.8 is used to compose the frame rate utilization and the **normalized** weighted utilization sum of all layers. The resulting utility function is therefore:

$$U(\overrightarrow{\vartheta}) = \frac{2}{\frac{1}{u_{\text{FrameRate}}(\vartheta_0)} + \frac{1}{\underbrace{\frac{1}{\sum_{n=1}^{l}\omega_n}\underbrace{\sum_{n=1}^{l}[\omega_n * u_{\text{FrameSize}_n}(\vartheta_n)]}_{\text{Frame Size Utilization Sum}}}_{\text{Normalized Frame Size Utilization}}}}, \tag{7.1}$$

where l denotes the number of layers and $\overrightarrow{\vartheta} \in [0,1]^{l+1} \subset \mathbb{R}^{l+1}$ the scale factor vector as described in section 5.1. The utility functions for frame rate and size are:

$$u_{\text{FrameRate}}(\vartheta) := c_0 * \ln(a_o * \vartheta + b_o),$$
$$u_{\text{FrameSize}_i}(\vartheta) := c_i * \ln(a_i * \vartheta + b_i).$$

For each QoS dimension j, a_j, b_j and c_j are calculated from the corresponding sensitivity parameter p_j as described in section 2.5.1. ω_n denotes the weight for layer #*n*. These weights are set to the following values:

- MPEG-1: $\underbrace{3}_{\text{I}}$ / $\underbrace{2}_{\text{P}}$ / $\underbrace{1}_{\text{B}}$
- MPEG-2: $\underbrace{9/8/7}_{\text{Base layer}}$ / $\underbrace{6/5/4}_{\text{1st Enhancement}}$ / $\underbrace{3/2/1}_{\text{2nd Enhancement}}$,
- H.263: $\underbrace{3}_{\text{I}}$ / $\underbrace{2}_{\text{P}}$ / $\underbrace{2}_{\text{PB}}$ / $\underbrace{1}_{\text{B}}$ and
- MP3: 1.

It is important to make a distinction between the layer weights used for the remapping interval calculation and the utilization weights here: The first ones refer to the layers' cost during transmission, but these values refer to the layers' importance for the user satisfaction!

For low requirements of frame rate or frame size, the parameter $p_{\text{Low}} = 12.5$ is used. This results in a convex utility function, giving higher utilization already for low settings. On the other side, for high frame rate or frame size requirements, the parameter $p_{\text{High}} = 7.5$ is used. The resulting concave utility function results in low utilization for low settings. These selections of the sensitivity parameter p should be reasonable for the used set of traces. Both functions are plotted in figure 7.1. For comparision, it also includes the linear utility function $u_{\text{Linear}}(\vartheta) = \vartheta$. Now, it is possible to plot the user satisfaction of all four possible frame rate/frame size requirement combinations: Low/Low, Low/High, High/Low and High/High. Table 7.4 shows the results.

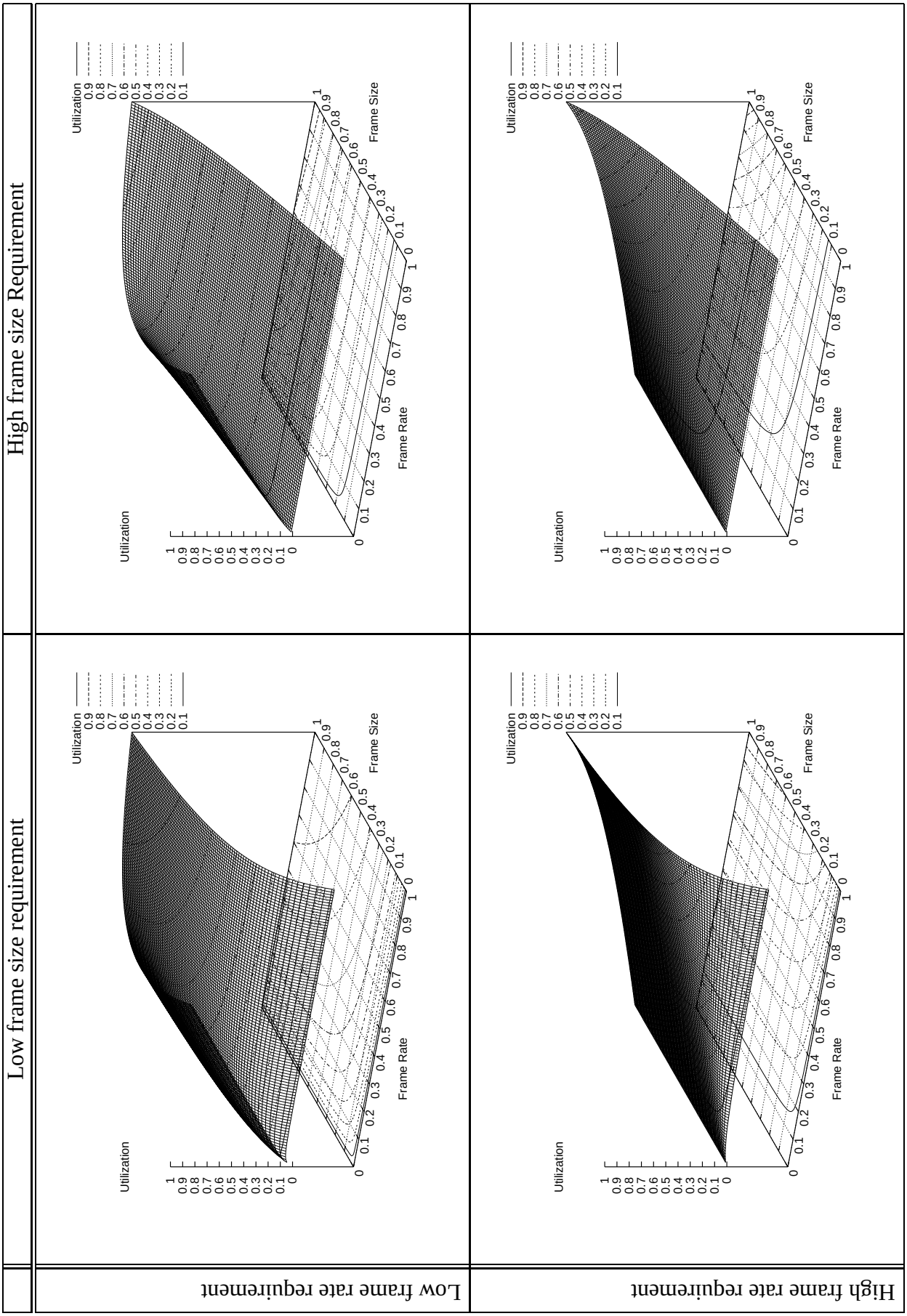


Table 7.4: User satisfaction for different frame rate and frame size requirements

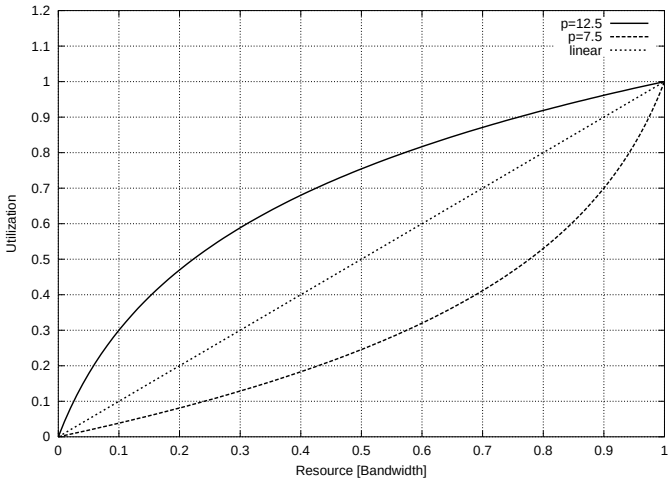


Figure 7.1: Utility functions for video frame rate and size

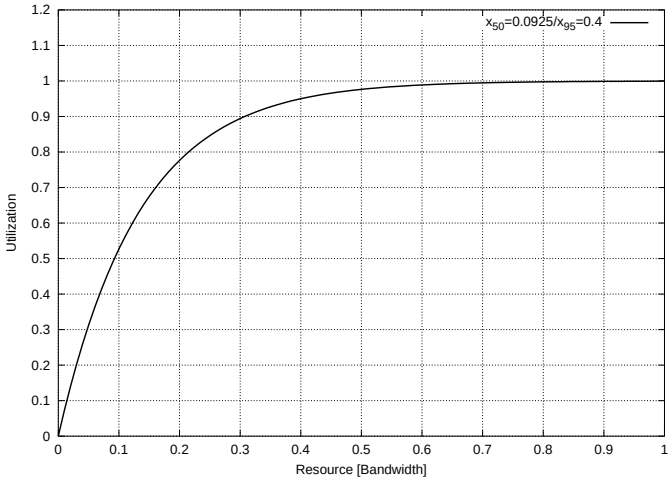


Figure 7.2: Utility function for audio frame size

For the utilization of the MP3 audio traces, no division into different genres is done, since there were no such user ratings available as for video genres in [AFK+95]. Therefore, a generic utility function for all types is used, based on the utility function from [LS98] and [LLR+99] (see formula 2.6). Since this function can be configured by giving utilizations for 50% and 95% resource, this function seems to be a good choice here: As mentioned in section 2.6.4, MP3 is optimized to compress high quality audio at a bandwidth of about 112 to 128 KBit/s. These are also the most common bandwidths used for CBR files downloadable via NAPSTER. The LAME encoder used for creating most of the used MP3s supports bandwidths from 8 KBit/s to 320 KBit/s, always selecting the best choice for the current frame in VBR mode. Therefore, it seems to be reasonable to set 95% utilization for 40% bandwidth (comparable to about 125 KBit/s using CBR) and 50% utilization for 10% bandwidth (comparable to about 32 KBit/s using CBR). The resulting utility function is plotted in figure 7.2.

The QoS requirements of each media type's layers for maximum jitter and loss rate are given in table 7.5. These values are used for the simulations and measurements in chapter 8. The maximum transfer delay depends on each simulation or measurement. Therefore, it will be given in its explaination.

H.263		
Layer	MaxLossRate	MaxJitter
#1 (I)	0.5 %	-
#2 (P)	1.0 %	-
#3 (PB)	1.0 %	-
#4 (B)	2.0 %	-

MP3		
Layer	MaxLossRate	MaxJitter
#1	5 %	100 ms

MPEG-1/2		
Layer	MaxLossRate	MaxJitter
#1 (I_B)	0.5 %	-
#2 (P_B)	1.0 %	-
#3 (B_B)	2.0 %	-
#4 (I_{E1})	-	-
#5 (P_{E1})	-	-
#6 (B_{E1})	-	-
#7 (I_{E2})	-	-
#8 (P_{E2})	-	-
#9 (B_{E2})	-	-

Table 7.5: Used jitter and loss rate limits for each media type

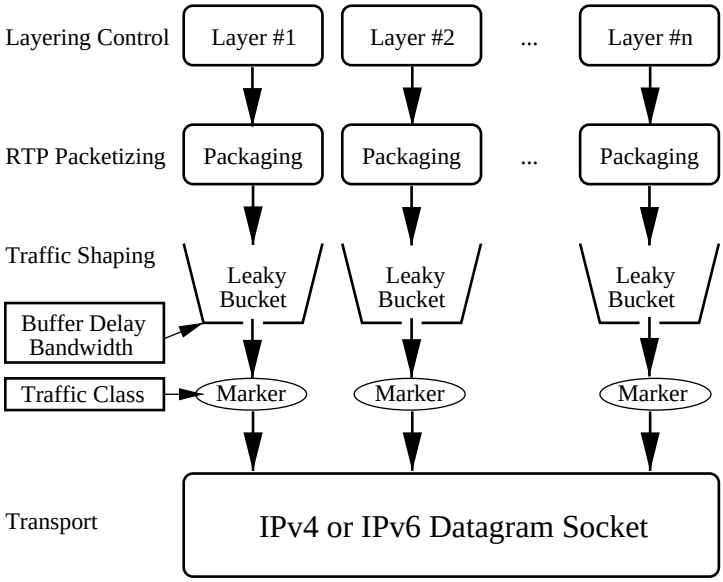


Figure 7.3: An overview of the layering, RTP transport and traffic shaping implementation

7.3 The System

For better understandability of the simulations and measurements in chapter 8, a few details of the system's implementation have to be explained. This can only be a short overview, since a detailed introduction would be out of this work's scope. Details can be found in the RTP AUDIO project's documentations (see [DSV00] and [RTP Audio]), since large parts of this project are reused.

As shown in figure 7.3, each transport layer has got its own leaky bucket buffer, but all output uses the same UDP/IPv4 or UDP/IPv6 socket. Before transmission, the packet's traffic class/TOS field is set to the corresponding transport layer's value, mapped to it by the bandwidth manager. Since the buffer implementation consists only of a queue of packets, the usage of 9 transport layers to transmit a 3-layered MPEG-2 stream does not consume significantly more CPU power than using a single transport layer for all MPEG-2 enhancement layers. Due to the single UDP/IP socket, it also does not require more UDP ports. Each leaky bucket is policed using formula 2.1. But this assumes, that each frame is scheduled accurately, that is e.g. at a frame rate of 25 frames/s, the next frame is appended to the queue exactly after $\frac{1}{25}th$ second. Of course, since a standard LINUX system is used for the measurements, such an accuracy can not be guaranteed. Therefore, some system delay tolerance as described in section 6.2.2 has to be allowed to avoid buffer overflows.

Class	Bandwidth	Cost factor	Variability	Delay	Loss rate	Jitter
BE	1 GBytes/s	1.0	25 %	600 ms	10.00 %	200 ms
AF11	1 GBytes/s	2.0	10 %	300 ms	2.00 %	100 ms
AF21	1 GBytes/s	2.35	10 %	260 ms	1.00 %	80 ms
AF31	1 GBytes/s	2.65	10 %	230 ms	0.75 %	65 ms
AF41	1 GBytes/s	3.0	10 %	200 ms	0.25 %	50 ms
EF	1 GBytes/s	4.0	5 %	100 ms	0.05 %	10 ms

Table 7.6: Standard class settings for the buffering, layering and weighting simulations

The measurements of chapter 8 consist of simulations and measurements on a real DiffServ scenario. The usage of simulations has got the following reasons:

- Due to the inaccuracy introduced by scheduling and varying network delays, different cost and bandwidth measurements are difficult to compare. Of course, the same measurement is not exactly repeatable, too. This makes verification extremely difficult.
- Further, the real system uses an own thread to serve each client. Since each client sends its commands via RTCP APP messages to the server, no serving order can be guaranteed for the clients.
- The real DiffServ scenario (see explaination below) consists of only two DiffServ routers in a local network, their distance is only a few meters. Therefore, the delays of each DiffServ class only differ in the range of a few microseconds. This is much too low for a realistic examination of the cost-optimized buffering.

To cope with these problems, most measurements are done using simulations: Instead of sending data via a real network, bandwidth management statistics are recorded. The settings for loss rate, jitter and transfer delay are set to constant values. Further, task scheduling and the simulation time are controlled by the simulator itself. This makes cost and bandwidth simulations of different scenarios comparable and repeatable. Figure 7.4 gives an overview of the simulator's scope. The other parts of this figure belong to the real scenario and will be described below.

Table 7.6 shows the standard DiffServ class settings for most simulations of chapter 8: BE, four AF classes and EF are used. All classes have got a given constant transfer delay, loss rate and jitter. These values are constant to simplify the comparision of different simulations. The effects of changing these values are examined in section 8.2.1. The reasons for this choice of settings in table 7.6 are as follows: A transfer distance of 15,000km (e.g. Bonn/Germany to Los Angeles/U.S.A.) via optical fibre lines is assumed, implying a signal speed of $\frac{2}{3}c_o$ ($c_o \approx 300,000\,\mathrm{km/s}$ is the vacuum light speed). Therefore, the signal requires 75ms for the given distance. An average amount of 25 hops is assumed. Each hop introduces a delay for receiving, queuing and transmitting a packet. For receiving and sending, a 100 MBit/s line is assumed. Therefore, 120μs are required to send or receive a packet of 1500 bytes, implying 240μs per packet total. For the routing itself (checking the destination address, decrementing the hop limit, calculating the IPv4 checksum, appending the packet to one of the queues, etc.), 500μs are assumed. For each DiffServ class, different average queuing delays are assumed:

- EF: An average of 2 packets is assumed already being in the queue, since EF queues are quite small (see section 2.2.3). Therefore, the total delay per hop is (2*120μs) + 240μs + 500μs = 980μs. For 25 hops and 75ms real transfer time, this makes about 100ms total transfer delay. Further, small queues introduce only a small jitter. Since EF's bandwidth limit may not be exceeded, the loss rate setting is very low.

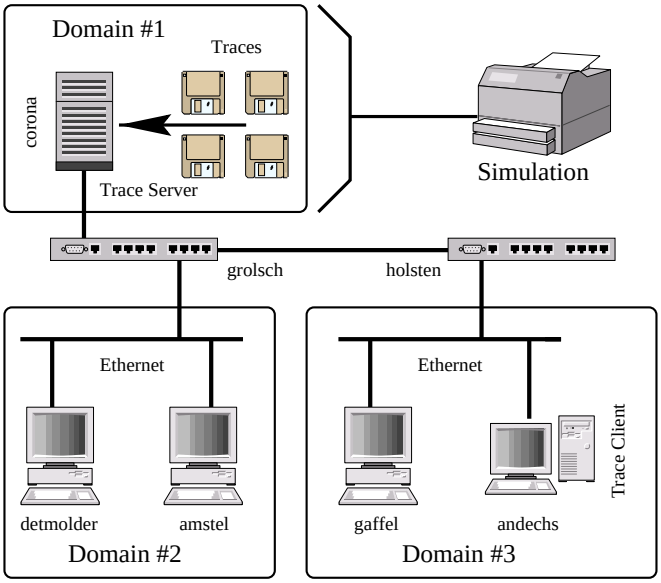


Figure 7.4: The real DiffServ scenario

- AF: Averages of 35 (AF41), 45 (AF31), 55 (AF21) and 70 (AF11) packets are assumed already being in the queue, since AF classes have got longer queues and different priorities (see section 2.2.3). The resulting total transfer delays are therefore about 200ms, 230ms, 260ms and 300ms. Further, higher jitters are assumed. Due to allowed congestion in AF and the RED algorithm's dropping strategy (see section 2.2.3), the loss rates are higher than for EF.

- BE: Tests using *ping* via BE from Bonn/Germany to Los Angeles/U.S.A. resulted in round trip times of about 1100ms to 1200ms. Therefore, a transfer delay of 600ms is quite realistic. Note, that the best effort data may be routed via satellite, which is cheaper but much slower than via cable! A satellite link introduces a transfer delay of 270ms (see [Tan96], page 328) per usage. Due to congestion and longer queues, a high loss rate and jitter is chosen.

The variability column gives the delay variability of each class, used in formula 6.2 (see section 6.2.2). Further, the bandwidth of each class is set to an amount sufficient to always ensure 100% utilization and usage of the most cost-efficient class. Due to the possibility of renegotiating the SLA with the bandwidth broker (see [Sel01]), this is a realistic assumption.

In the case of simulation, a buffer overflow may only happen **once** after a frame rate[3] or buffer delay change: If a layer has got a high buffer delay (e.g. due to using EF) and now changes to a slower class (e.g. BE) having a smaller buffer delay, the buffer contents are not necessarily flushed. If possible, they are adapted to the new settings of bandwidth and buffer delay. This has been called *speed adjustment approach* and is done to avoid visible or audible errors caused by dropped packets. The result may be a single buffer overflow, possibly some time later, depending on the traffic's behavior. But after such a buffer overflow and completely emtying the buffer (buffer flush), no more exceeds are possible until the next quality change, due to the a priori calculated traffic descriptions.

Using a real system, additional buffer overflows can be caused by inaccurate process scheduling, e.g. a set of packets is added to the leaky bucket's queue too early or too late. Therefore, the measurements in section 8.5 examine different delay tolerance settings in a real system consisting of two DiffServ routers. The used DiffServ scenario is shown in figure 7.4. All hosts run LINUX using 2.2.x kernels. The trace server runs on *corona* (Pentium-III, 500 MHz), sending data via the DiffServ routers *grolsch* and *holsten* (both Dual Pentium-II, 300 MHz) to the clients running on *andechs*

[3]Since each frame rate has got its own remapping intervals and therefore traffic descriptions.

Class	Bandwidth	Cost factor	Variability
BE	1,250,000 Bytes/s (10 MBit/s)	1.0	25 %
AF11	1,250,000 Bytes/s (10 MBit/s)	2.0	10 %
AF21	1,250,000 Bytes/s (10 MBit/s)	2.5	10 %
EF	625,000 Bytes/s (5 MBit/s)	4.0	5 %

Table 7.7: The DiffServ scenario's SLA

(Pentium-II, 300 MHz). The routers use the DIFFSERV ON LINUX implemenation. A detailed description can be found at [LinuxDS]. The scenario's supported classes are BE, AF11, AF21 and EF, the routers' SLA is shown in table 7.7. *detmolder* (Dual Pentium-II, 300 MHz), *amstel* (Pentium-II, 300 MHz) and *gaffel* (Pentium-II, 175 MHz) are used to generate background traffic, e.g. UDP and TCP traffic via BE. As explained in section 3.2, the transfer delay of each class is measured using ICMP echo requests and replies. Loss rate and jitter are obtained from RTCP receiver reports (see section 2.1.5). It is important to denote here, that buffer flushes do not cause increased loss rate calculation, since the RTP sequence numbers are adjusted after a flush. This is useful, since the loss rate has to measure the network's quality.

7.4 Summary

This chapter has described the used MPEG-1, MPEG-2, H.263 and MP3 traces using some statistics. Then, the used remapping interval calculation parameters have been explained. Further, the choice of utility functions and compositions has been presented. Finally, a short overview of important implementation details and the real network scenario has been given, since this is necessary to understand the simulations and measurements in chapter 8.

Chapter 8

Measurements and Evaluation

Evaluation of the system by simulations and real network measurements is the goal of this chapter. First, the effects of buffering and the weighted remapping interval calculation are explained by comprehensive examples. Then, buffering, layering, weighting and scalability are examined for all traces described in section 7.1. Further, the system's behavior on network quality changes (e.g. packet losses) and the functionality of the priority system are shown. Next, the system is examined using a large, realistic video/audio on demand scenario to show its practical usability. Then, complete and partial remapping settings are tested using this scenario. Finally, different system delay tolerances are examined in a real network scenario. This also includes the measurement of durations for stream description initializations and complete remappings.

8.1 Buffering, Layering, Weighting and Scalability Simulations

8.1.1 A Buffering Example

This simulation's intention is to explain the buffering and the delay optimization of the bandwidth manager by a detailed example. No comparision between different transfer delay limits and cost is done here. Such simulations can be found in section 8.1.3.

In this example, the MPEG-1 video "*The Silence of the Lambs*" with a maximum transfer delay of 350ms has been transmitted for a duration of 120 seconds. The used DiffServ class settings are shown in table 7.6. Due to the layers' QoS settings for maximum loss rate as shown in table 7.5, layer #1 (I-frames) can only use EF and AF41. Layer #2 (P-frames) may also use AF31 and AF21. Finally, layer #3 (B-frames) can also use AF11. Due to BE's loss rate of 10%, this class is unusable here.

Figure 8.1 presents the buffering gain of layer #1 (I-frames) on the left side: The "*Layer #1, With Buffering*" graph shows the reserved bandwidth for using the maximum possible buffer delay as described in section 6.2.2 by formula 6.2. For comparision, the required bandwidth without buffering is presented by the "*Layer #1, Without Buffering*" graph. As it is shown, this requirement is usually about 4 times higher. Therefore, there is a huge buffering gain for layer #1. On the right side of figure 8.1, the layer's mapping to the DiffServ classes can be found: In this case, only EF is used for the complete transmission. Due to the high short-term burstiness of I-frames (one I-frame after 11 other frames for this video, see section 7.1), it is cheapest to transmit via the expensive but fast EF class here and use the saved time for buffering.

Next, figure 8.2 shows the same contents for layer #2 (P-frames). As expected, the buffering gain is slightly smaller, since there are only two B-frames or 4 B-frames and one I-frame between two P-frames. Therefore, this layer uses AF21 more than half of the time, sometimes AF31 and AF41 and some more frequently EF. Since the AF classes are slower, a much smaller buffering gain can be noticed when these classes are used. However, the used class is cheapest, since the gain of a faster class would not exceed its additional cost (see section 6.2.2).

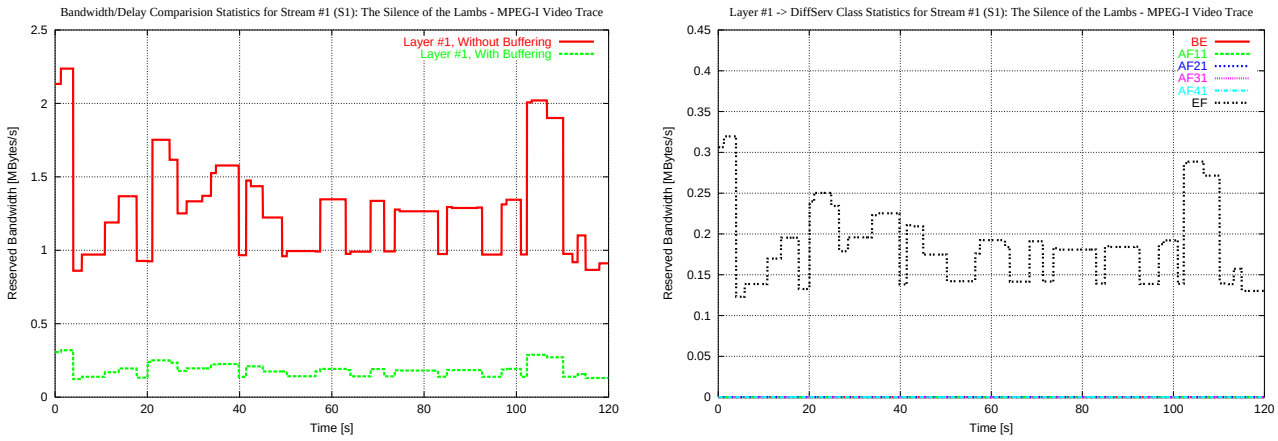


Figure 8.1: Buffering gain and DiffServ class usage for layer #1 (I-frames)

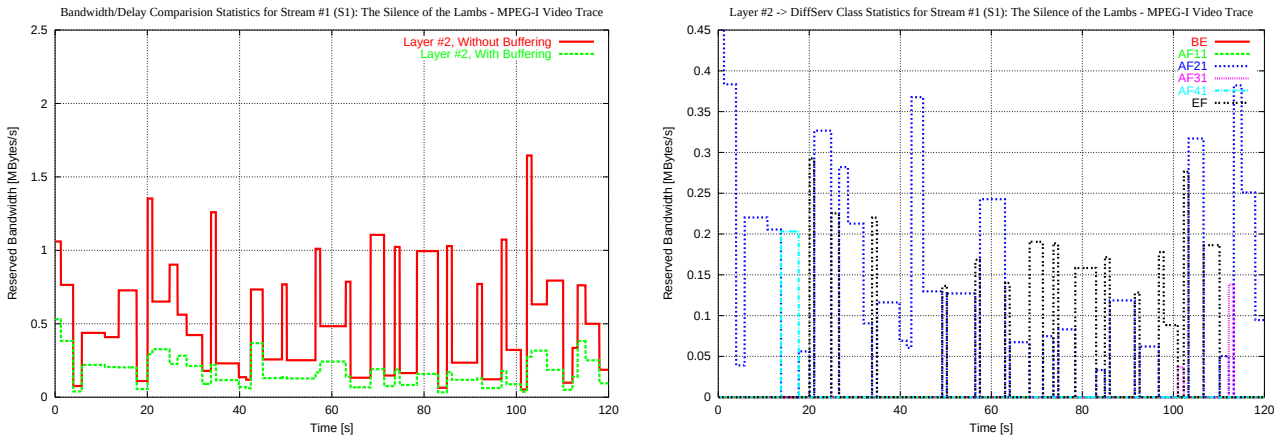


Figure 8.2: Buffering gain and DiffServ class usage for layer #2 (P-frames)

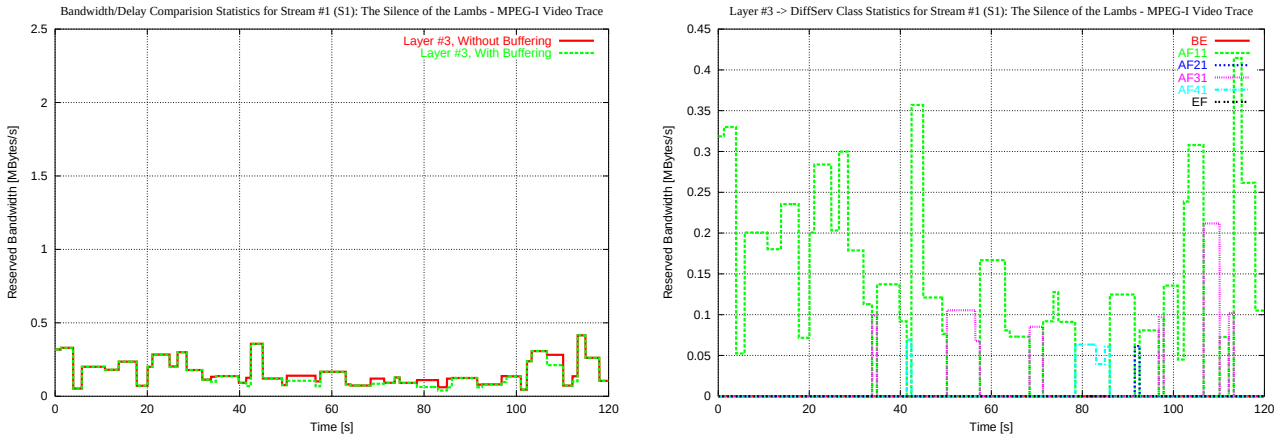


Figure 8.3: Buffering gain and DiffServ class usage for layer #3 (B-frames)

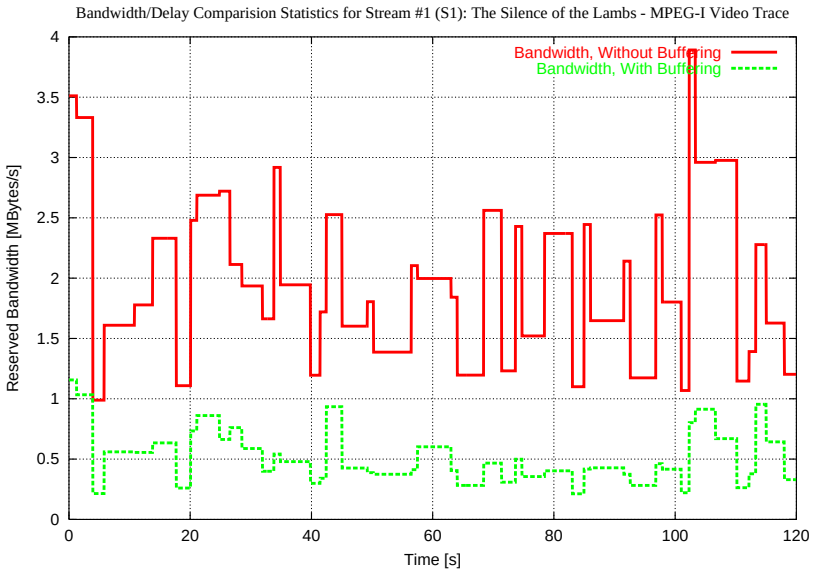


Figure 8.4: Reserved bandwidth for MPEG-1 video "*The Silence of the Lambs*"

Finally, the buffering gain and DiffServ class mappings of layer #3 (B-frames) can be found in figure 8.3. Since the short-term burstiness for B-frames is low (only one I- or P-frame between 4 B-frames for this video, see section 7.1), AF11 is cheapest most of the time. Due to the transfer delay limit of 350ms and AF11's transfer delay of 300ms and delay variance of 10%, no buffering is possible here (see formula 6.2): A delay of only 20ms is available for buffering, but 40ms are required to buffer an additional frame ($\frac{1}{25}th$ second, due to the frame rate of 25 frames/s). Only in a few cases, buffering is efficient for this layer. In these cases, AF31 and AF41 are used, as shown on the right side of figure 8.3.

The total bandwidth gain for all layers is shown in figure 8.4. Again, the bandwidth required for buffering enabled can be found in the "*Bandwidth, With Buffering*" plot. The other plot "*Bandwidth, Without Buffering*" shows the bandwidth required without any buffering. As it is shown, there is a huge buffering gain of about 75% in this example - although only a few milliseconds have been used for buffering!

Results

As it is shown for the video example "*The Silence of the Lambs*", buffering may result in a huge bandwidth gain. Further, the system always uses the most cost-efficient class for the transport. Now, the total gain has to be examined for a wide range of maximum transfer delay limits and the complete set of traces. This is done in section 8.1.3.

8.1.2 A Weighting Example

Before the bandwidth and cost comparision simulation can be realized, it is first necessary to explain another fact: The weighted remapping interval calculation. As shown in section 4.3, the traffic description's bandwidth of each layer is weighted. This will result in higher cost for 'expensive' layers. Therefore, new remapping intervals are started earlier, trying to save expensive bandwidth. The intention of this simulation is to show this behavior by a detailed example. Comparisions of different traces and transfer delay limits can be found in section 8.1.3.

Again, this simulation has used the DiffServ class settings shown in table 7.6. Two versions of the MPEG-1 video "*Terminator II*" have been transmitted, both with a maximum transfer delay of 400ms: The first one's remapping intervals have been calculated without weighting, the second one's

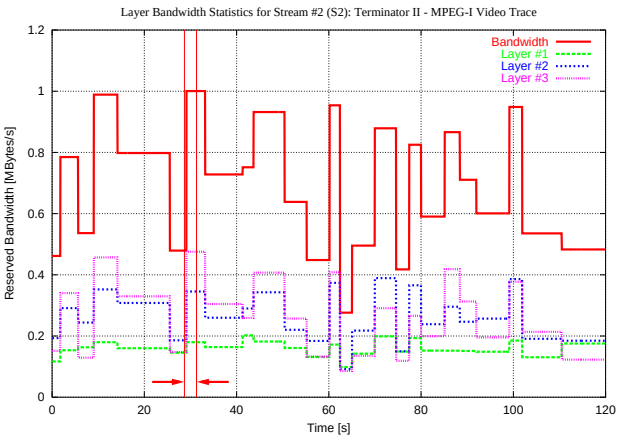


Figure 8.5: Unweighted remapping intervals

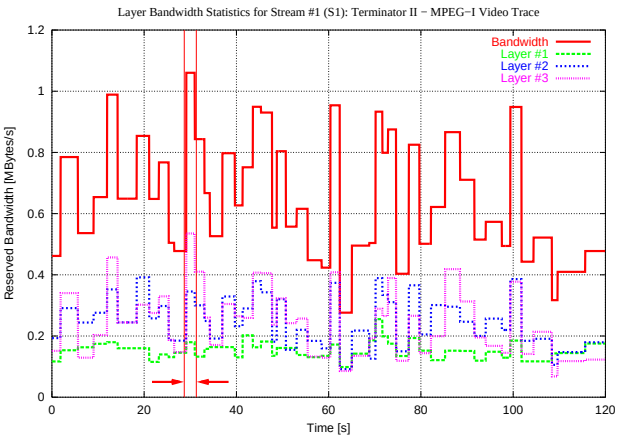


Figure 8.6: Weighted remapping intervals: 4/3/2

using the 4/3/2-weighting as explained in section 7.1.4. Figure 8.5 shows the unweighted stream's reserved bandwidth for the simulation time of 120 seconds - total and for each layer. The weighted version's results can be found in figure 8.6.

As a comparision between these two figures shows, the weighted case contains much more peaks for the reason of more bandwidth remappings. Note especially the two large blocks from about 8s to 25s and 30s to 50s in the unweighted case. These blocks are split into several peaks in the weighted case, resulting in a lower total bandwidth requirement. It should be denoted here, that some peaks in the weighted case are higher than in the unweighted one, e.g. at 29s to 31s (marked by two arrows in both figures): About 1.1 MBytes/s instead of 1.0 MBytes/s. The reason for this is that the remapping intervals of the unweighted case are independent of the weighted case's ones. Therefore, the current traffic description for a certain time stamp (e.g. 29s) in both simulations may differ. In this example (29s to 31s), the layer is mapped to AF21 in the unweighted case, leading to a lower bandwidth requirement due to buffering. For the weighted case, the different traffic description results in a transport via AF11, where the buffering gain is lower and therefore the bandwidth requirement is higher.

Finally, the resulting total bandwidth and cost requirements for the playtime of 120s are:

Trace	Bandwidth [Bytes]	Cost [Bytes*$]	Avg. cost factor [$\frac{\$}{\text{Byte}*\text{s}}$]
Unweighted	86,051,278	260,921,343	3.03216
Weighted	80,141,161	235,831,582	2.94270

The weighing reduced the total cost by 25,089,761 Bytes*$, this is a reduction of 9.61% for this example. The total reserved bandwidth reduced by 5,910,117 bytes, leading to a reduction of 6.87%. The average cost factor is simply the quotient $\frac{\text{Total cost}}{\text{Total bandwidth}}$.

Results

As it is shown for the "*Terminator II*" video example, the layer-weighted calculation of remapping intervals may result in a significant cost gain. Changes of more expensive layers will affect the calculation's cost more than cheaper ones. This implies earlier bandwidth remappings and therefore a shorter usage of expensive bandwidth. Now, the total gain has to be examined for a wide range of maximum transfer delay limits and the complete set of traces. This is done in section 8.1.3.

8.1.3 The Buffering, Layering and Weighting Comparision

As mentioned above, this simulation will compare the gain of buffering and weighted remapping interval calculation using all traces mentioned in section 7.1 and a wide range of buffer delays. Further, the effects of layered (each transport layer may use its own DiffServ class) and unlayered transmission (all transport layers must use the same DiffServ class) are examined.

Introduction

Again, the DiffServ class settings of table 7.6 have been used to enable usage of the cost-optimal class. As explained in section 7.3, this is realistic because of the bandwidth broker. Due to the loss rate requirements of table 7.5, BE is only usable for the MPEG-2 enhancement layers. The transfer delay limit ranges from 100ms to 1000ms in steps of 50ms. For this simulation, utilization is always 100%, examinations of the scalability behavior can be found in section 8.1.4. For each trace, four simulations have been computed, each having a duration 800 seconds:

1. Weighted remapping interval calculation and layered transmission,
2. weighted remapping interval calculation but only unlayered transmission,
3. unweighted remapping interval calculation and layered transmission,
4. unweighted remapping interval calculation and unlayered transmission.

Since MP3 traces contain only one layer, simulation #2 to #4 are senseless for this type and have therefore been skipped. To fill the playtime of 800 seconds for shorter medias, they are auto-repeated. The results for each trace can be found in appendix B.

MPEG-1

For MPEG-1, figure 8.7 (left side) shows the average cost for simulation #1 to simulation #4. The cost increment in percent, compared to simulation #1 (weighted and layered) can be found on the right side. Analogous plots are presented in figure 8.8 for the average bandwidth and figure 8.9 for the average cost factor ($\frac{\text{Total Cost}}{\text{Total Bandwidth}}$).

First of all, a large buffering gain is noticable. Even for a very small maximum transfer delay of 200ms, the cost and bandwidth requirements nearly halve. For maximum transfer delays above 600ms, they even reduce by up to about 85% for cost and 75% for bandwidth.

As shown on the left side of figure 8.9, the average cost factor for all simulations is almost 4.0 for transfer delays up to 250ms. This means, that most layers are transported via EF (cost factor 4.0), due to its high buffering gain. Therefore, the cost increment for unlayered transmission is very low (0 for transfer delays less than 200ms). At transfer delay limits of 200ms and more, other classes than EF become usable and the average cost factor of the layered transmissions decreases to less than 3.0 for 300ms and more. This results in a cost increment for the unlayered transmission to 15% and more (see right side of figure 8.7).

The usable classes for layer #1 are only EF and AF41, for layer #2 EF, AF41, AF31 and AF21 and for layer #3 EF and all AF classes but not BE. Due to the different loss rate requirements as explained in section 8.1.1, the cost increment for unlayered transmission is highest for transfer delays above 600ms - more than 20% (for weighted remapping intervals)! Here, cheap classes (AF11 for layer #3 and AF21 for layer #2) could be used. But since layer #1 requires at least the loss rate of AF41, the same expensive class as for layer #1 has to be used. Due to the already high transfer delay and therefore high possible buffer delay, the additional possible buffer delay introduced by the faster class results in no gain anymore. This can be viewed on the right side of figure 8.8: For transfer

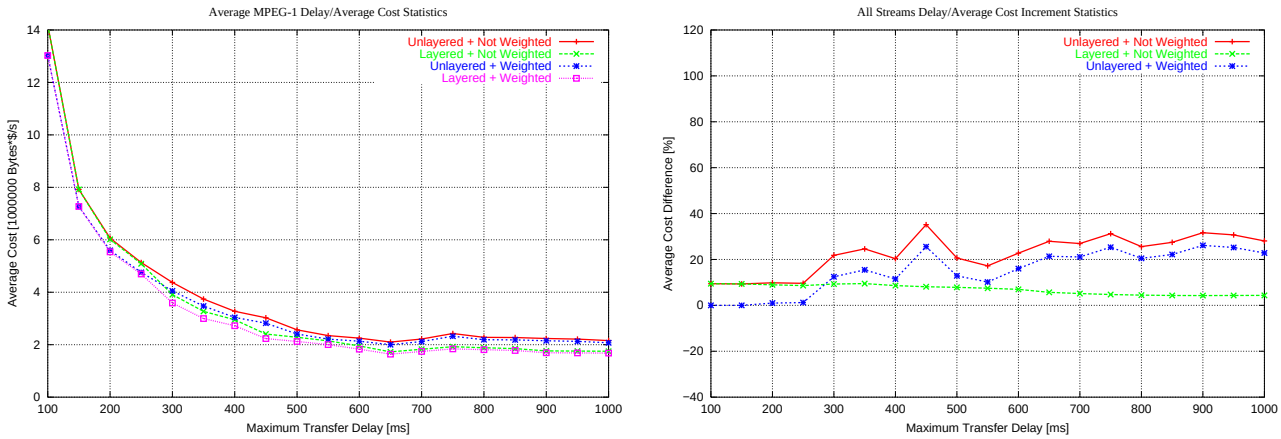


Figure 8.7: MPEG-1 average cost/delay comparision

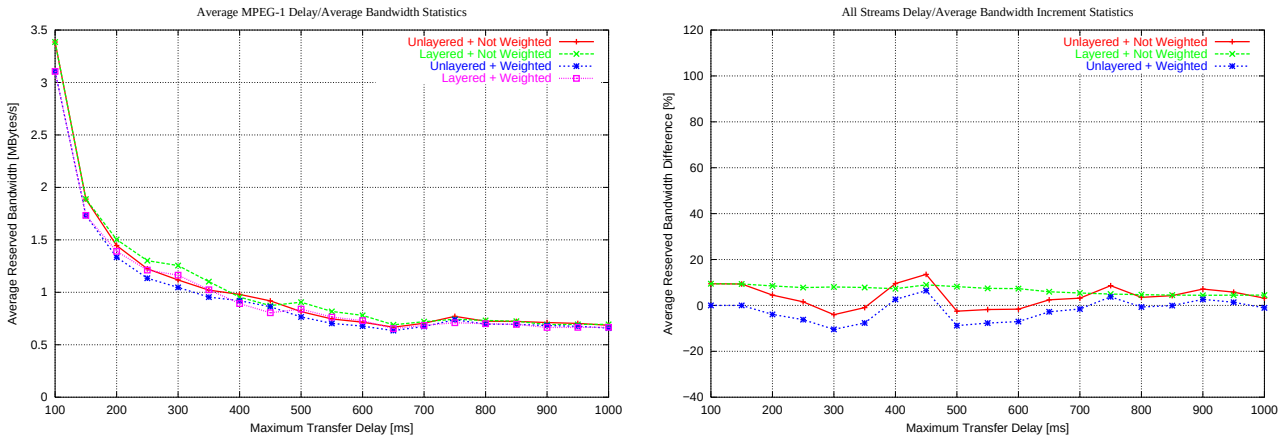


Figure 8.8: MPEG-1 average bandwidth/delay comparision

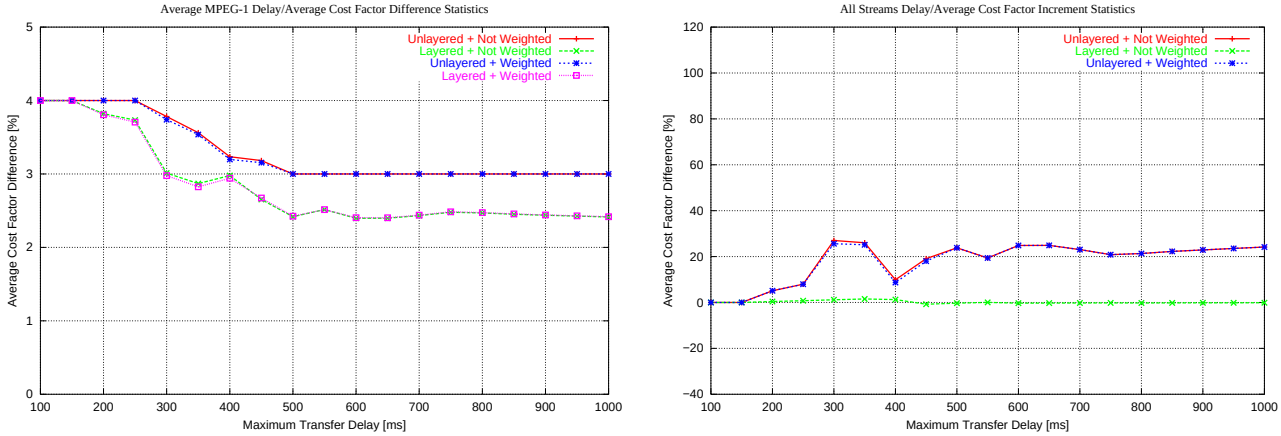


Figure 8.9: MPEG-1 average cost factor/delay comparision

delays below 600ms, the unlayered transmissions' bandwidth is usually less than the layered case's one. Here, the faster class's higher buffering gain is able to partially compensate the cost increment. For example at 300ms transfer delay, the unlayered transmissions' bandwidth decrement is about 10% (for weighted remapping intervals). But at the same time, the cost factor (see right side of figure 8.9) increases by about 25%.

As it is shown on the right side of figure 8.7, the cost increment for using unweighted remapping intervals is about 10% for transfer delays up to 350ms and steadily decreases to about 5% for a transfer delay of 1000ms. The effect on the cost is therefore highest, when expensive classes are used. This is also the expected behavior.

MPEG-2

Corresponding to the results of MPEG-1, the results for MPEG-2 can be found in figure 8.10 (average cost), figure 8.11 (average bandwidth) and figure 8.12 (average cost factor). Again, the left side shows the absolute values and the right one the increments for simulation #2 to #4 in percent.

Due to the DiffServ class settings of table 7.6, the layers' loss rate requirements (see table 7.5) allow usage of BE for the MPEG-2 enhancement layers (transport layer #4 to #9). Again, a large buffering gain is noticable: The cost requirement reduced by about 92% and the bandwidth requirement by about 78% for a transfer delay of 1000ms. Further, layered transmission results in a huge cost gain here, as expected: For a maximum transfer delay of 1000ms, the unlayered transmission increments the cost by more than 100% (see right side of figure 8.10). Due to the limit of at least AF41 for layer #1, all 9 layers have to use at least this class in the unlayered case. Therefore, no cost reduction is possible.

Especially for transfer delays of 600ms and more, a huge cost increment compared to layered transmission can be viewed, introduced by the availability of BE (600ms transfer delay, see table 7.6). The average cost factor remains at 3.0 (AF41) for 500ms and more transfer delay (see left side of figure 8.12). The layered transmission manages to reduce this value to about 1.2. This is a cost factor increment of 250% for the unlayered transmission (see right side of figure 8.12).

As shown on the right side of figure 8.11, the required bandwidth for unlayered transmission is usually much lower, in two cases by up to about 30%. Again, as explained for MPEG-1, the higher bandwidth cost is partially compensated by a higher buffering gain. But of course, this gain is far too low to completely compensate the low cost of BE.

As explained for MPEG-2 in section 7.1, the enhancement layers' fraction of the streams is about 83%. Since the weights mainly affect the expensive MPEG-2 base layer (transport layer #1 to #3 - a size fraction of about 17% only), the visible effect of the weighted remapping intervals is much lower compared to MPEG-1: Only about 1.5% to 0.5%.

H.263

Now, corresponding to the results of MPEG-1 and MPEG-2, the results for H.263 can be found in figure 8.13 (average cost), figure 8.14 (average bandwidth) and figure 8.15 (average cost factor). Futher, the left side again shows the absolute values and the right one the increments for simulation #2 to #4 in percent.

This simulation is comparable to MPEG-1, except that another frame type, PB-frames, is sent in an additional layer. The PB-frames' QoS requirements are set to the same values as P-frames (see table 7.5). None of the used traces has got B-frames, therefore AF11 is not usable for H.263 due to the loss rate settings of table 7.6.

The left side of figure 8.13 shows, that the buffering gain of the H.263 traces is not so high as for the MPEG-1 traces: Only about 65% for cost and 40% for bandwidth at 500ms transfer delay in the layered and weighted case. Since H.263 is mainly used for video conferences, it is useful to have only

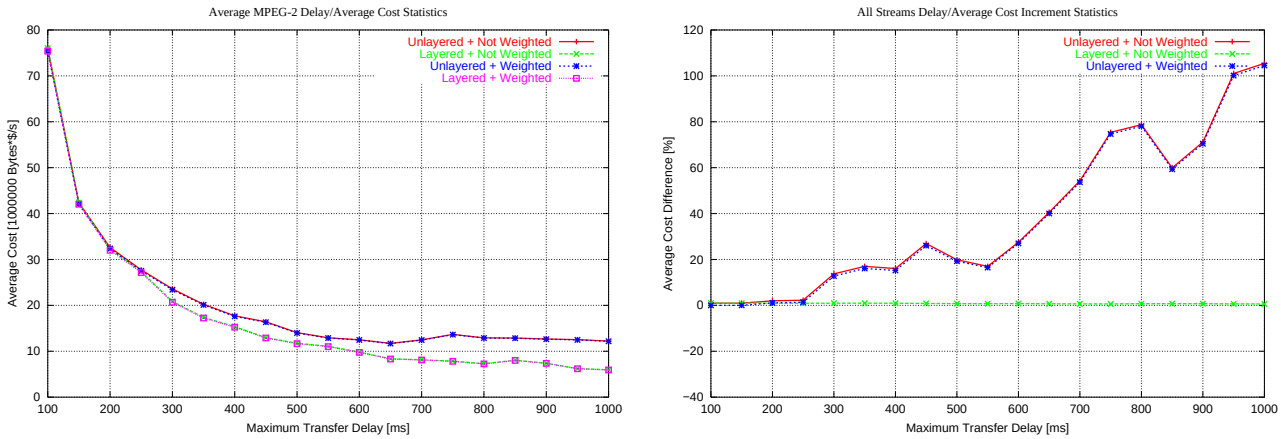


Figure 8.10: MPEG-2 average cost/delay comparision

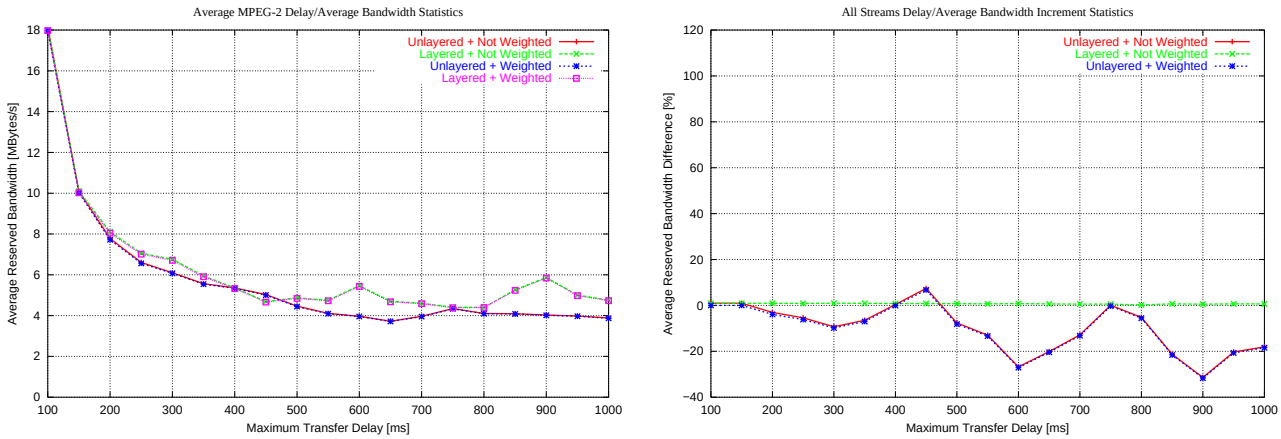


Figure 8.11: MPEG-2 average bandwidth/delay comparision

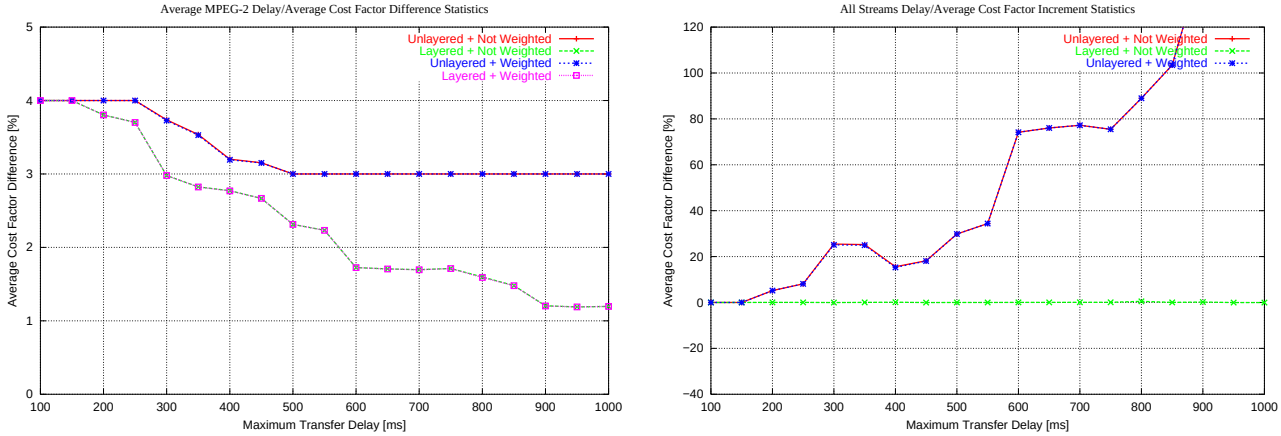


Figure 8.12: MPEG-2 average cost factor/delay comparision

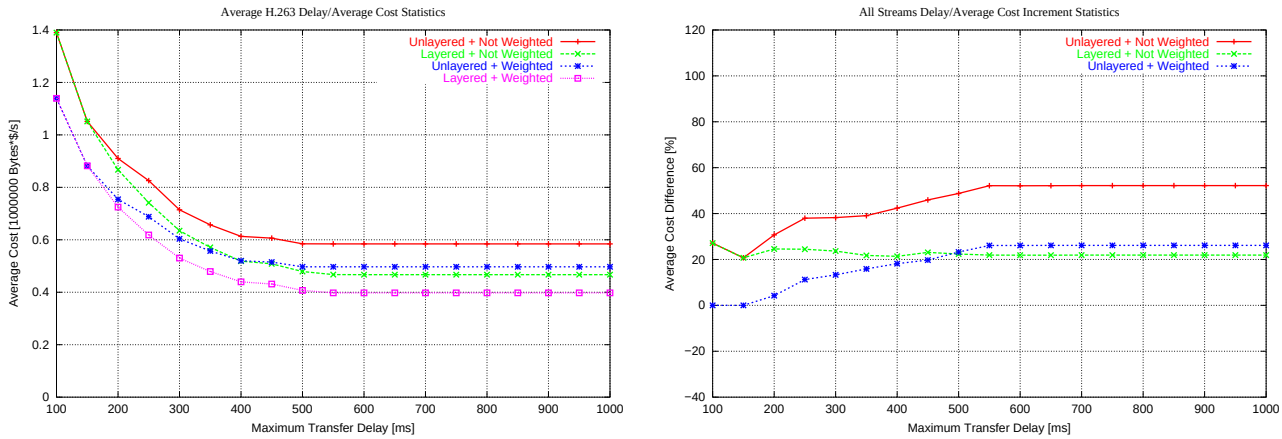


Figure 8.13: H.263 average cost/delay comparision

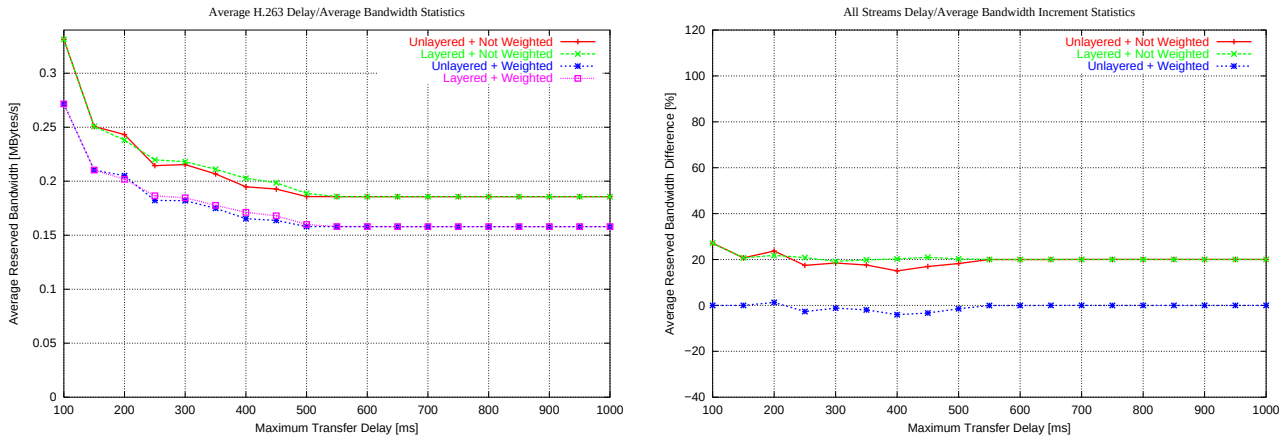


Figure 8.14: H.263 average bandwidth/delay comparision

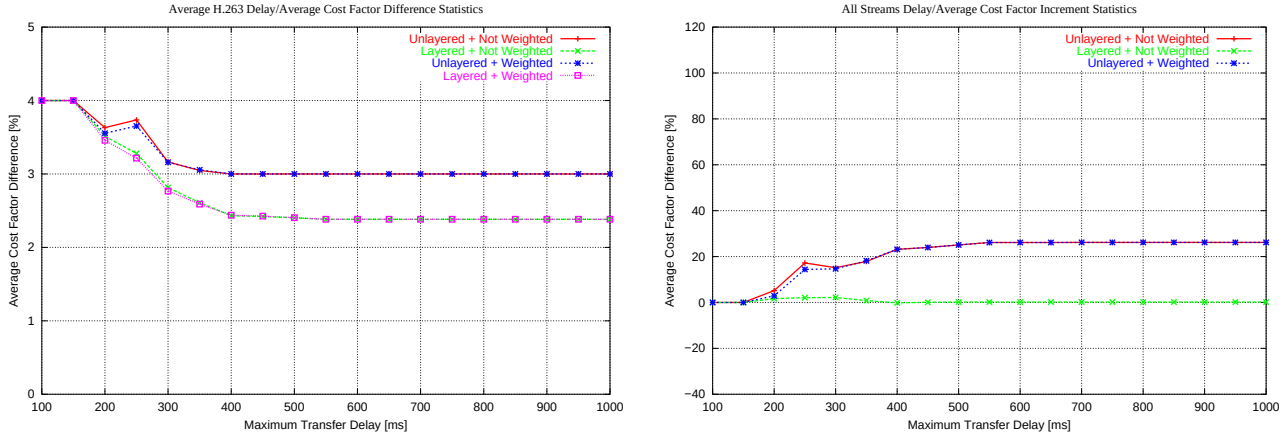


Figure 8.15: H.263 average cost factor/delay comparision

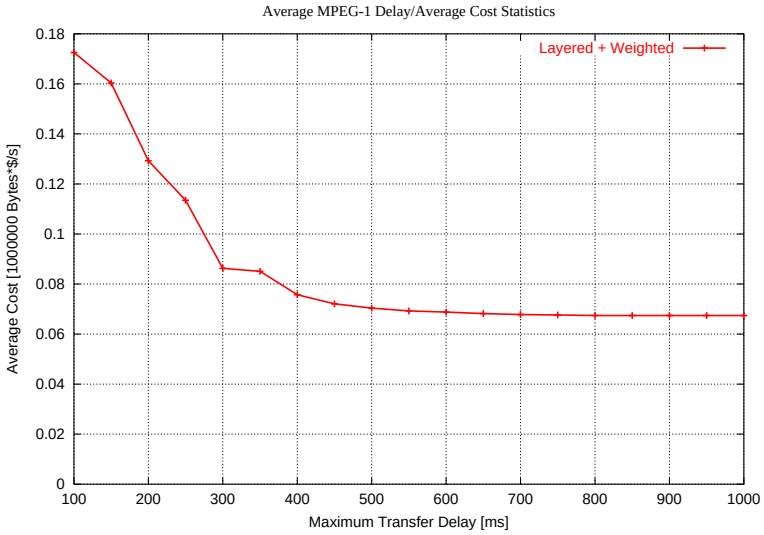


Figure 8.16: MP3 average cost/delay comparision

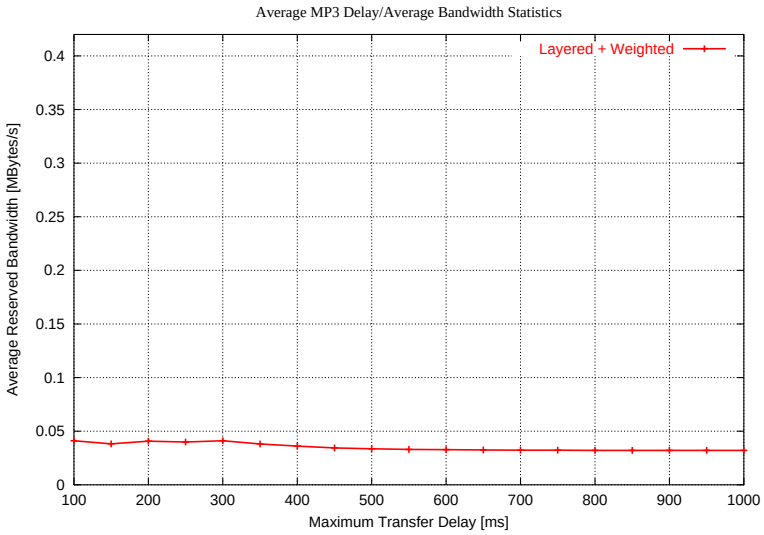


Figure 8.17: MP3 average bandwidth/delay comparision

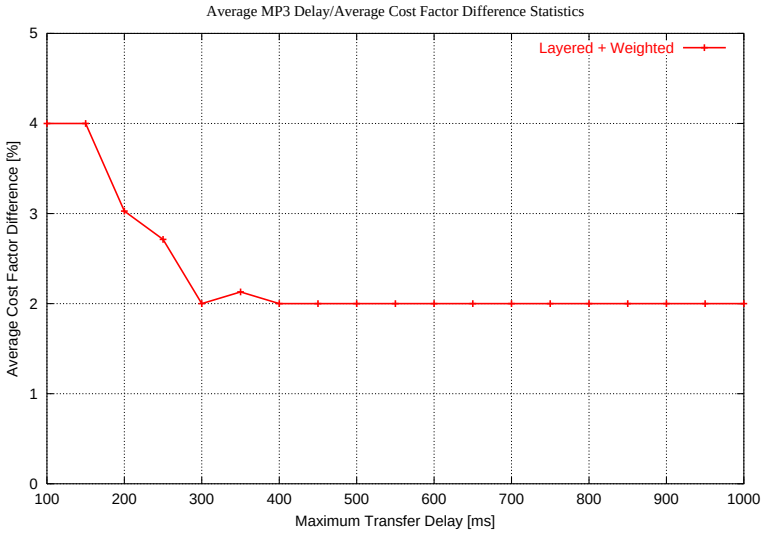


Figure 8.18: MP3 average cost factor/delay comparision

low bandwidth variances - even for variable bitrates. This is achieved using PB-frames and usually no I-frames (see section 2.6.3). The generated streams are therefore smoother than for MPEG-1.

As it is shown on the right side of figure 8.13, the cost increment for unlayered transmission is much higher than for MPEG-1: More than 20%. The unweighted remapping intervals increase the cost almost steadily up to about 25%, similar to MPEG-1. The total cost increment for the unweighted and unlayered case is therefore up to more than 50% - compared to up to about 30% for MPEG-1.

MP3

Finally, MP3 has to be examined. Since there is only one layer, weighted and unweighted remapping interval calculation and layered and unlayered transmission are equal here. The results can be found in figure 8.16 (average cost), figure 8.17 (average bandwidth) and figure 8.18 (average cost factor).

In figure 8.16, a large buffering gain is noticable. At a maximum transfer delay of 300ms, the cost requirement nearly halves. For maximum transfer delays above 600ms, it even reduces by up to about 64%. But the bandwidth requirement for transfer delays up to 300ms is varying. For 300ms transfer delay, it is as high as for 100ms (see figure 8.17). The reason for this is the decreased cost factor: It is most efficient to use more but cheaper bandwidth than less but expensive. As it is shown in figure 8.18, it is 4.0 (EF) for 100ms transfer delay and it decreases to 2.0 (AF11) at a transfer delay of 300ms. Therefore, the total cost keeps always decreasing.

Results

As it is shown, there is a huge gain using buffering combined with transport via the most cost-efficient DiffServ class for all examined media types. Additional gains can be achieved by layered transmission and the usage of layer-weighting for the remapping interval calculations.

8.1.4 The Scalability Comparision

Now, the effects of scalability to buffering and cost have to be examined for each media type. Therefore, simulation #1 of section 8.1.3 (weighted remapping intervals and layered transmission) has been repeated for maximum utilization limits of 25%, 50%, 75% and 100%. Again, the same DiffServ class settings as described in table 7.6 have been used.

For the video media types (MPEG-1/2 and H.263), the results of each trace are averaged, grouped by frame rate/frame size requirements as described in section 7.1: High/High (action, sports, music), Low/High (comedy, movie, news), High/Low (cartoon) and Low/Low (talk). Since the used MP3 traces are not grouped, the complete MP3 set is averaged. The full results of this simulation for each trace can be found in appendix C.

MPEG-1

The results for MPEG-1 can be found in figure 8.19 (100% utilization), figure 8.20 (75% utilization), figure 8.21 (50% utilization), and figure 8.22 (25% utilization). As it is shown in figure 8.19 for 100% utilization, there is already a cost difference of about 15% to 30% between the curves for high and low frame rate requirement. As expected, videos like talk shows have got a lower bandwidth requirement than e.g. action and sports. Since the utilization is 100% here, no significant difference between the frame size requirements can be noticed. The buffering gain for all types is almost equal, about 85% for 600ms transfer delay.

The utilization reduction to a maximum of 75% in figure 8.20 already shows a difference of about 15% to 20% between high and low frame size requirement for a high frame rate requirement. As expected, the High/High curve has got a higher bandwidth requirement. But the buffering gain for both curves is nearly unchanged compared to the 100% case: Still about 85%. The reason for this is,

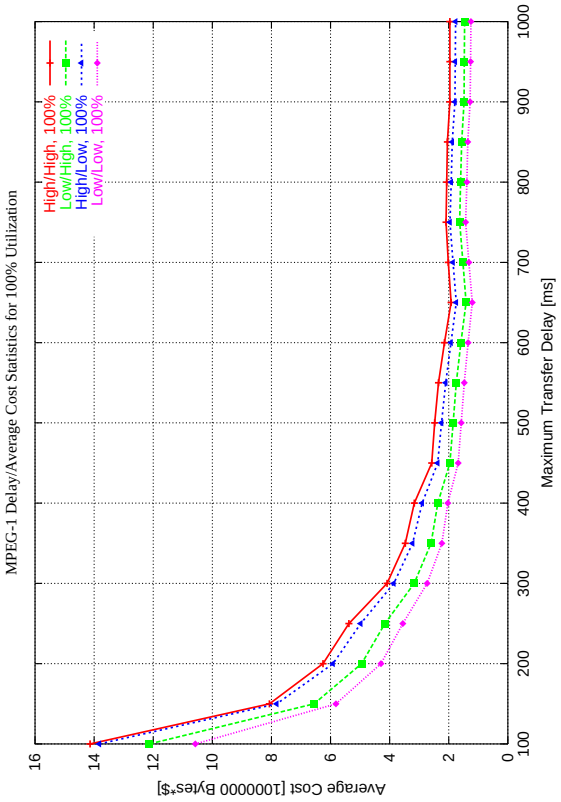


Figure 8.19: MPEG-1 cost for 100% utilization

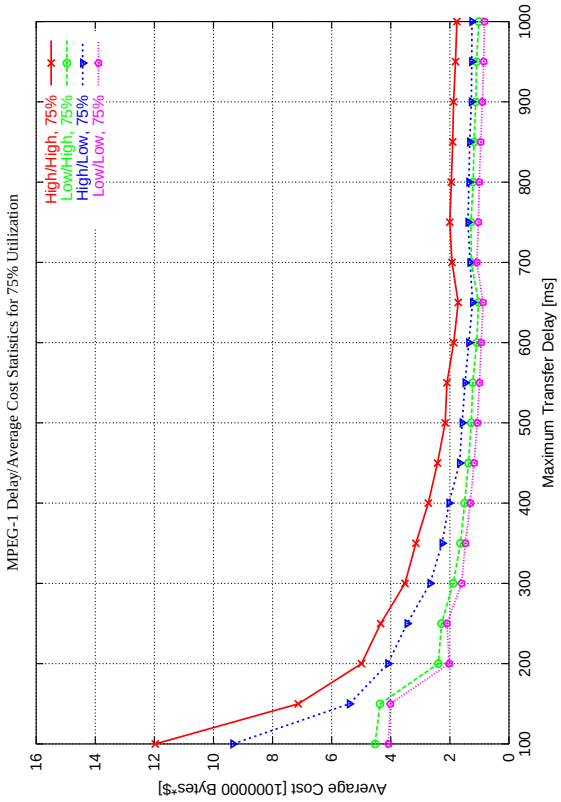


Figure 8.20: MPEG-1 cost for 75% utilization

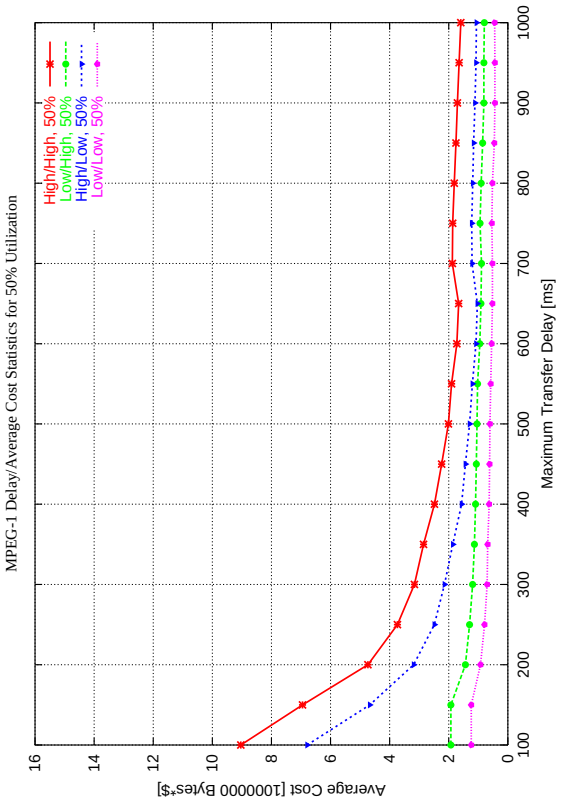


Figure 8.21: MPEG-1 cost for 50% utilization

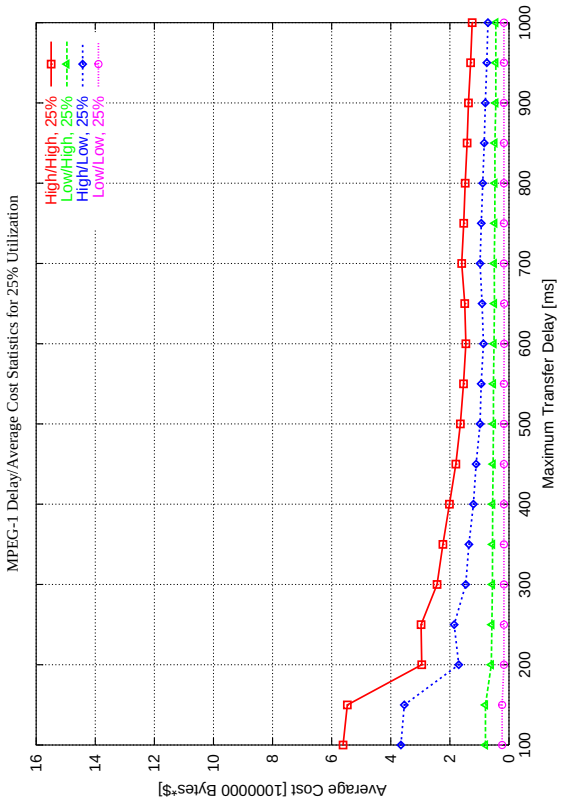


Figure 8.22: MPEG-1 cost for 25% utilization

that both curves have got a high frame rate requirement. Therefore, the frame rate for 75% utilization limit is large enough to provide a huge buffering gain. In contrast, the low frame rate requirement's curves have got a gain of about 60% for 600ms transfer delay only. Due to the lower frame rate, more B-frames and P-frames are missing. Therefore, the I- and P-frame 'gaps' (see table 4.1 for comparision) are smaller, resulting in lower buffering ability. The continuation of these buffering results can be noticed for 50% utilization (see figure 8.21) by 80% buffering gain for high and 50% for low frame rate requirement at 600ms transfer delay. Finally for 25% utilization (see figure 8.22), the gain for a high frame rate requirement is still about 70% (600ms transfer delay). But for a low requirement, almost no gain is noticable anymore.

As expected, the curves of the four requirement combinations steadily separate. While for 100% utilization the low and high frame size requirements are mainly overlapping, they separate with decreasing utilization. Finally, the results show, that low frame rate/low frame size is cheapest and high frame rate/high frame size is most expensive. The total scalability gain is about 60% for High/High, 70% for High/Low, 85% for Low/High and 95% for Low/Low frame rate/frame size requirements at 25% utilization limit, compared to the 100% case.

MPEG-2

The results for MPEG-2 can be found in figure 8.23 (100% utilization), figure 8.24 (75% utilization), figure 8.25 (50% utilization), and figure 8.26 (25% utilization).

The buffering results are comparable to MPEG-1. Therefore, it is not necessary to explain them again. As expected, there is a huge scalability gain: About 95% for High/High and High/Low and about 99,5% for Low/High and Low/Low frame rate/frame size requirements. The reasons for this is, that the MPEG-2 enhancement layers, having a fraction of about 83% of the total size (see section 7.1), may simply be skipped. This results in an excellent scalability gain.

H.263

Now, the H.263 traces are examined: The results are shown in figure 8.27 (100% utilization), figure 8.28 (75% utilization), figure 8.29 (50% utilization), and figure 8.30 (25% utilization). It is important to denote here, that the video "*Sendung mit der Maus IP*" is not included in the High/Low frame rate/frame size requirements plots, due to its very unusual constant GoP "IPIP . . . ". The results for this trace can be found in appendix C.

As expected, the buffering results look similar to the MPEG-1 results and therefore need no further explaination. As shown in section 8.1.3, the buffering gain for H.263 is lower than for MPEG-1. This can also be found for the 75%, 50% and 25% utilization limit results. The total scalability is about 75% for High/High, 80% for High/Low, 90% for Low/High and slightly more than 90% for Low/Low frame rate/frame size requirements at 25% utilization.

MP3

Finally, figure 8.31 shows the results for MP3 at 100%, 75%, 50% and 25% utilization limit. As mentioned in section 7.2, all traces have got the same utility function. Due to the reasons explained there, 95% utilization is given to 40% bandwidth and 50% utilization to 10%. This is reflected by figure 8.31: Compared to the 100% utilization curve, the cost decreases by about 60% for 75% utilization. The cost curves for 75%, 50% and 25% are tight together due to the low bandwidth differences of these utilizations. As expected, the buffering gain is independent of the utilization limit: About 60% for all limits. Since the frame rate of MP3 remains constant (38 frames/s), the buffering gain is not affected by frame rate scalability. And frame size scalability does not affect the buffering. Therefore, there is no reduction of its gain for reduced utilization.

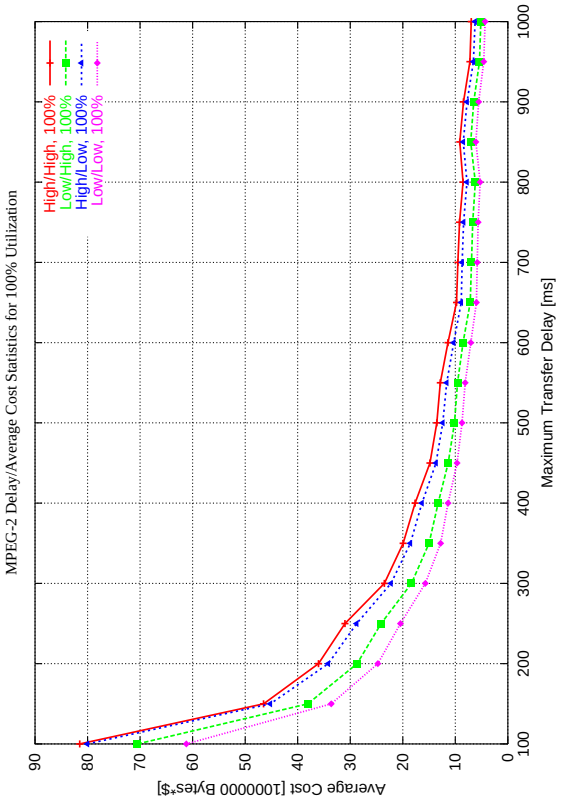


Figure 8.23: MPEG-2 cost for 100% utilization

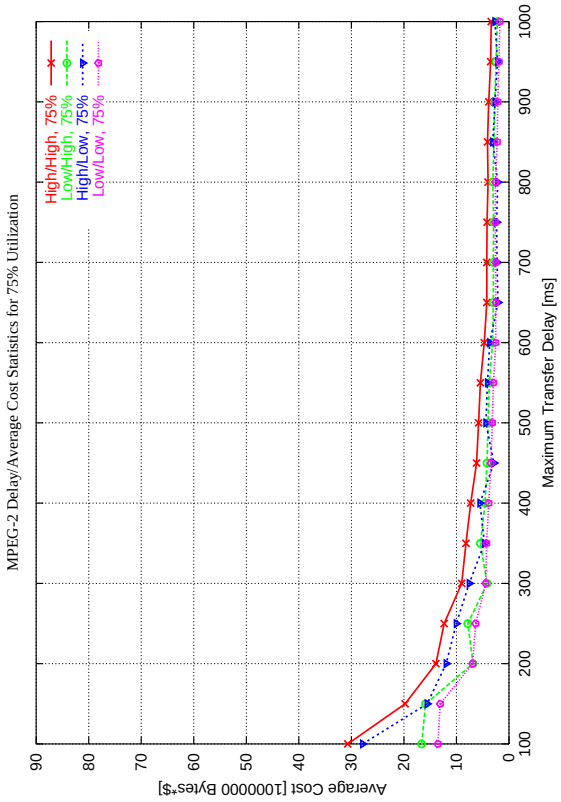


Figure 8.24: MPEG-2 cost for 75% utilization

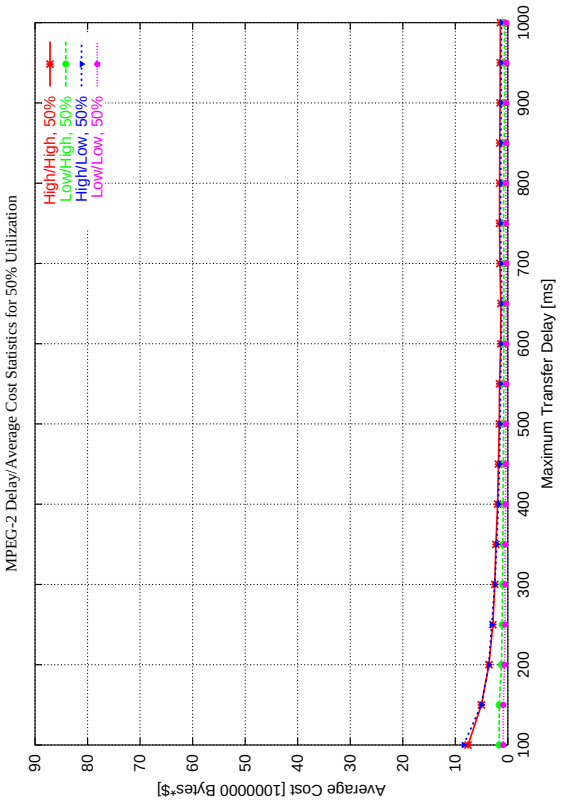


Figure 8.25: MPEG-2 cost for 50% utilization

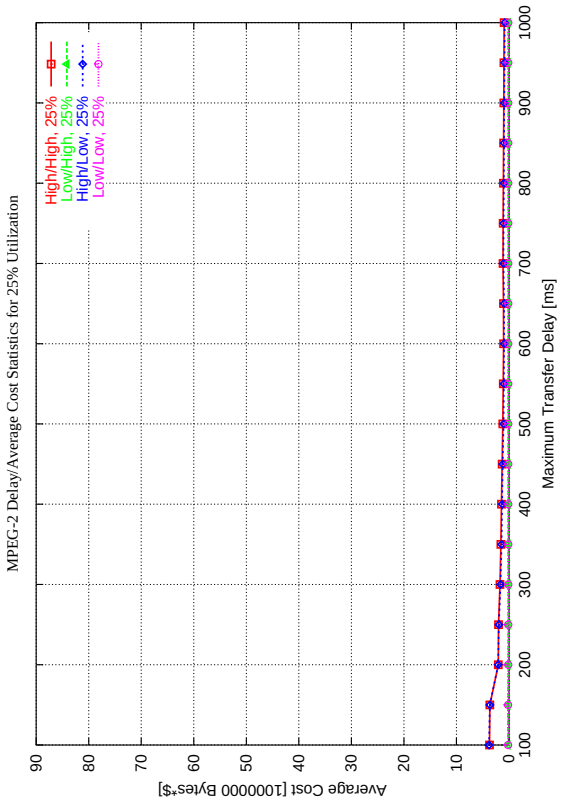


Figure 8.26: MPEG-2 cost for 25% utilization

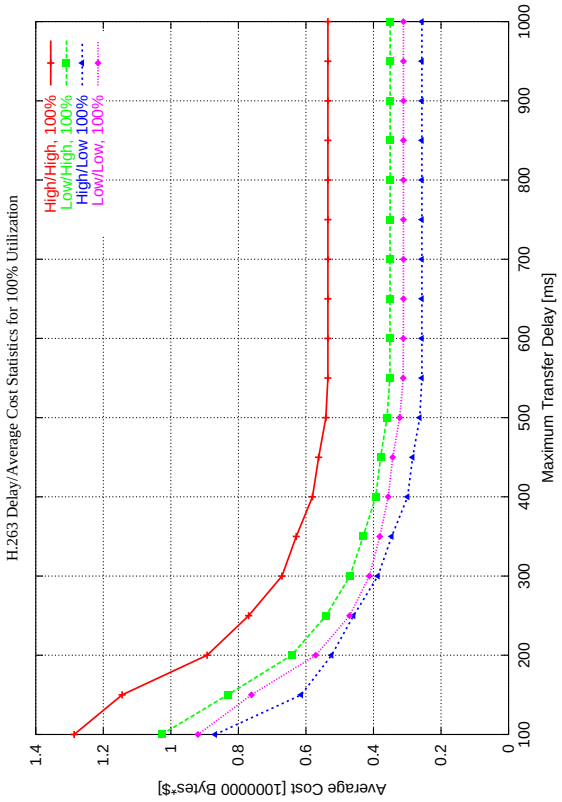


Figure 8.27: H.263 cost for 100% utilization

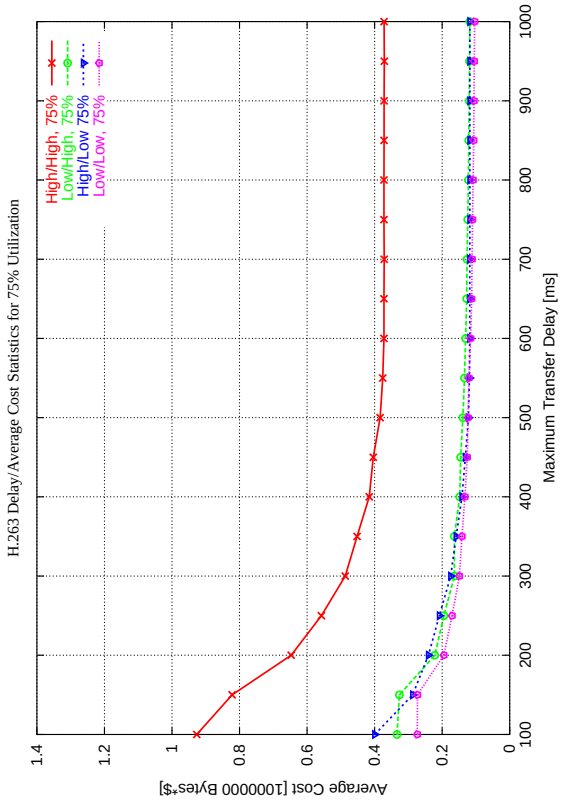


Figure 8.28: H.263 cost for 75% utilization

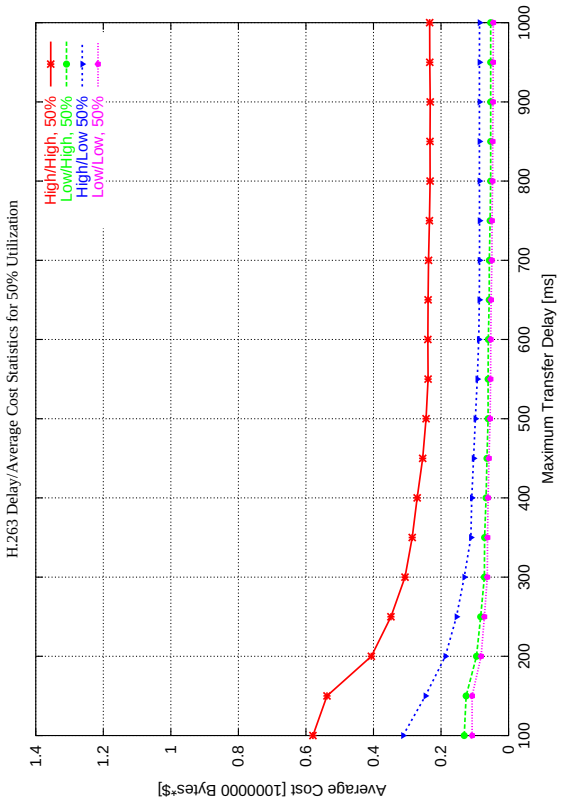


Figure 8.29: H.263 cost for 50% utilization

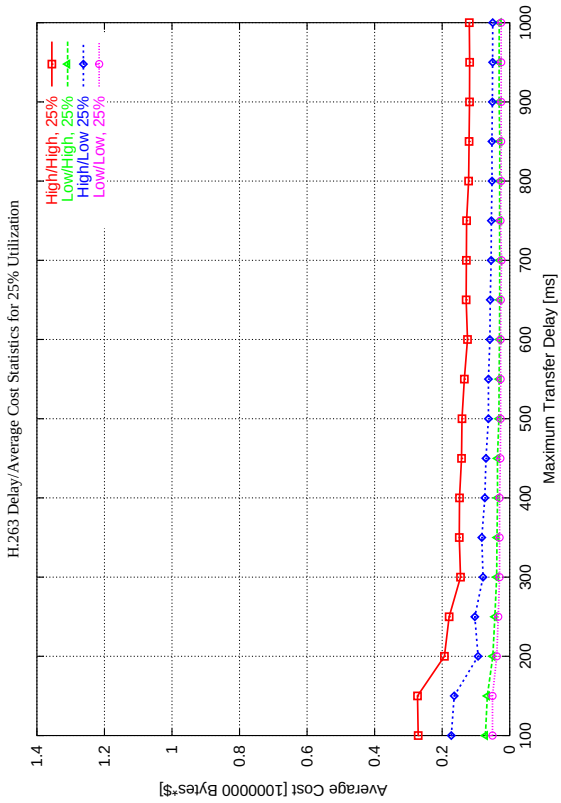


Figure 8.30: H.263 cost for 25% utilization

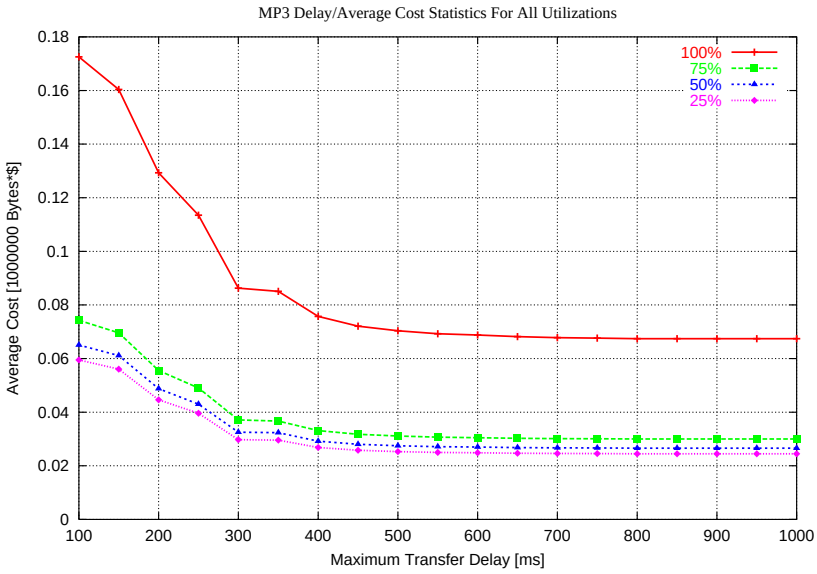


Figure 8.31: MP3 cost for different utilizations

Class	Bandwidth	Cost factor	Variability	Delay	Loss rate	Jitter
BE	20 MBytes/s	1.0	25 %	600 ms	10.00 %	200 ms
AF11	5 MBytes/s	2.0	10 %	300 ms	0.05 %	100 ms
AF21	5 MBytes/s	2.5	10 %	200 ms	0.02 %	50 ms
EF	5 MBytes/s	4.0	5 %	100 ms	0.01 %	10 ms

Table 8.1: The DiffServ class settings of the network quality example

Results

As expected, the cost requirement of each video stream depends on its genre. Streams having high requirements need more bandwidth than streams having lower ones. As it is shown, streams of higher frame rate have got a smaller buffering gain due to the greater amount of 'zero-sized' frames. For MP3, high scalability gains can be noticed for utilizations below 75%, due to its utilization of 95% for only 40% bandwidth. As expected, buffering is not affected by lower utilizations, since the frame rate for MP3 remains constant.

8.2 Bandwidth and QoS Management Functionality

8.2.1 A Network Quality Change Example

This simulation demonstrates the bandwidth manager's behavior on changes of the network's quality of service and the SLA's bandwidth settings by a comprehensive example. Only one MPEG-1 video has been used: "*Formula One Race*", at a maximum transfer delay of 375ms for a duration of 180 seconds. The used DiffServ class settings are shown in table 8.1. Scalability is not used in this simulation, the utilization will always be 100%. First, the simulation has been computed without any changes. Next, some quality changes have been applied:

- From 30s to 60s:
  The available EF bandwidth has been set to 0.

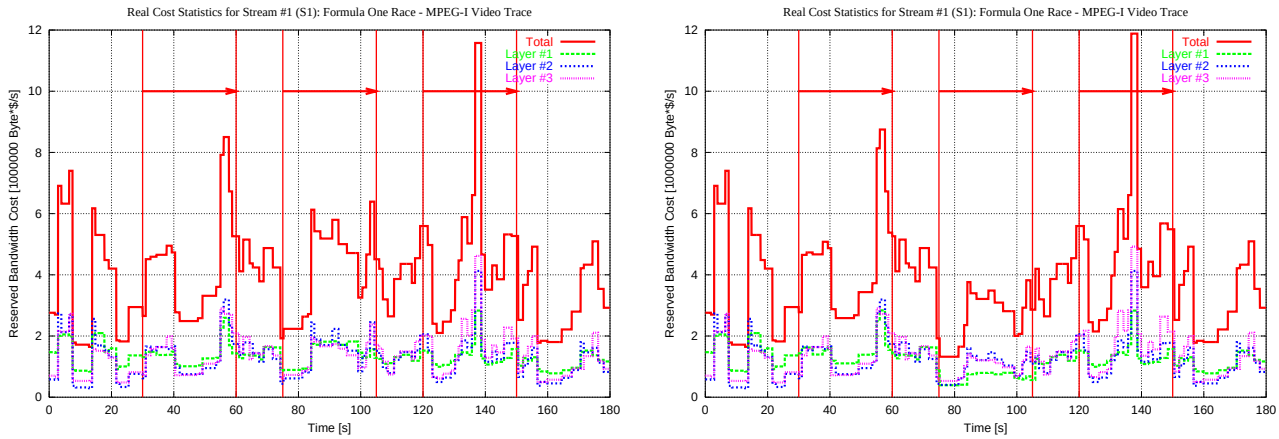


Figure 8.32: Total cost comparision without (left) and with (right) network quality changes

- From 75s to 105s:
  The transfer delay of AF11 has been improved from 300ms to 75ms.
- From 120s to 150s:
  The loss rate of AF21 has been changed from 0.02% to 10%.

Figure 8.32 shows the cost (total and for each layer) of simulation #1 (left side) and simulation #2 (right side). The comparision of the layer to DiffServ class mappings for layer #1 (I-frames) can be found in figure 8.33. Here and in the following figures, each DiffServ class's bandwidth reservation for the corresponding layer is shown. The quality change intervals described above are marked by lines and arrows. Again, the result of simulation #1 can be found on the left side and simulation #2's result on the right side. As expected, AF21 bandwidth is used instead of EF from 30s to 60s. Since its transfer delay is higher, the buffer delay is lower. Therefore, the total bandwidth requirement is significantly increased: A large peak can be found at about 55s (see right side of figure 8.33), resulting in an about nearly doubled bandwidth requirement. But due to the lower cost factor of AF21 (2.5) compared to EF (4.0), the cost is only about 10% higher (compare layer #1 in figure 8.32). The transfer delay improvement of AF11 from 300ms to 75ms (see 75s to 105s in figure 8.33) causes layer #1 to be mapped from EF to AF11, since its cost factor is only 2.0 compared to 4.0 of EF. Due to its small transfer delay of 75ms, the buffer delay can be increased, resulting in an about 10% lower bandwidth requirement. But due to the quite low cost of this class, there is a huge cost reduction of about 60% (compare layer #1 in figure 8.32). An effect of the increased loss rate of AF21 from 120s to 150s can only be viewed for the small interval from 140s to 142s, where layer #1 uses AF21 in simulation #1 (see left side of figure 8.33). Instead of AF21, EF is used here (see right side of figure 8.33), resulting in a slightly lower bandwidth requirement due to a higher buffer delay at a slightly higher cost (see layer #1 in figure 8.32).

In figure 8.34, the results for layer #2 (P-frames) are presented: This layer is not affected by the removal of EF bandwidth (30s to 60s) and the increased AF21 loss rate (120s to 150s), since it uses AF11 most of the time. The fast AF11 bandwidth from 75s to 105s results in a significant bandwidth and cost gain of about 25% (compare layer #2 in figure 8.32).

Finally, figure 8.35 presents the mappings of layer #3 (I-frames). Since this layer's usual class is AF21, there are no effects introduced by the removal of EF bandwidth from 30s to 60s. During acceleration of AF11 from 75s to 105s, AF11 is used instead of AF21, implying decreased bandwidth (since it is faster) and cost (since it is cheaper, too) requirements. The increased loss rate of AF21 makes this class unusable from 120s to 150s, resulting in usage of AF11 and a significantly higher bandwidth requirement of about 25% (see figure 8.35), due to the lower buffer delay. But because of

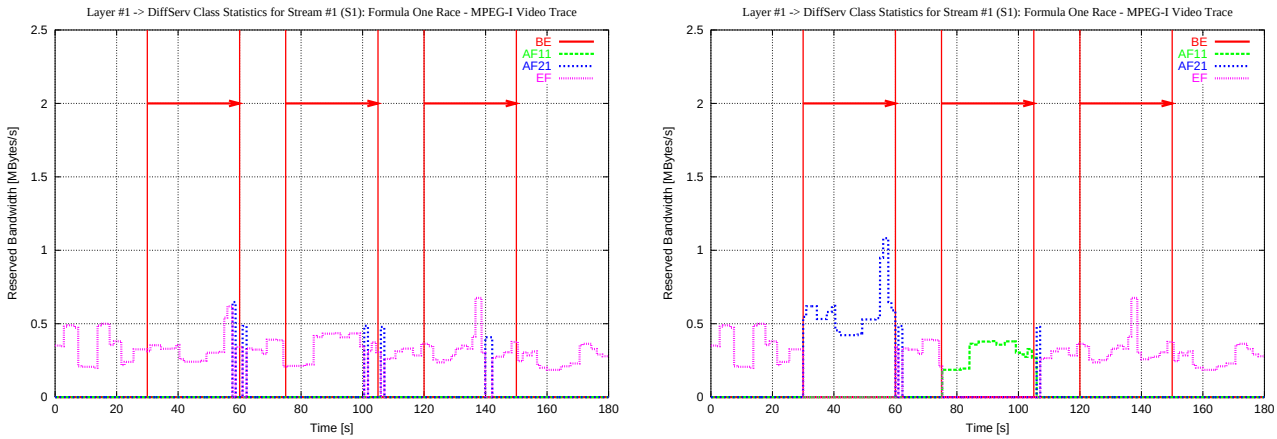


Figure 8.33: DiffServ class for layer #1 (I) for both network quality simulations

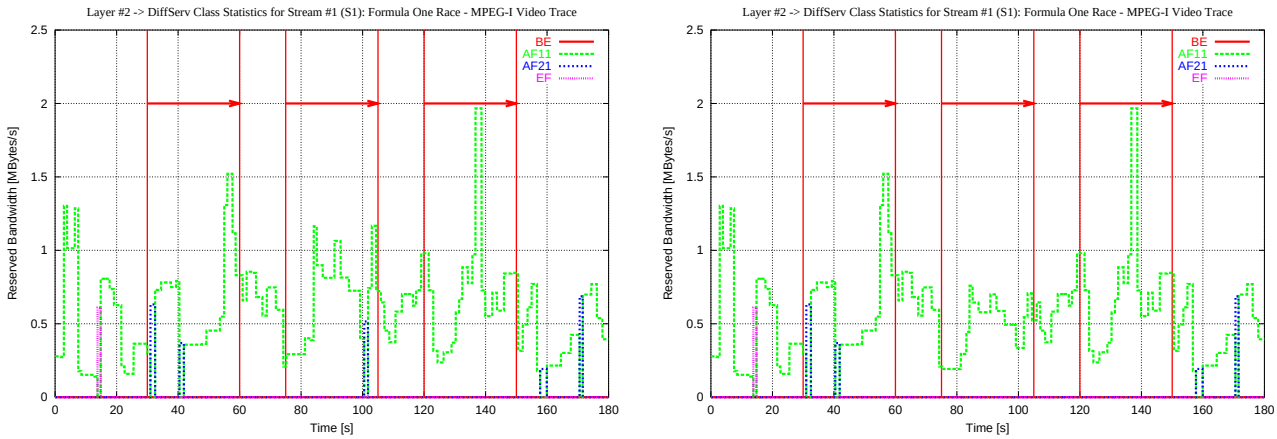


Figure 8.34: DiffServ class for layer #2 (P) for both network quality simulations

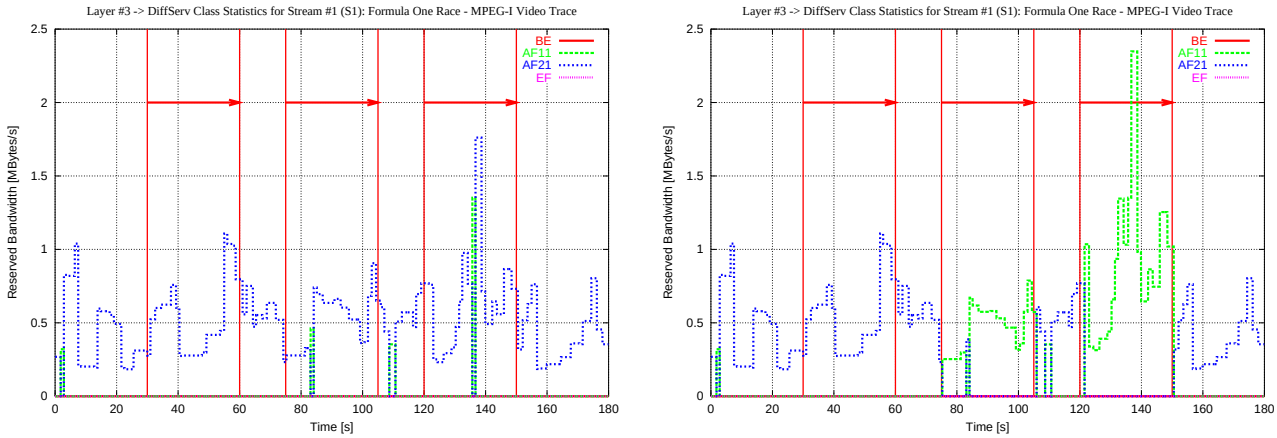


Figure 8.35: DiffServ class for layer #3 (B) for both network quality simulations

Class	Bandwidth	Cost factor	Variability	Delay	Loss rate	Jitter
BE	2 MBytes/s	1.0	25 %	600 ms	10.00 %	200 ms
AF11	1 MBytes/s	2.0	10 %	300 ms	2.00 %	100 ms
AF21	1 MBytes/s	2.35	10 %	260 ms	1.00 %	80 ms
AF31	0,75 MBytes/s	2.65	10 %	230 ms	0.75 %	65 ms
AF41	0,75 MBytes/s	3.0	10 %	200 ms	0.25 %	50 ms
EF	0.5 MBytes/s	4.0	5 %	100 ms	0.05 %	10 ms

Table 8.2: The DiffServ class settings of the priority example

Stream	Media	Type	Stream priority	Sim. #1	Sim. #2
Session #1, Priority 0					
#1	Talk	MPEG-1	0	96.2 %	74.6 %
#2	Go West	MP3	-100	96.6 %	51.8 %
#3	Born to be Wild	MP3	0	95.9 %	72.3 %
#4	Hellraiser	MP3	100	95.0 %	99.3 %
Session #2, Priority 100					
#1	Super Bowl	MPEG-1	0	71.1 %	91.3 %
#2	San Francisco	MP3	0	70.3 %	90.7 %
#3	Die glorreichen Sieben	MP3	0	69,6 %	90.9 %
#4	Seven Tears	MP3	0	68.6 %	89.6 %

Table 8.3: The session and stream priority example scenario inclusive resulting utilizations

AF11's lower cost factor (2.0 instead of 2.5 for AF21), the cost only slightly increases by about 2% (compare layer #3 in figure 8.32).

Results

The system's behavior on changes of the network quality is as expected: Layers are mapped to other DiffServ classes to keep satisfying their given QoS requirements. Further, the cost-optimization always uses the most cost-efficient class, reducing the system's total cost. As it is shown, it is tried to keep the additional cost as low as possible.

8.2.2 A Session and Stream Priority Example

The intention of this simulation is to show the system's behavior on different settings of session and stream priorities. Its scenario is shown in table 8.3 and consists of two sessions, each containing one MPEG-1 video and three MP3 audio streams. The simulation duration is 180s, the maximum transfer delay is 400ms for the videos and 100ms for the MP3 streams. The used DiffServ class settings are shown in table 8.2. For the fairness constants, the settings $\omega_{\text{SessionFairness}} = 0$ (session fairness, see section 6.3) and $\omega_{\text{Fairness}} = 1$ (stream fairness, see section 6.2.4) are used. Therefore, the resulting bandwidth distribution is utility-fair within sessions, but it maximizes the system utilization globally. These settings seem to be useful in most cases, since users get fair sharing within their sessions and the provider can serve as many customers as possible with a high-quality transmission.

First, the simulation has been computed using priority 0 for all streams and sessions. The resulting average utilization of each stream can be found in the "*Sim. #1*" row of table 8.3. The sports video

"*Super Bowl*" of session #2 has got a significantly higher bandwidth requirement, compared to the talk show video "*Talk*" of session #1 (see table A.1 for trace statistics). Therefore, the unfair bandwidth distribution for sessions results in about 96% average utilization for the first, but only about 70% for the second session. But within each session, the distribution is utility-fair. This is proven by the quite similar utilizations in table 8.3.

Now, the simulation has been repeated using the priority values given in table 8.3. The resulting average utilizations are shown in the "*Sim. #2*" row of the table. Due to session #2's priority of 100, the "*Super Bowl*" video has got an average utilization of 91.3% now. The MP3 streams of this session have also increased their average utilizations to about the same level. Session #1's bandwidth is decreased, implying a decreased average utilization. But since the priority of the MP3 stream "*Hellraiser*" has been increased to 100, it now has got 99.3% utilization. On the other side, the reduction of the "*Go West*" MP3 stream's priority to -100 decreased its average utilization to 51.8%.

Results

The session and stream priority system works as expected. The distribution is fair within a session to provide the same quality to inhomogeneous streams for the same user. Globally, the system maximizes the average utilization of the complete scenario to provide best possible quality to as many users as possible.

8.3 A System Load Scenario

The intention of this simulation is to demonstrate the system's buffering, layering and scalability behavior using a comprehensive example of a realistic and large scenario, consisting of serveral heterogeneous media streams having different maximum transfer delays. Table 8.4 shows the scenario. It contains several movies, action and sport videos but only one talk show and one news video. This seems to be realistic for a video and audio on demand service. The maximum transfer delays are set accordingly to the streams' bandwidth requirements: Large streams get higher delays than smaller ones, since it may be assumed that users try to reduce their cost. For example, receiving full-quality MPEG-2 streams having a maximum transfer delay of 1000ms at $\frac{1}{5}$th of the cost compared to 100ms. The DiffServ class settings from table 7.6 are used again to provide usage of the optimal class. Due to the usage of a bandwidth broker (see [Sel01] for details), inefficient SLAs can be minimized. Therefore, such an assumption is realistic. Five simulations have been computed for the given scenario:

1. No buffering, lowest possible transfer delay, no weighted remapping intervals, no layering:
   All streams have got a maximum transfer delay limit of 100ms. Therefore, only EF without buffering is usable. The unweighted remapping intervals are used. Maximum utilization is not limited (100%).

2. Buffering enabled:
   Like simulation #1, but using the maximum transfer delay settings as shown in table 8.4. Now, the AF classes and BE (for MPEG-2 enhancement layers only) are usable, too. But layered transmission is still disabled: All layers of a stream have to use the same DiffServ class!

3. Layered transmission enabled:
   Additionally, different layers may use different DiffServ classes.

4. Weighted remapping intervals:
   Further, the weighted remapping intervals are used.

Stream	Priority	Media	Media type	Transfer delay
Session #1, Priority 0				
#1	-100	Talk	MPEG-2	1,000 ms
#2	-50	The Silence of the Lambs	MPEG-1	350 ms
#3	0	Music	MPEG-2	1,000 ms
#4	50	Go West	MP3	300 ms
#5	-50	Born to be Wild	MP3	250 ms
Session #2, Priority 100				
#1	0	Hellraiser	MP3	350 ms
#2	0	Christmas Time ...	MP3	300 ms
#3	100	Siegfried-Idyll	MP3	200 ms
#4	50	Star Wars	H.263	300 ms
Session #3, Priority 0				
#1	-100	The Silence of the Lambs	MPEG-1	350 ms
#2	-100	Hellraiser	MP3	450 ms
#3	100	Goodbye Bora Bora	MP3	350 ms
Session #4, Priority -50				
#1	0	James Bond	MPEG-2	900 ms
#2	100	Terminator II	MPEG-1	350 ms
#3	0	Mr. Bean	H.263	300 ms
Session #5, Priority 0				
#1	0	Asterix	MPEG-1	350 ms
#2	0	Simpsons	MPEG-1	350 ms
#3	-100	News	MPEG-2	1,000 ms
#4	100	Speedy Gonzales	MP3	500 ms
Session #6, Priority 100				
#1	0	Jurassic Park	H.263	300 ms
#2	50	Die Firma	H.263	300 ms
#3	0	Formula One Race	MPEG-2	1,000 ms
#4	100	Super Bowl	MPEG-2	1,000 ms
#5	100	Seven Tears	MP3	150 ms

Table 8.4: The system load scenario

Sim.	Bandwidth*Time [Bytes]	Cost*Time [Bytes*$]	Avg. cost factor [$\frac{\$}{Byte*s}$]
#1	17,493,311,686	69,973,246,704	4.00000
#2	4,151,937,309 (-75.3 %)	12,805,231,022 (-81.7 %)	3.08416 (-22.9 %)
#3	5,043,425,335 (-71.2 %)	7,374,236,772 (-89.5 %)	1.46215 (-63.4 %)
#4	4,961,449,479 (-71.7 %)	7,124,407,931 (-89.8 %)	1.43595 (-64.1 %)
#5	2,100,218,331 (-88.0 %)	3,820,986,675 (-94.5 %)	1.81933 (-54.5 %)

Table 8.5: System load results

5. Maximum utilization limit of 75%:
   Only resource/utilization points having an utilization of less or equal 75% are used for the bandwidth remapping.

Figure 8.36 shows bandwidth (left side) and cost (right side) of the first simulation. Cost reduction (left side) and bandwidth reduction (right side) in percent for simulation #2 to #5 can be found in figure 8.37. Table 8.5 shows the total reserved bandwidth and cost and the average cost factor for each simulation for better comparision. The values within brackets present the difference compared to simulation #1 in percent.

As it is shown, the enabled buffering and the usage of all available classes already reduces the bandwidth requirement of this example by 75.3% and the cost by 81.7%. Allowing an own DiffServ class for each layer in simulation #3 gives an additional cost reduction of 7.8%. As it is shown by the average cost factor, much more BE bandwidth is used here by the enhancement layers of the MPEG-2 streams (1.46215, that is much lower than the cost factor of the cheapest reserved class AF11: 2.0). On the other side, the bandwidth is slightly increased by about 4.1%, due to the lower buffering gain using cheaper classes for some layers.

Since this scenario consumes most of the available bandwidth by large MPEG-2 streams (6 times larger than MPEG-1, see section 7.1), the weighted remapping interval calculation's gain of simulation #4 is only very small: 0.3%. The MPEG-2 streams have got a high maximum transfer delay for the reasons explained above. Since all layers map to cheap classes (usually AF for the base layers and BE for the enhancement layers), weighting does not result in a significant gain here. This corresponds to the MPEG-2 simulation results of section 8.1.3.

Finally, the maximum utilization limitation of 75% results in a bandwidth reduction by 88.0% and a cost reduction by 94.5% at an average utilization of 69.2%[1]. This means, that by accepting maximum transfer delays of up to 1000ms as given in table 8.4 and a utilization reduction to about 70%, the same scenario may be transported **nearly 20 times** to reach the same cost as for using 100ms transfer delay and 100% utilization!

Results

As it is shown, the buffering, layering, weighting and scalability simulations of section 8.1.3 and section 8.1.4 are also useful for a realistic scenario of inhomogeneous streams. The developed system works as expected and provides an excellent cost and bandwidth gain.

8.4 Complete and Partial Remappings

This simulation examines the effects of different partial remapping configurations. If sufficient bandwidth is available for all streams of a scenario, there is no difference of quality, cost and bandwidth between complete remappings and partial remappings. In this case, the partial remapping would always select the 100% utilized resource/utilization point. This is the same as a complete remapping finally would do. But if bandwidth is scarce, partial remappings may introduce inefficiency, since they only take care for keeping the invoking stream's utilization level, but not for bandwidth distribution among other streams (see section 6.4.2 for details). Therefore, the DiffServ class settings of table 8.6 are used to reach the desired effect for the used scenario. The scenario of system load simulation #4 (no utilization limit) of section 8.3 has been reused (see table 8.4 for the scenario), because an evaluation is only possible for a specific scenario: The settings for maximum time between two complete remappings and bandwidth fraction to be reserved exclusively for partial remappings (again,

[1] Only resource/utilization points having utilization **less or equal** 75% are used. If there is no point available at exactly 75%, the average utilization will be below 75%.

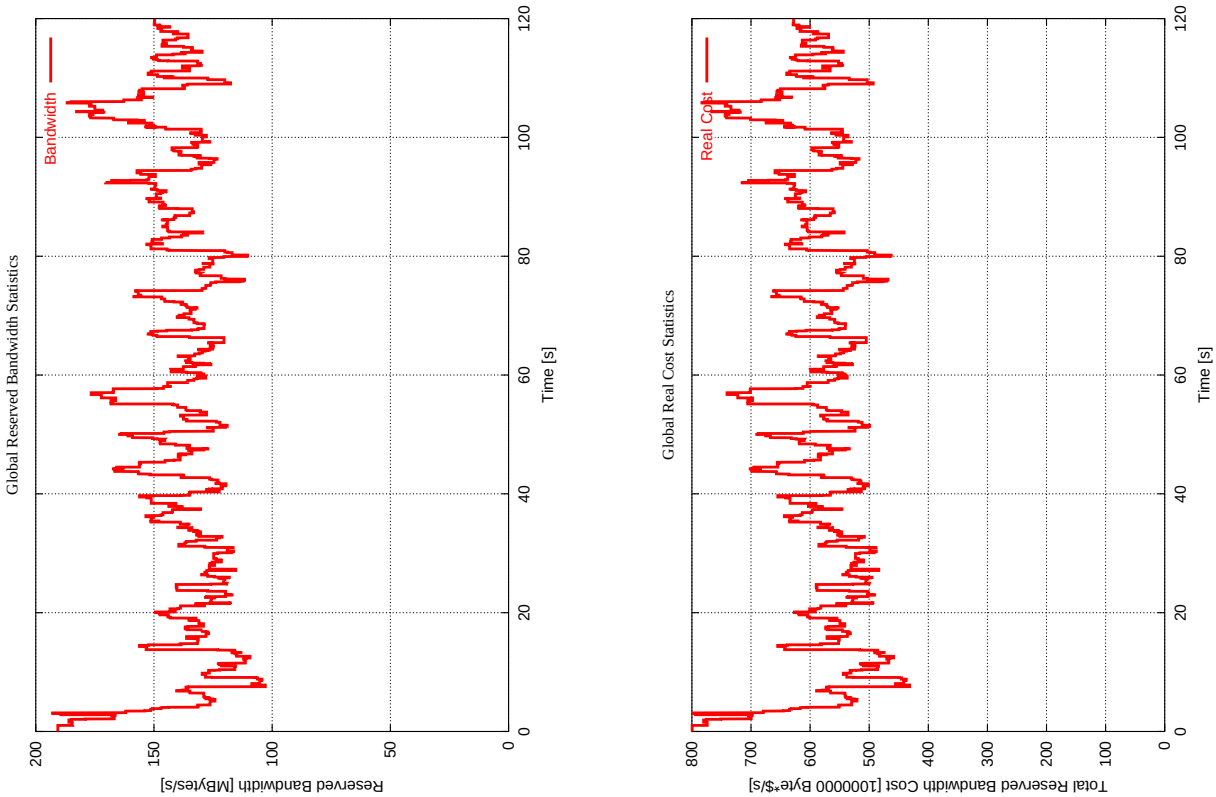


Figure 8.36: Total reserved bandwidth and cost of the first system load simulation

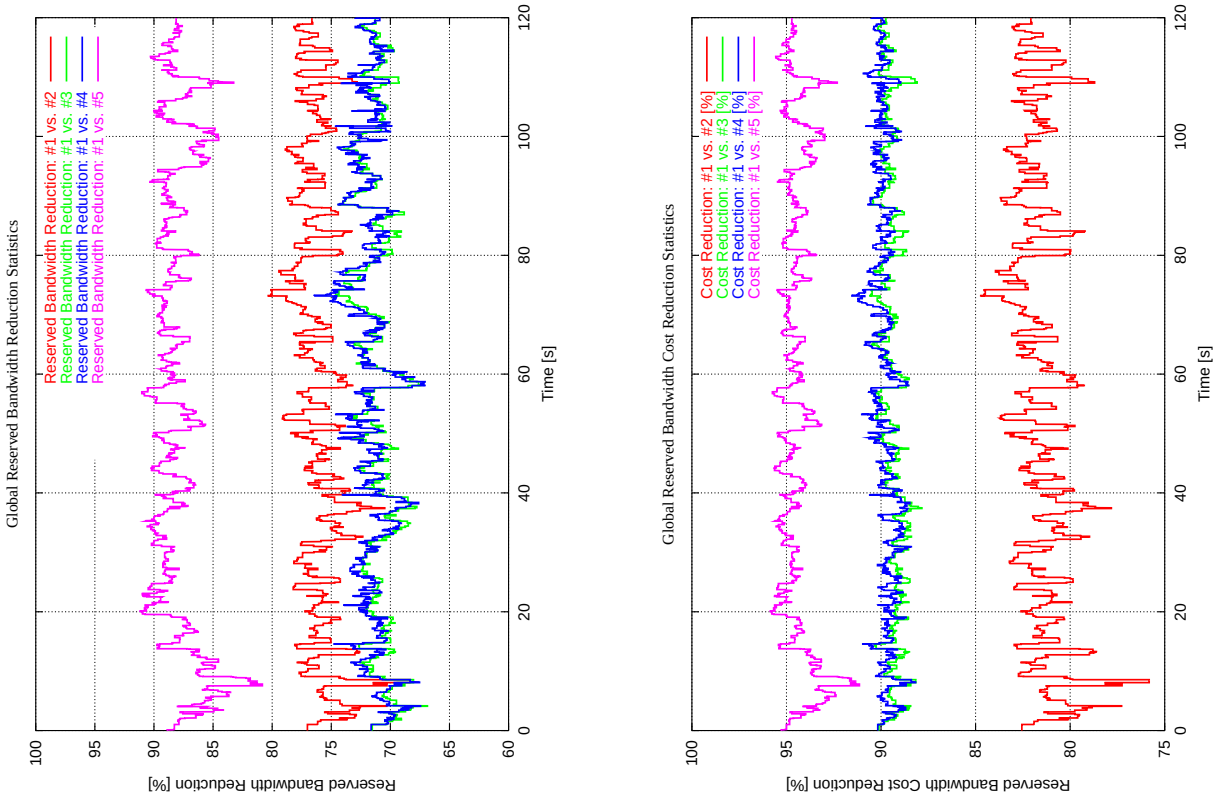


Figure 8.37: Reduction of bandwidth and cost, compared to figure 8.36

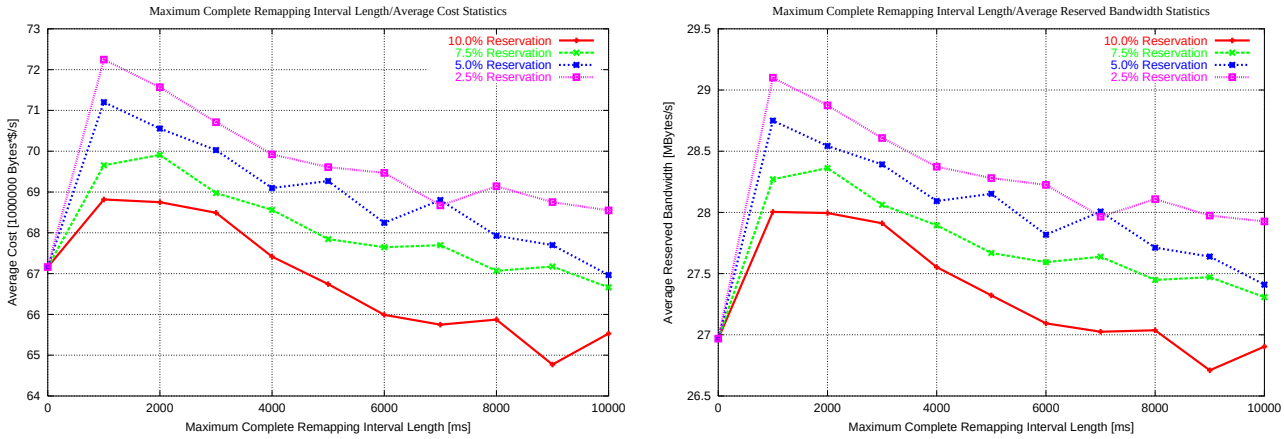


Figure 8.38: Average cost and bandwidth for different maximum complete remapping interval settings

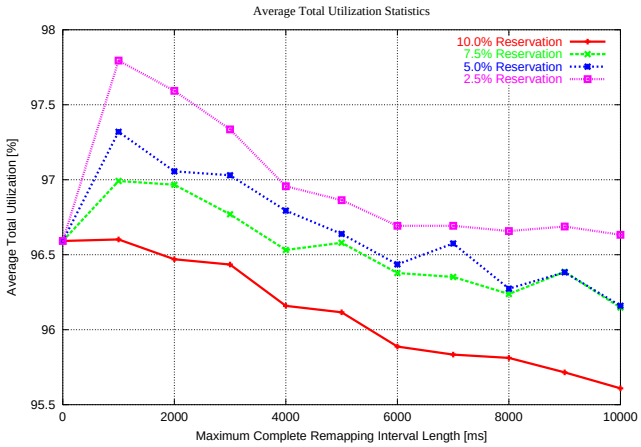


Figure 8.39: Average utilization for different maximum complete remapping interval settings

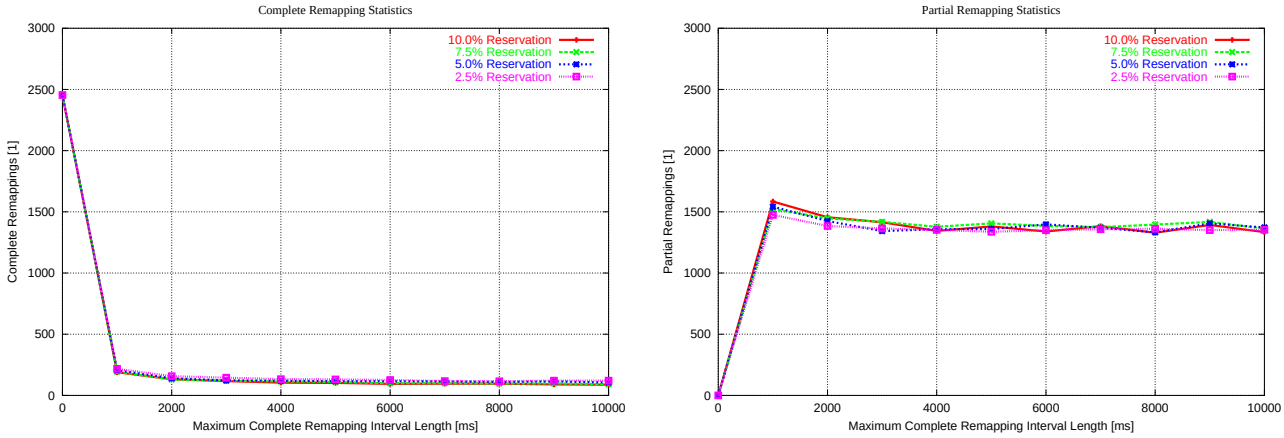


Figure 8.40: Remappings for different maximum complete remapping interval settings

Class	Bandwidth	Cost factor	Variability	Delay	Loss rate	Jitter
BE	4 MBytes/s	1.0	25 %	600 ms	10.00 %	200 ms
AF11	8 MBytes/s	2.0	10 %	300 ms	2.00 %	100 ms
AF21	7 MBytes/s	2.35	10 %	260 ms	1.00 %	80 ms
AF31	6 MBytes/s	2.65	10 %	230 ms	0.75 %	65 ms
AF41	5 MBytes/s	3.0	10 %	200 ms	0.25 %	50 ms
EF	3 MBytes/s	4.0	5 %	100 ms	0.05 %	10 ms

Table 8.6: Class settings for the partial remapping simulations

see section 6.4.2 for detailed explainations) are strongly dependent on the scenario. For example, the maximum time between two complete remappings may be higher and the reserved bandwidth fraction lower for scenarios containing only streams of low bandwidth variance (or even CBR) and vice versa.

Simulations for every of the following settings have been computed:

- Maximum times between two complete remappings: 0 to 10,000ms in steps of 1000ms. Here, 0ms means always using complete and therefore no partial remappings.
- Bandwidth fractions reserved for partial remappings: 2.5%, 5.0%, 7.5% and 10.0%.

The resulting average bandwidth and cost are plotted in figure 8.38, the average utilizations can be found in figure 8.39. Figure 8.40 shows the total amount of complete (left side) and partial remappings (right side) for every setting. As shown on the left side, the amount of complete remappings for 120s simulation time is reduced to less than 10% by the usage of partial remappings. But as expected, partial remappings usually increase cost and bandwidth requirements (see figure 8.38). The reason for these facts is the simplified bandwidth remapping, based only on the utilization of the stream which requires remapping (see section 6.4.2 for details). Therefore, more bandwidth may be given to a stream with increasing requirements. In this situation, a complete remapping would, assuming bandwidth is scarce, possibly decrease the stream's utilization. This is proven by figure 8.39: The average utilization is higher when partial remappings are used.

Further, the bandwidth and cost increment is higher for a lower reserved bandwidth fraction: Since bandwidth is scarce due to the bandwidth settings of table 8.6, most cheap bandwidth has already been allocated during complete remappings. Therefore, only the reserved amount of cheap bandwidth is still available. Therefore, smaller reservations cause higher cost.

Comparing cost and bandwidth for different maximum times between two complete remappings, decreasing cost and bandwidth requirements can be noticed (see figure 8.38). Obviously, inefficient allocations are compensated by efficient ones due to averaging over longer intervals. But further, also the average utilization is decreased (see figure 8.39).

Results

Partial remappings save a large amount of complete remappings but may increase cost and bandwidth requirements to increase the system's total utilization. These variations are strongly dependent on the scenario and the both settings for partial remappings: The maximum time between two complete remappings and the bandwidth fraction exclusively reserved for partial remappings. These parameters have to be adapted by a system administrator to achieve a good compromise between remapping efficiency, additional cost and bandwidth requirements and the overall user satisfaction.

Stream	Session	Media	Type	Transfer delay
#1	#1	The Silence of the Lambs	MPEG-2	1000 ms
#2	#2	Talk	MPEG-1	1000 ms
#3	#3	Born to be Wild	MP3	1000 ms
#4	#4	ARD News	H.263	1000 ms

Table 8.7: The real network scenario

Delay tolerance [ms]	0	10	20	30	40	50	60	70	80	90	100
Measurement set #1	386	275	126	46	39	23	20	13	14	10	11
Measurement set #2	391	274	120	47	36	24	19	15	11	17	11
Average	$388\frac{1}{2}$	$274\frac{1}{2}$	123	$46\frac{1}{2}$	$37\frac{1}{2}$	$23\frac{1}{2}$	$19\frac{1}{2}$	14	$12\frac{1}{2}$	$13\frac{1}{2}$	11
Average per minute	$194\frac{1}{4}$	$137\frac{1}{4}$	$61\frac{3}{4}$	$23\frac{1}{4}$	$18\frac{3}{4}$	$11\frac{3}{4}$	$9\frac{1}{2}$	7	$6\frac{1}{4}$	$6\frac{3}{4}$	$5\frac{1}{2}$
$\frac{\text{Average}}{\text{Layers}*\text{Minutes}}$	11.43	8.07	3.62	1.37	1.10	0.69	0.57	0.41	0.37	0.40	0.32

Table 8.8: Total number of buffer flushes for different system delay tolerances

8.5 Measurements on a Real Transport Scenario

As the simulation results of the previous sections show, the complete system works quite well. But using a real system, inaccuracy is introduced by process scheduling as explained in section 7.3: Transmissions may be started too early or too late, causing overflows of the leaky bucket buffers. Therefore, it is necessary to check which system delay tolerance (see section 6.2.2 for details, especially formula 6.2) is necessary to provide a reliable service, that is having a low number of buffer flushes. Further, some duration measurements for the stream description initializations (see section 6.2) and the complete remappings (see section 6.4.1) are necessary to show the system's real-time usability.

The system is configured as explained in section 7.3, see also figure 7.4 for the scenario. An UDP sender transmits background traffic via BE from *amstel* to *gaffel* at a constant rate of 375,000 Bytes/s (3 MBit/s). The available bandwidth for the trace server is set to 1,000,000 Bytes/s (8 MBit/s) for BE, 1,250,000 Bytes/s (10 MBit/s) for each of both AF classes and 625,000 Bytes/s (5 MBit/s) for EF. Therefore, BE would be overloaded by 10% in case of full usage. Table 8.7 shows the scenario, it contains one stream of each examined type. Each stream has got its own session, all priorities are set to zero. The maximum transfer delay is set to 1000ms to ensure a high buffer usage.

A measurement set for all system delay tolerances from 0ms (none) to 100ms in steps of 10ms has been run. Each of the single measurements has got a duration of 120 seconds. To ensure, that the results are not affected by other running processes like CPU- and I/O-intensive *cron* jobs, the complete measurement set has been repeated some time later. For comparision, a simulation is done to find out the number of 'allowed' buffer flushes, caused by the speed adjustment approach at quality changes as explained in section 7.3.

Table 8.8 shows the total number of buffer flushes for both sets. A plot of these results can be found in figure 8.41. Further, table 8.8 also shows the average of both measurements, the average buffer flushes per minute (total playtime: 120s) and per transport layer and minute. There is a total of 17 layers here: 9 for MPEG-2, 3 for MPEG-1, 4 for H.263 and 1 for MP3 (see section 4.2 for details). As it is shown, both measurements only differ by a small inaccuracy, except for an outlier at 90ms for set #2. The number of 'allowed' buffer flushes (see section 7.3) in the simulation is 6, therefore an average of 3 per minute and 0.18 per minute and layer. Note, that the table's values also incorporate these 'allowed' flushes!

As expected, the number of buffer flushes is extremely high for no system delay tolerance. This

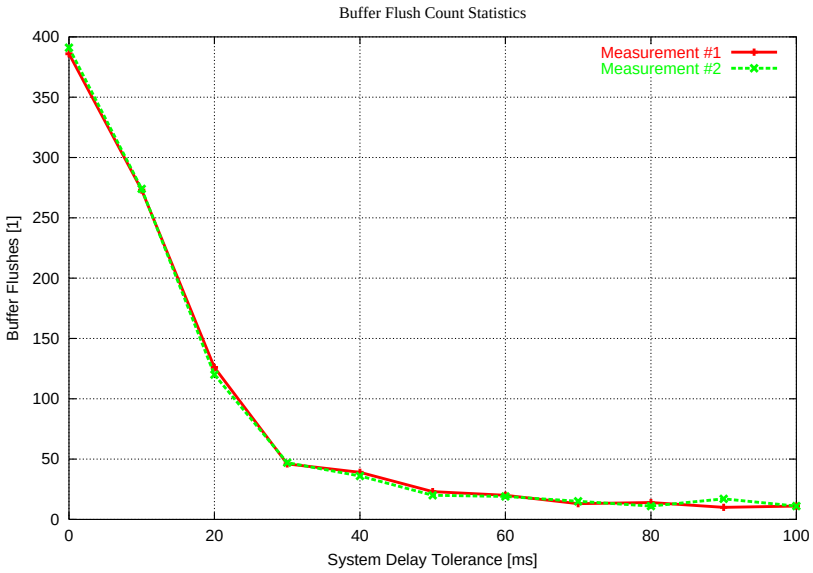


Figure 8.41: Total number of buffer flushes for different system delay tolerances

Media type	Average duration
MPEG-1	6 ms
MPEG-2	16 ms - 18 ms
H.263	6 ms - 8 ms
MP3	1 ms

Table 8.9: Average durations of stream description initializations

number decreases as higher the tolerance is set - by about 90% for 40ms and more than 95% for 70ms. As a result, system delay tolerances between 50ms and 70ms seem to be realistic values for the used INTEL-based LINUX system, leading to about one buffer flush per layer within two minutes (inclusive 'allowed' ones!). For applications like video and audio on demand, this delay tolerance is realistic and acceptable. As the simulations in section 8.1.3 show, no significant reduction of the buffering gain is introduced for maximum transfer delay requirements of more than about 600ms (somewhat more for MPEG-2). But if very low delay tolerances are really needed, this could be achieved by porting the system to a real-time operating system like RTLINUX (see [RTLinux]). In this case, very strict scheduling limits can be specified, leading to expected delay tolerances of about 1ms.

Finally, table 8.9 shows the average durations of stream description initializations as explained in section 6.1. That is, translating the payload bandwidths to raw ones, finding out all possible DiffServ classes and buffer delays for each layer, calculating bandwidth and cost requirements for each layer and possible DiffServ class, sorting the choices by cost and eliminating inefficient resource/utilization points. Such a stream description initilization is always necessary if the media reaches a new resource/utilization list. Since this is independent of all other streams, these calculations are done as soon as possible and not just during the complete remapping (see section 6.5). Therefore, they distribute homogeneously over the complete playtime. As it is shown in table 8.9, the results are quite acceptable for a real-time system. The resulting average duration of a complete remapping is about 6.5ms to 7.5ms for the scenario's four streams, which is quite efficient.

Results

As it is shown, the used INTEL-based LINUX system produces quite acceptable results for system delay tolerances of about 50ms to 70ms. Therefore, the simulation results of the previous sections can be adapted to the real system by adding this small system delay tolerance. Further, the durations

of stream description initializations and complete remappings are quite low and provide efficient real-time usage of the system.

8.6 Summary

The goal of this chapter has been the evaluation of the system by simulations and real network measurements. The first part of the simulations has examined buffering, the weighted remapping intervals, layered transmission and scalability. This has begun with two comprehensive examples for the effects of buffering and the usage of weighted remapping intervals. Then, buffering, the weighted remapping intervals and layered transmission have been examined for a large set of traces and a wide range of maximum transfer delay limits. This has leaded to the following results:

- The cost optimization works as expected: The system always uses to most cost-efficient DiffServ class.
- Buffering results in a huge cost and bandwidth gain, even for small buffer delays.
- Weighted remapping interval calculation causes the choice of remapping intervals to be more affected by 'expensive' layers: Especially when expensive DiffServ classes are used, there is a significant cost gain noticable.
- Layered transmission reduces cost by the ability of using cheaper DiffServ classes for layers having lower QoS requirements. Especially for large QoS differences (see MPEG-2), this cost gain can be huge.

Next, the effects of scalability have been examined using different utilization limits from 25% to 100% with the following results:

- Streams of low frame rate and/or frame size requirements need a smaller amount of bandwidth when scaled.
- Frame rate scaling reduces the buffering ability due to smaller frame 'gaps'. Therefore, the higher the frame rate, the higher the buffering gain.
- MP3 does not use frame rate scalability. This results in no reduction of the buffering ability when scaled.

The second part of the simulations first has examined the system's behavior on changes of the network's quality of service. As it has been shown, the system works as expected and adapts the layer to DiffServ class mappings as necessary to fit the given QoS requirements. Further, the cost optimization tries to keep the additional cost as low as possible. The next simulation has shown the behavior of the priority system for a utility-fair distribution within sessions and a utilization-maximizing global mapping.

Finally, the last part of the simulations has examined the system's behavior on a large, realistic video/audio on demand scenario of inhomogeneous streams. As it has been shown, the system also works well for such scenarios. Further, the effects of using partial remappings to reduce the amount of complete remappings for saving CPU power have been examined on the scenario above. As it has been shown, this amount can be highly decreased. But the effects on cost, bandwidth and utilization are strongly dependent on the scenario and the two partial remapping parameters: The maximum time between two complete remappings and the bandwidth fraction to be exclusively reserved for partial remappings. These parameters have to be tuned by a system administrator for a certain system to

achieve a good compromise between the reduction of complete remappings and cost, bandwidth and utilization.

The chapter has closed with some measurements on a real DiffServ network scenario to examine system delay tolerances and durations. The results have shown, that tolerances of about 50ms to 70ms for the used LINUX system are sufficient. Therefore, the simulation results can be adapted to a real system. Further, the measurement of durations for the stream description initializations and complete remappings has shown the system's real-time usability.

Chapter 9

Conclusions

The design, implementation and evaluation of an efficient solution to manage layered and scalable variable bitrate multimedia streams in the CORAL project, based on a priori analyzation of the medias, has been the global goal of this work.

Design and Implementation

First, the a priori analyzation of multimedia streams has been developed, consisting of two steps:

1. The a priori remapping interval calculation algorithm of section 2.4 has been extended to support layered transmission using an own D-BIND traffic description for each layer and a weighted sum for the total cost. This weighting has improved the cost function to be more affected by bandwidth requirement changes of more expensive layers, resulting in more cost-efficient bandwidth remappings. Further, scalability support has been added.

2. An efficient algorithm for the calculation of the so called resource/utilization lists has been developed. It generates a homogeneous distribution of the points over the whole utilization range from 0% to 100%.

Next, the online bandwidth management for multimedia streams of different media types has been developed by extending a QoS optimization algorithm (ASRMD1, see section 2.5.4). The resulting algorithm's properties are:

- Usage of several DiffServ classes instead of only one resource (= bandwidth),
- cost-optimized usage of buffering,
- support for sessions by the usage of so called resource/utilization multipoints,
- independently configurable fairness for streams and sessions and
- priorities for streams and sessions.

Evaluation

To evalute the implemented system, simulations and measurements have been made. These simulations have shown that the system works as expected and provides a cost efficient transmission for a priori analyzed, variable bitrate multimedia streams by the usage of cost-optimized buffering, layered transmission and weighted remapping intervals. Further, due to the usage of utility functions, it is possible to evaluate the effects of bandwidth changes to the user satisfaction. Therefore, it is possible to provide utility-fair sharing and the maximization of the global utilization. Since global fairness and

fairness within sessions are independently configurable, it is possible to provide utility-fair sharing for streams within a session (desired by the user, e.g. 50% utilization for a video and its audio), but to maximize the system utilization globally (desired by the provider, e.g. only give 10% utilization to a large video session and 100% to 20 small audio sessions, instead of e.g. 20% to all). To emphasize different importances of some sessions or streams, priorities can be given for both.

Finally, it has been shown, that the simulations' results may also be applied to a real LINUX-based network scenario of INTEL-based PCs by allowing a small system delay tolerance of 50ms to 70ms. Although this prevents the system from usage for extremely low transfer delay requirements, it should be sufficient for most audio/video on demand purposes. In this case, it may be assumed that the users accept some transfer delay, since buffering highly reduces their cost. Section 8.1.3 has shown, that e.g. full-quality MPEG-2 streams having a maximum transfer delay of 1000ms can be transmitted at $\frac{1}{5}$th of the cost compared to 100ms transfer delay.

Further Examinations and Ideas

There are several issues which could not be discussed in this work but which should be examined more closely in the future:

- At the moment, the bandwidth broker (see section 3.1) is still under development (see [Sel01]). Therefore, no measurements have been possible. It would be interesting to examine the efficiency of a scenario of several servers having their SLAs managed by the bandwidth broker.
- Further, more examinations of the partial remapping configuration are necessary. In this work, only constant settings are used. It may be assumed, that a dynamic adaption may highly improve its efficiency. This results in the challenge to develop an algorithm for this adaption.
- Next, more tests of different fairness configurations can be made. This especially includes the fairness settings 'between' utility-fairness and system utilization maximization.
- An additional cost and bandwidth reduction is assumed by the adaptive choice of layer weights during the remapping interval calculation. Adapting these weights to the current position's traffic behavior will probably result in better remapping intervals than using constants for the whole stream. Especially, this seems to be effective for H.263, where no constant GoP is used.
- The simulations described in this work only concern the bandwidth management. Network quality is only introduced by constant settings. Therefore, it would be very interesting to do a simulation of the complete system in a large DiffServ network scenario, using for example the network simulator *ns2*.
- If extremely low transfer delay requirements are necessary, the delay tolerances of the LINUX-based implementation may be too high. In this case, porting the system to a real-time operating system like RTLINUX (see [RTLinux]) will probably highly decrease this necessary delay tolerance. This could be examined more closely.
- In [Vey01], the transmission of variable bitrate streams without a priori knowledge is examined. An efficiency comparision of both methods for the same scenario would therefore be interesting.
- Finally, it would be very interesting to implement the transport of real medias instead of traces. This would lead to the possibility of using quality metrics for the utility function calculation and to do user ratings to evaluate the clients' outputs.

Appendix A

Trace Statistics

This appendix contains the trace statistics as explained in section 7.1. FS_{Min}, FS_{Max} and FS_{Avg} denote the minimum, maximum and average original frame size for all layers and B_{Total} denotes the global burstiness. B_I, B_P and B_B show the burstiness of the corresponding frame type only. It is important to denote here, that the burstiness calculation also includes frame sizes of 0. For example, as shown in the layering example of table 4.1, an I-frame is sent every 12th frame. Since zero-sized frames are important for the buffering gain, it is useful to also include them into the calculation of burstiness.

Note, that since the enhancement layers of the MPEG-2 traces are a result of the multiplication by a constant factor (see section 7.1), their burstiness is the same as for the base layer.

Futher, it is important to explain some values for the H.263 traces of table A.3: Since the traces contain only one, two or even no (see "*ARD Talk*") I-frames for the reasons described in section 2.6.3, the calculated I-burstiness is extremely high. The only exception is "*Sendung mit der Maus IP*", which uses the constant GoP "IPIP . . . ". Further, since it is usually most efficient to use PB-frames (see section 2.6.3 for details), no B-frames are used in any trace.

Name	Genre	N_{25}	FS_{Min}	FS_{Max}	FS_{Avg}	B_{Total}	B_I	B_P	B_B
Asterix	Cartoon	530	304	147376	20522	7.18	26.3	18.2	10.1
James Bond	Movie	443	1912	168608	22229	7.58	26.2	15.7	8.3
Lambs	Movie	306	304	134224	7530	17.82	41.1	45.9	23.6
Mr. Bean	Comedy	297	1760	229072	16490	13.89	22.6	26.8	37.1
MTV	Music	620	816	229200	25733	8.90	33.0	22.3	11.4
News	News	366	1760	194416	19929	7.76	27.8	22.4	12.8
Formula 1	Sports	549	4192	186048	30867	6.03	28.0	16.0	6.4
Simpsons	Cartoon	563	336	148496	19204	7.73	23.8	24.6	10.1
Super Bowl	Sports	494	312	140840	23279	6.05	25.1	13.9	10.4
Talk	Talk	304	2728	106768	14274	7.48	19.9	19.5	6.8
Terminator II	Action	390	320	79560	11168	7.12	25.6	16.9	9.2
			Total				Base Layer		

Table A.1: MPEG-1 trace statistics

Name	Genre	N_{25}	FS_{Min}	FS_{Max}	FS_{Avg}	B_{Total}
Asterix	Cartoon	774	1824	884256	123132	7.18
James Bond	Movie	693	11472	1011648	133374	7.58
Lambs	Movie	521	1824	805344	45182	17.82
Mr. Bean	Comedy	501	10560	1374432	98941	13.89
MTV	Music	837	4896	1375200	154399	8.90
News	News	551	10560	1166496	119577	7.76
Formula 1	Sports	796	25152	1116288	185200	6.03
Simpsons	Cartoon	794	2016	890976	115226	7.73
Super Bowl	Sports	769	1872	845040	139672	6.05
Talk	Talk	543	16386	640608	85643	7.48
Terminator II	Action	737	1920	477360	67006	7.12
			Total			

Name	B_I	B_P	B_B	B_{E_1I}	B_{E_1P}	B_{E_1B}	B_{E_2I}	B_{E_2P}	B_{E_2B}
Asterix	26.3	18.2	10.1	26.3	18.2	10.1	26.3	18.2	10.1
James Bond	26.2	15.7	8.3	26.2	15.7	8.3	26.2	15.7	8.3
Lambs	41.1	45.9	23.6	41.1	45.9	23.6	41.1	45.9	23.6
Mr. Bean	22.6	26.8	37.1	22.6	26.8	37.1	22.6	26.8	37.1
MTV	33.0	22.3	11.4	33.0	22.3	11.4	33.0	22.3	11.4
News	27.8	22.4	12.8	27.8	22.4	12.8	27.8	22.4	12.8
Formula 1	28.0	16.0	6.4	28.0	16.0	6.4	28.0	16.0	6.4
Simpsons	23.8	24.6	10.1	23.8	24.6	10.1	23.8	24.6	10.1
Super Bowl	25.1	13.9	10.4	25.1	13.9	10.4	25.1	13.9	10.4
Talk	19.9	19.5	6.8	19.9	19.5	6.8	19.9	19.5	6.8
Terminator II	25.6	16.9	9.2	25.6	16.9	9.2	25.6	16.9	9.2
	Base Layer			Enhacement 1			Enhancement 2		

Table A.2: MPEG-2 trace statistics

Name	Genre	Length	Source	N_{30}
ARD News	News	833 s	[Ber00]	123
ARD Talk	Talk	833 s	[Ber00]	123
Die Firma	Movie	833 s	[Ber00]	142
Sendung mit der Maus IP	Cartoon	55 s	Bonn	17
Sendung mit der Maus HQ	Cartoon	160 s	Bonn	31
Sendung mit der Maus LQ	Cartoon	160 s	Bonn	18
Formula 1	Sports	833 s	[Ber00]	337
Jurassic Park	Movie	833 s	[Ber00]	279
Mr. Bean	Comedy	833 s	[Ber00]	235
N3 Talk	Talk	833 s	[Ber00]	201
Star Wars	Action	833 s	[Ber00]	106
Tagesschau HQ	News	83 s	Bonn	23
Tagesschau LQ	News	83 s	Bonn	14

Name	FS_{Min}	FS_{Max}	FS_{Avg}	B_{Total}	B_I	B_P	B_{PB}	B_B
ARD News	54	15310	3445	4.44	-	20.8	5.6	-
ARD Talk	449	9562	2299	4.16	14932	55.6	3.9	-
Die Firma	34	9173	1453	6.31	12984	38.9	7.4	-
Sendung mit der Maus IP	56	18807	6211	3.03	3.5	12.4	-	-
Sendung mit der Maus HQ	56	18807	2875	6.54	4680	6.5	-	-
Sendung mit der Maus LQ	56	7318	931	7.86	4680	7.9	-	-
Formula 1	438	14114	3924	3.60	24990	7.0	6.59	-
Jurassic Park	24	18168	4180	4.35	24990	15.3	5.4	-
Mr. Bean	54	16221	2995	5.41	24990	21.5	7.0	-
N3 Talk	68	13956	2267	6.16	13748	50.1	4.7	-
Star Wars	20	8989	1092	8.23	24990	21.1	10.7	-
Tagesschau HQ	199	24416	3068	7.91	2490	7.9	-	-
Tagesschau LQ	75	8722	746	11.70	2490	11.7	-	-

Table A.3: H.263 trace statistics

Name	Length	Source	N_{38}	FS_{Min}	FS_{Max}	FS_{Avg}	B_{Total}
Born to be Wild	211 s	LAME	24	104	1044	754	1.38
Christmas Time Is ...	173 s	LAME	19	104	1044	656	1.59
Die glorreichen Sieben	340 s	LAME	42	104	1044	696	1.50
Doggy Dogg World	271 s	LAME	29	104	1044	671	1.56
Eat the Rich	277 s	LAME	41	104	1044	746	1.40
Go West	305 s	LAME	28	104	1044	697	1.50
Bora Bora	236 s	LAME	26	104	1044	685	1.53
Hellraiser	277 s	LAME	36	104	1044	681	1.53
Iron Fist	170 s	LAME	24	104	1044	665	1.57
It's Christmas ...	153 s	LAME	17	104	1044	661	1.58
Old Pop in an Oak	213 s	LAME	23	104	1044	690	1.51
Positive NRG	172 s	LAME	17	104	1044	688	1.52
Reincarnation	405 s	LAME	38	104	1044	670	1.53
Robin Hood	353 s	LAME	46	104	1044	682	1.53
San Francisco	178 s	LAME	27	313	1044	683	1.53
Seven Tears	229 s	via NAPSTER	22	104	1044	706	1.48
Siegfried-Idyll	913 s	via NAPSTER	89	104	835	514	1.62
Speedy Gonzales	154 s	LAME	17	104	1044	666	1.57
Summer in the City	161 s	LAME	19	104	1044	822	1.27
Terminator	128 s	LAME	28	104	1044	691	1.51
Tutti Frutti	121 s	LAME	16	313	1044	768	1.36
We Wish You a ...	76 s	LAME	7	104	1044	714	1.46
White Christmas	187 s	LAME	31	313	1044	696	1.50

Table A.4: MP3 trace statistics

Appendix B

Buffering Measurement Results

This appendix contains the buffering simulation results for each trace as explained in section 8.1.3. Each table shows the plots for the average (divided by the simulation duration of 800s) bandwidth, cost or cost factor of simulation #1 in absolute values. For better comparision of simulation #1 with the results of simulation #2 to #4, these plots display the difference ($value_2 - value_1$) to simulation #1 in percent!

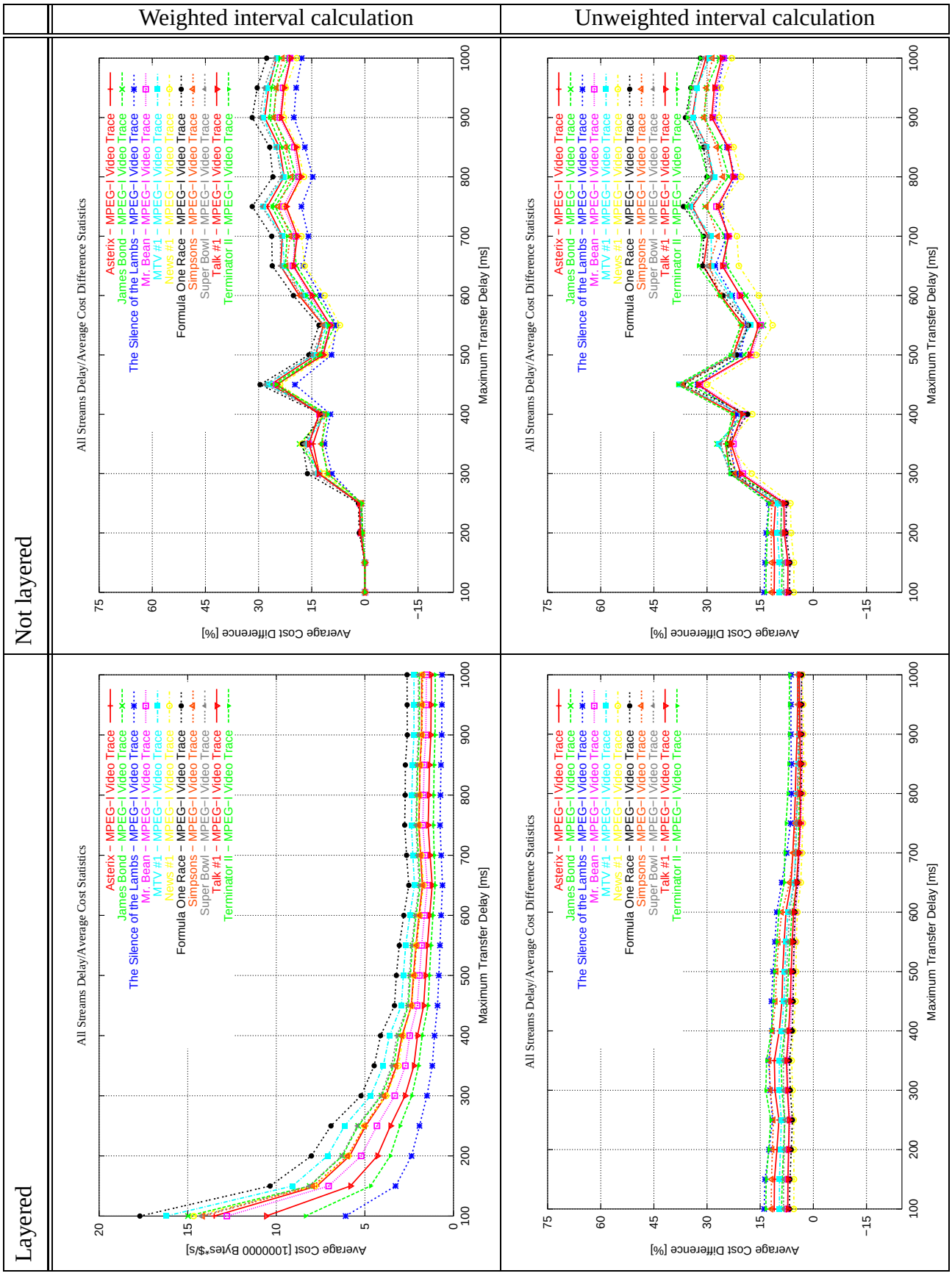


Figure B.1: MPEG-1 cost/delay comparision

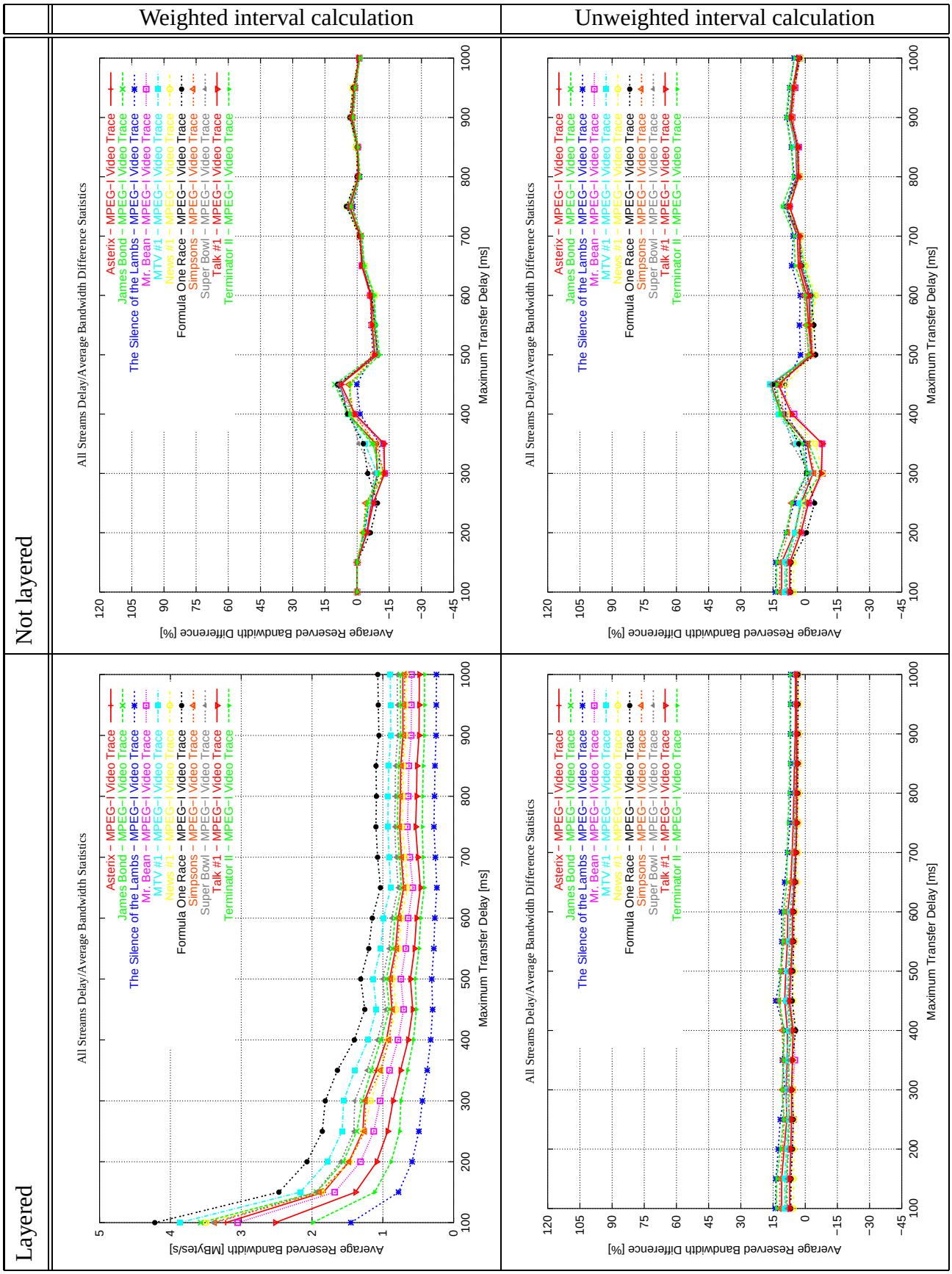


Figure B.2: MPEG-1 bandwidth/delay comparision

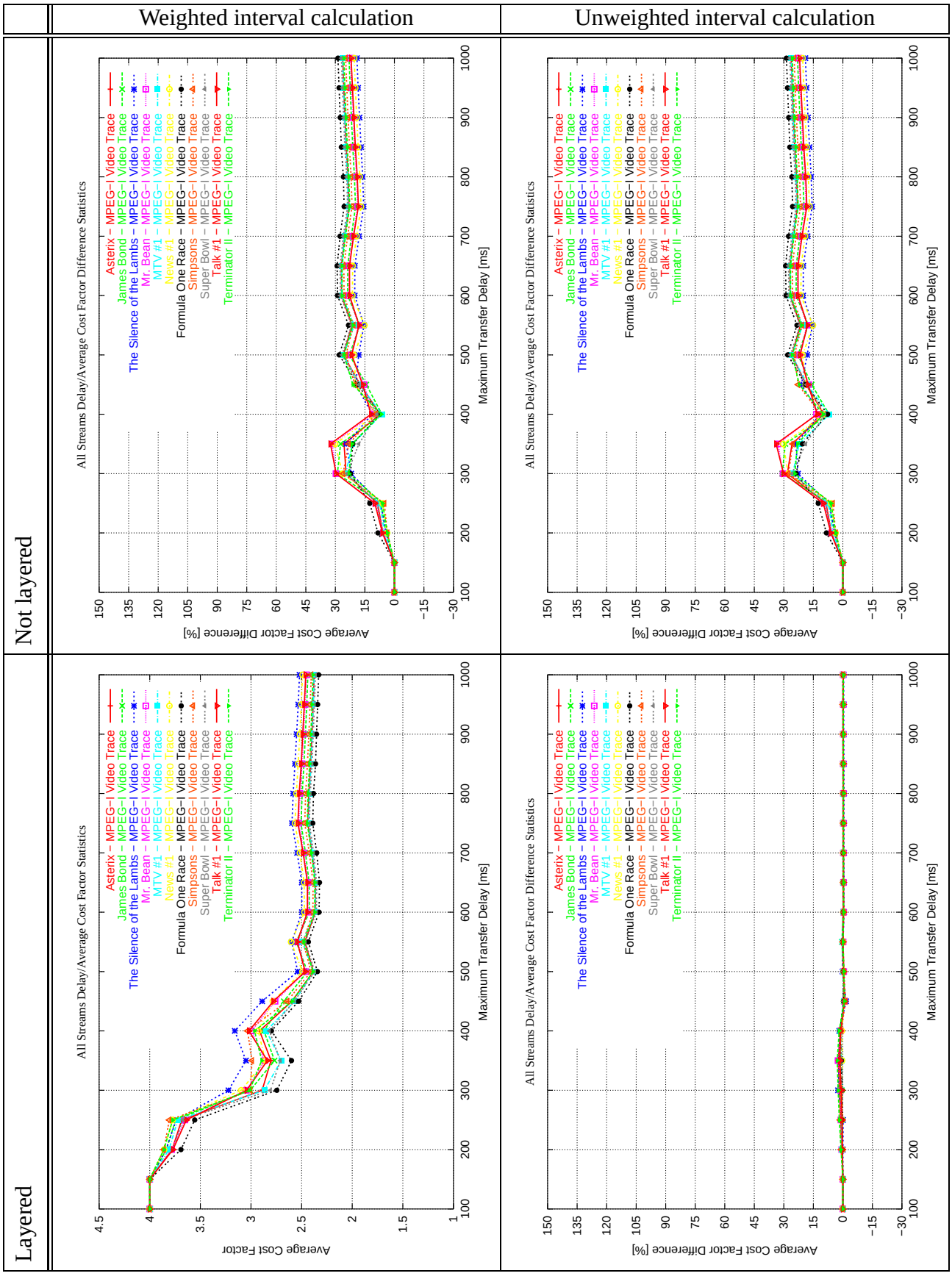


Figure B.3: MPEG-1 average cost factor/delay comparision

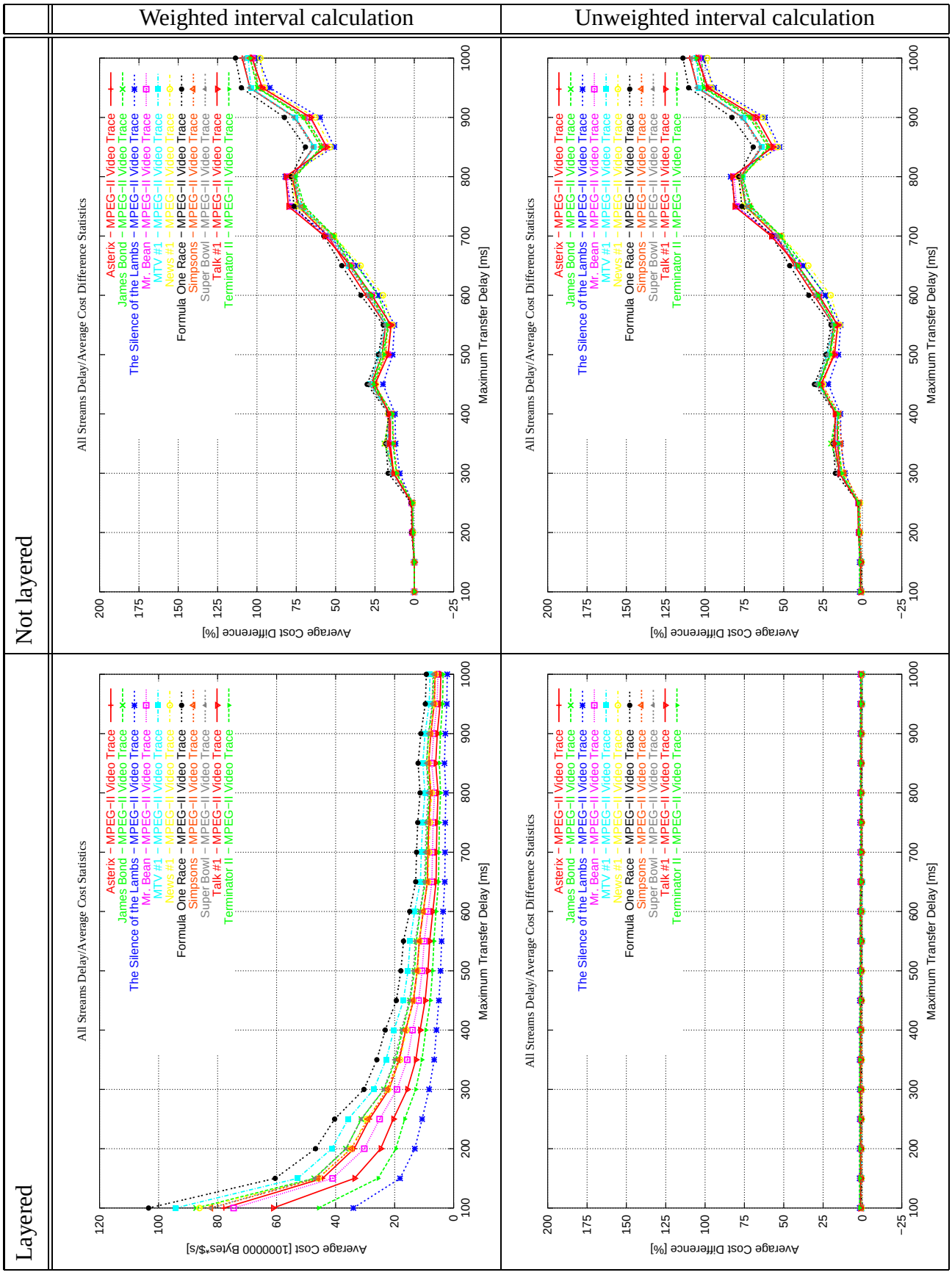


Figure B.4: MPEG-2 cost/delay comparision

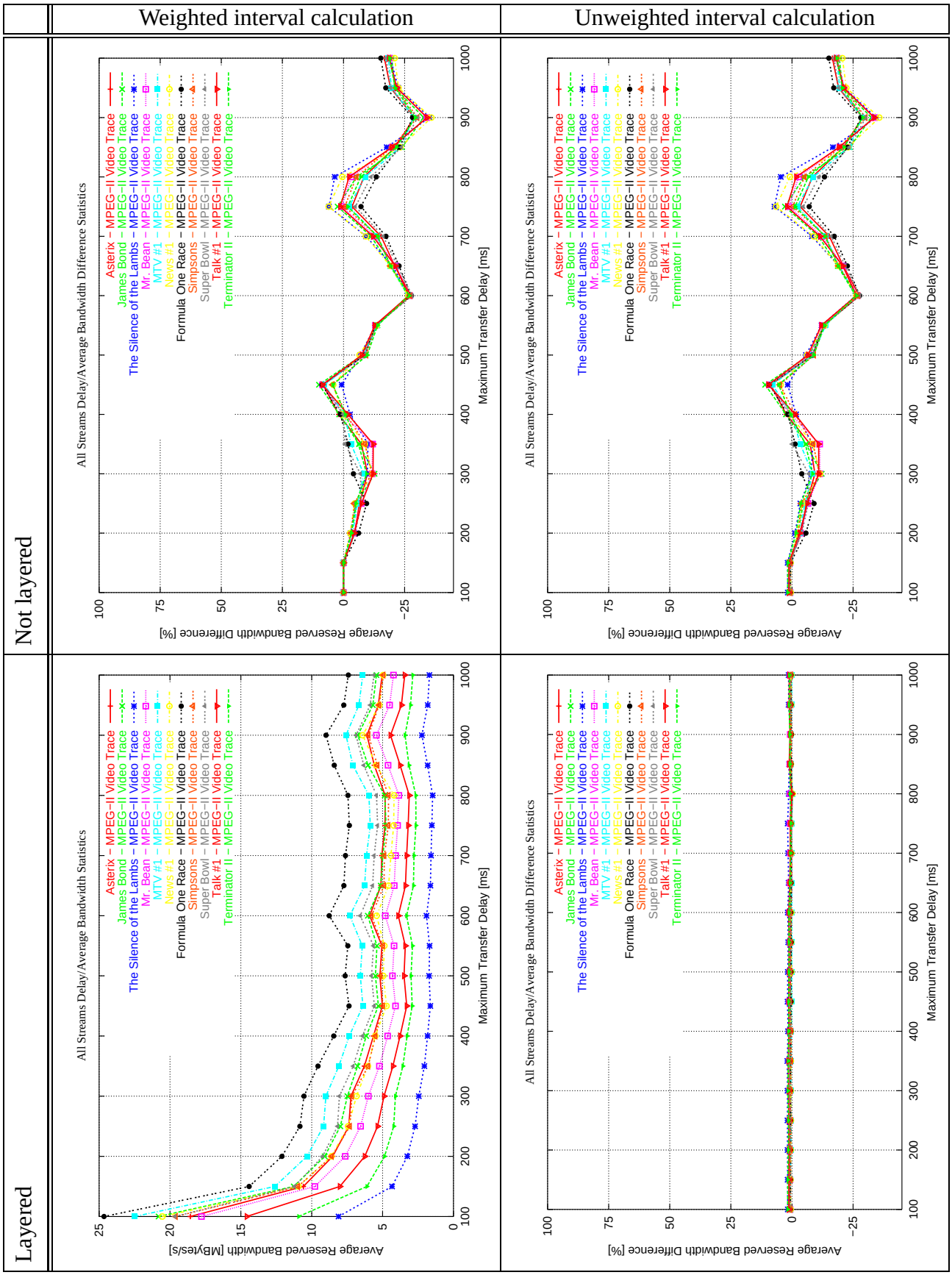


Figure B.5: MPEG-2 bandwidth/delay comparision

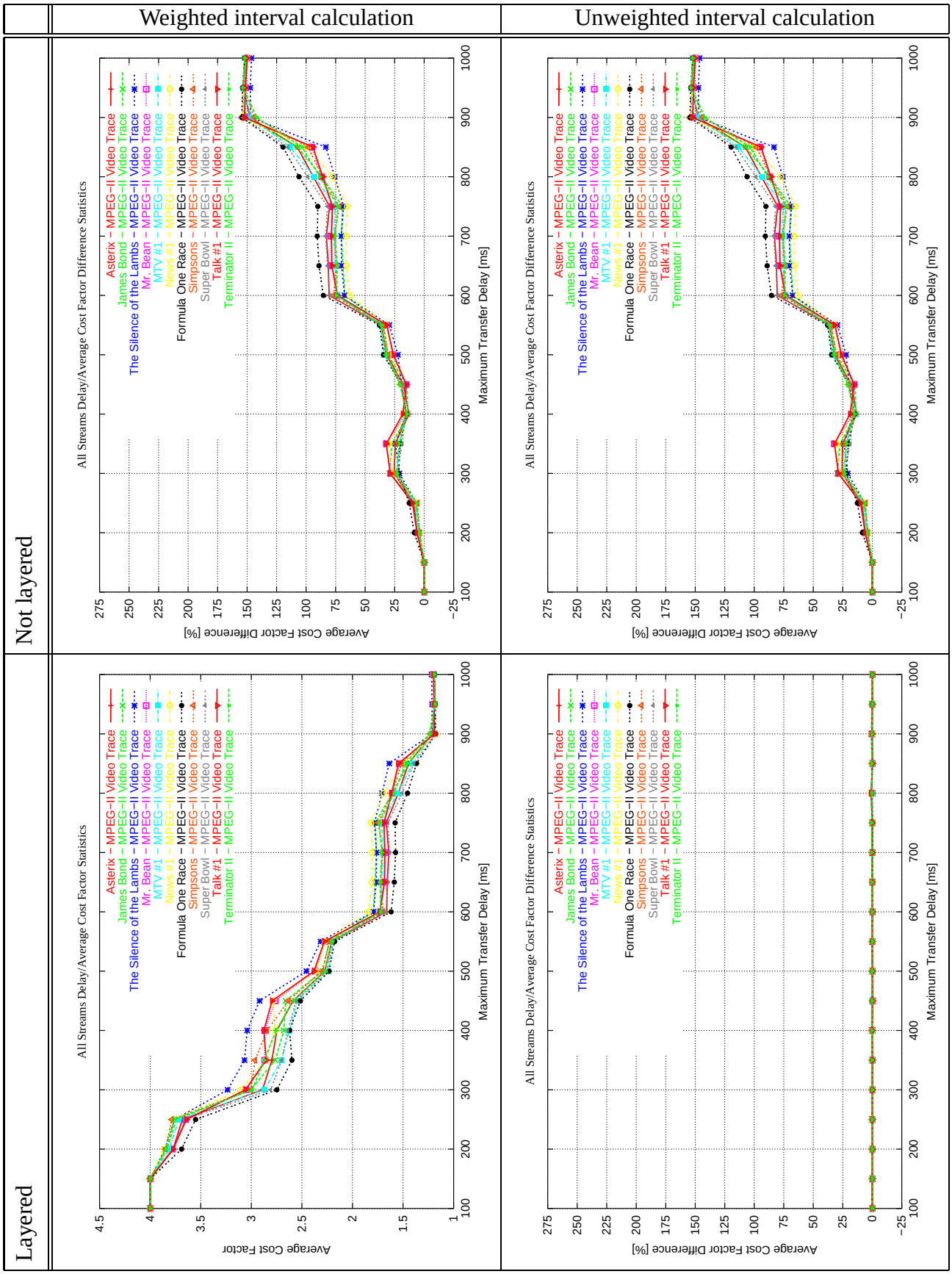


Figure B.6: MPEG-2 average cost factor/delay comparision

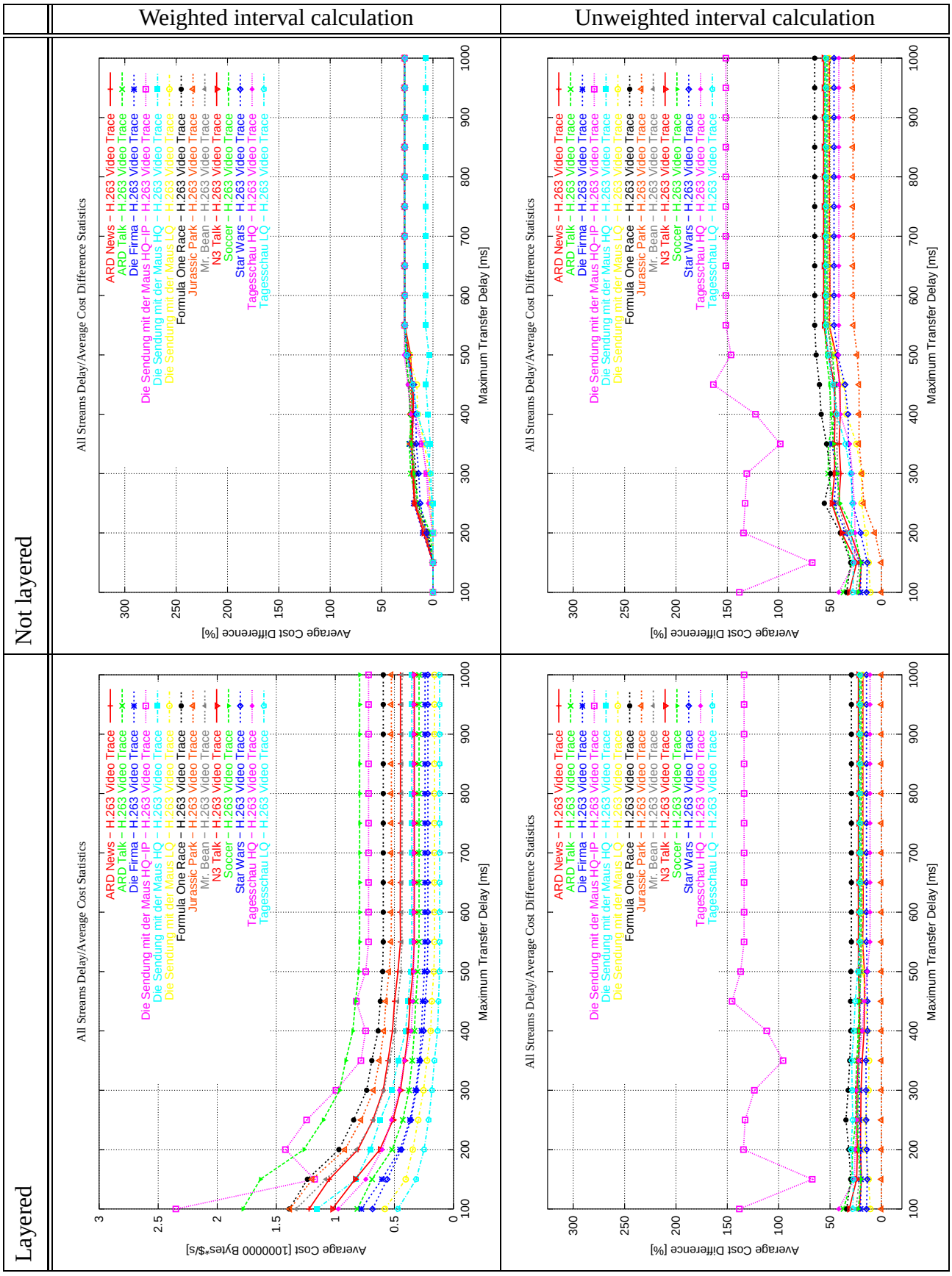


Figure B.7: H.263 cost/delay comparision

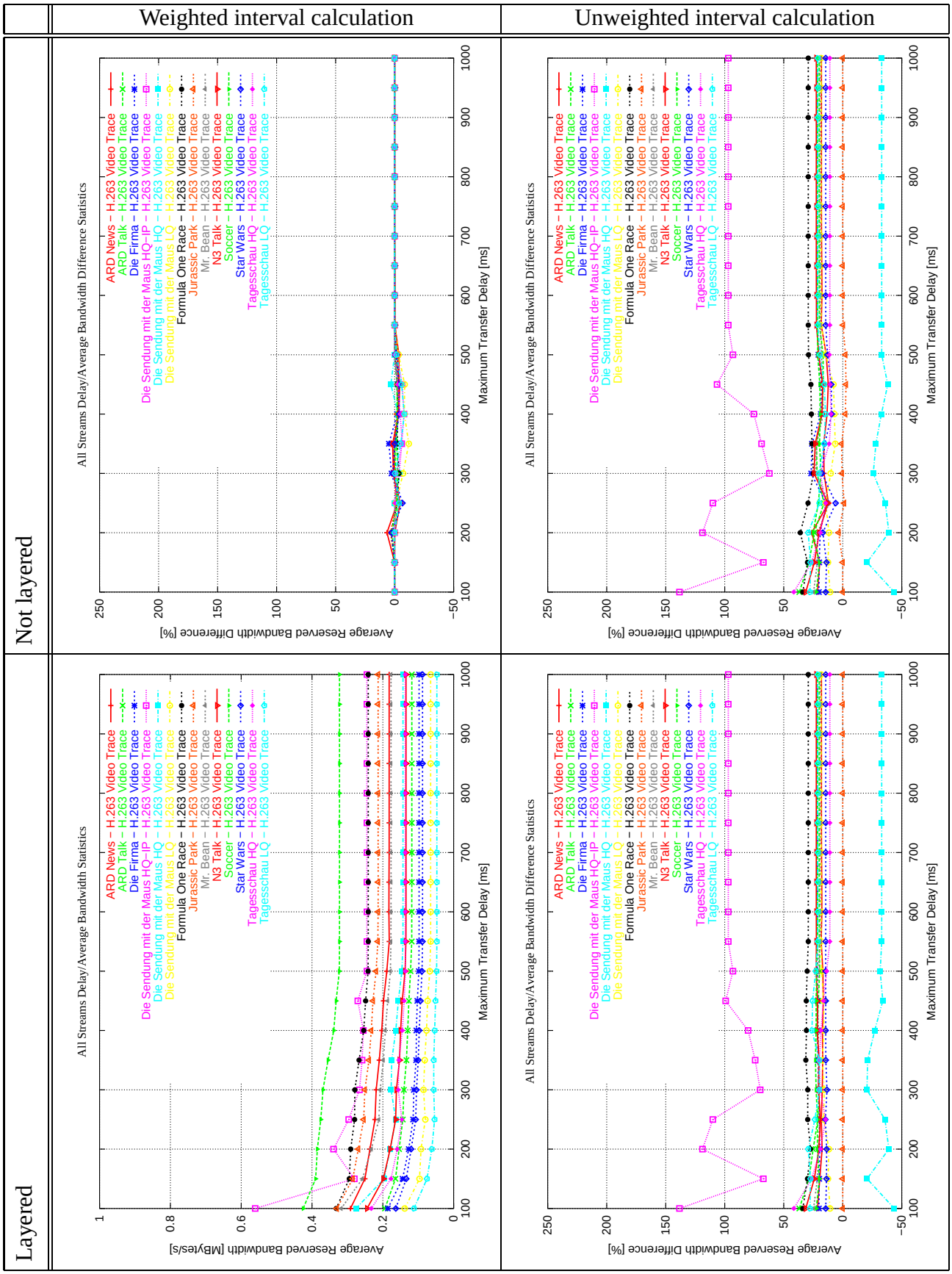


Figure B.8: H.263 bandwidth/delay comparision

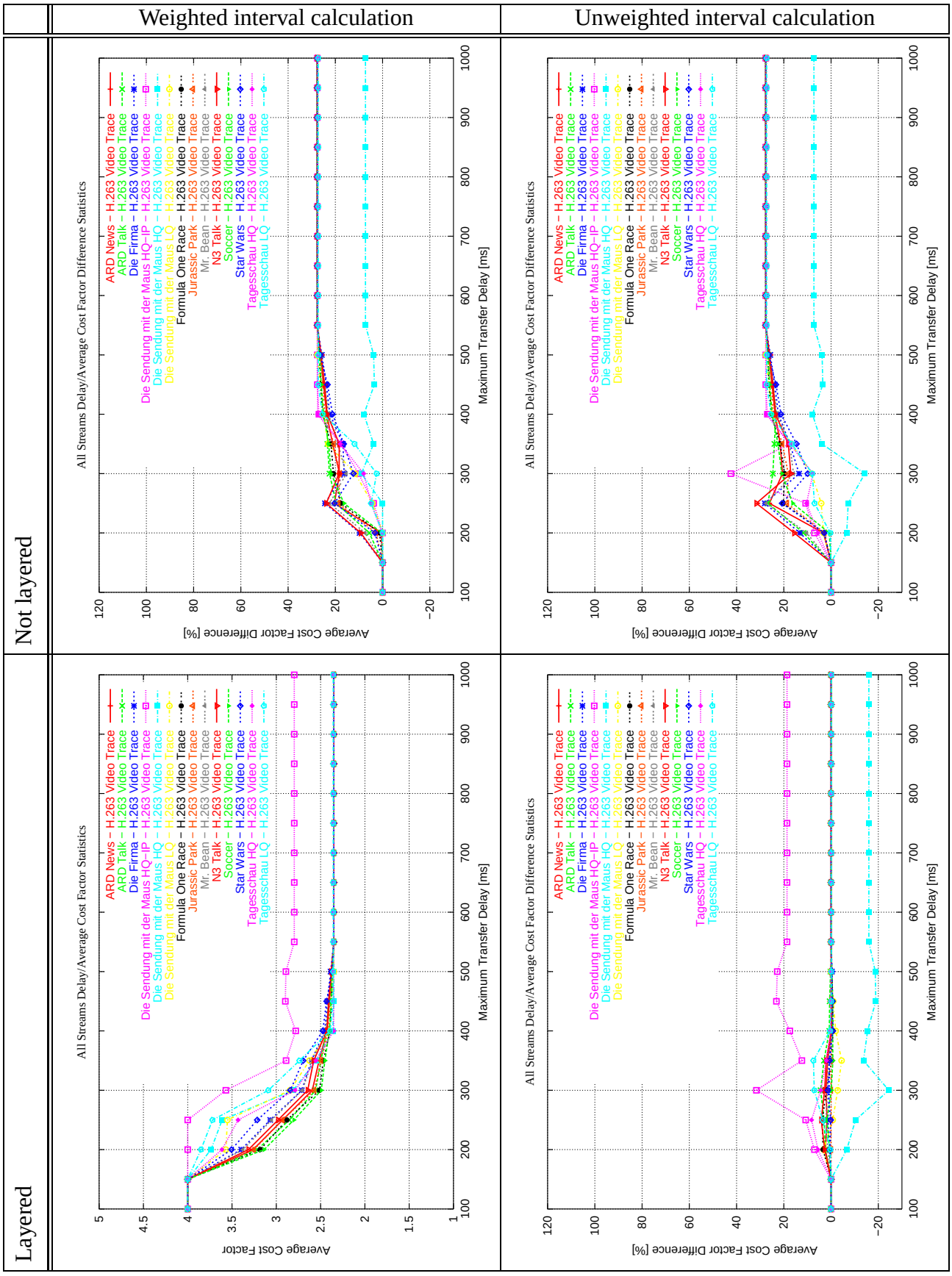


Figure B.9: H.263 average cost factor/delay comparision

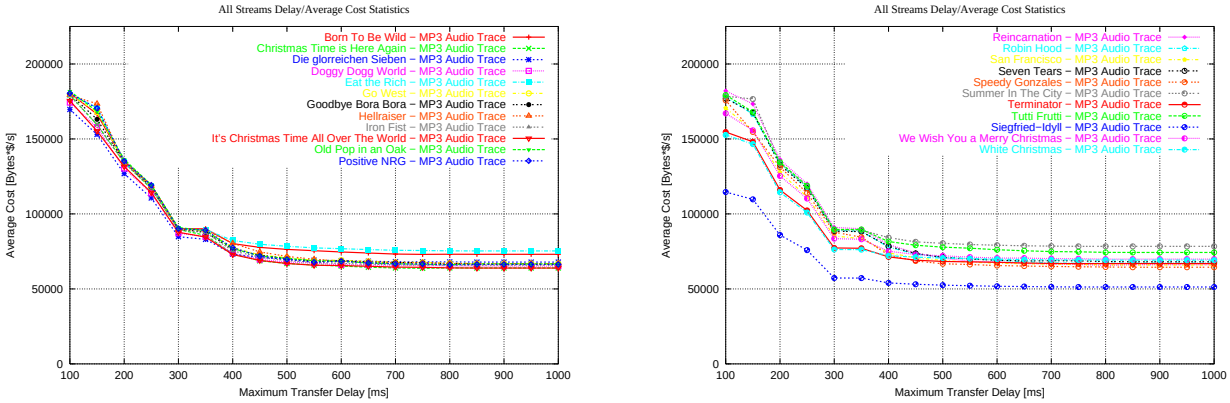


Figure B.10: MP3 cost/delay comparision

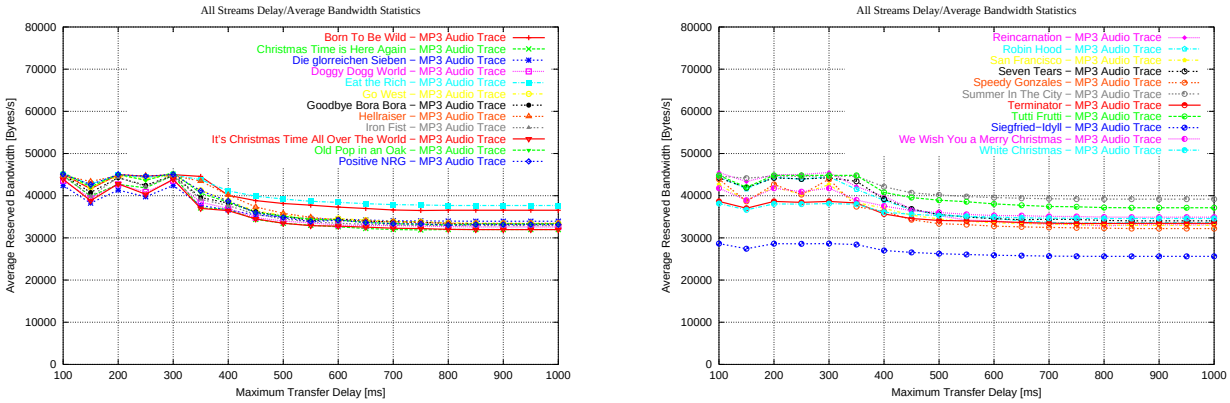


Figure B.11: MP3 bandwidth/delay comparision

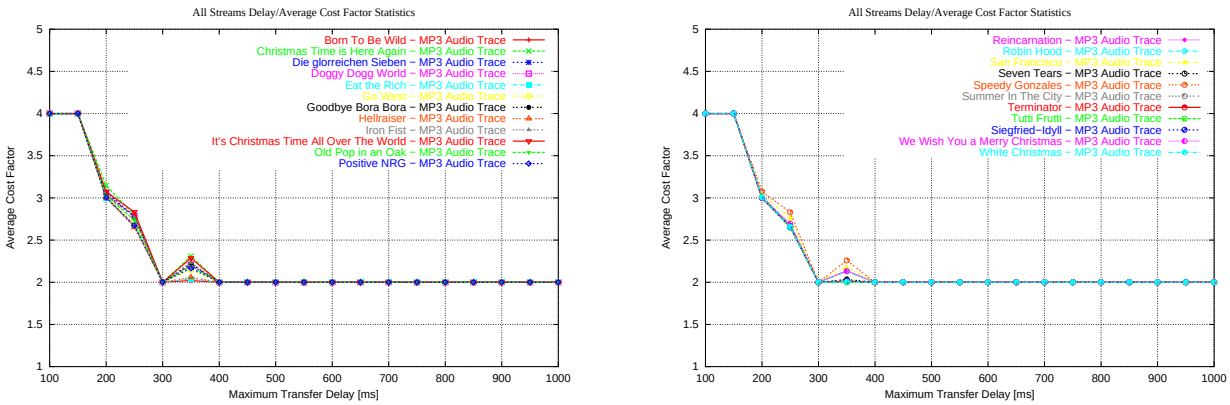


Figure B.12: MP3 average cost factor/delay comparision

Appendix C

Scalability Measurement Results

This appendix contains the scalability simulation results for each trace as explained in section 8.1.4. The average (divided by the simulation duration of 800s) cost, bandwidth and cost factor are shown for 100% utilization. For better comparision, the figures for 25%, 50% and 75% utilization limit show the cost, bandwidth or cost factor reduction compared to the 100% figure in percent.

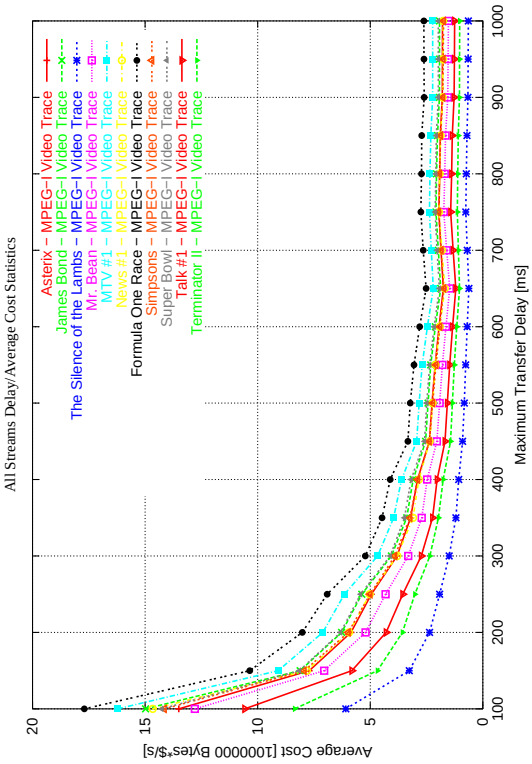


Figure C.1: MPEG-1 cost, original

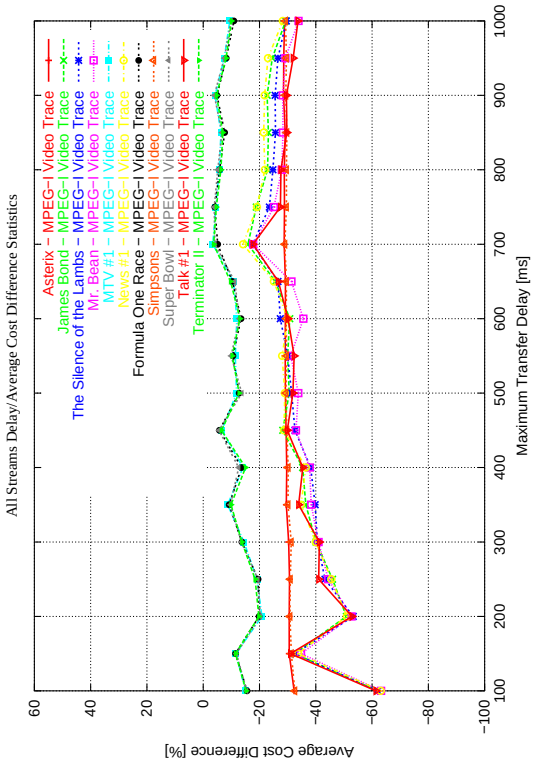


Figure C.2: Cost reduction for 75% util.

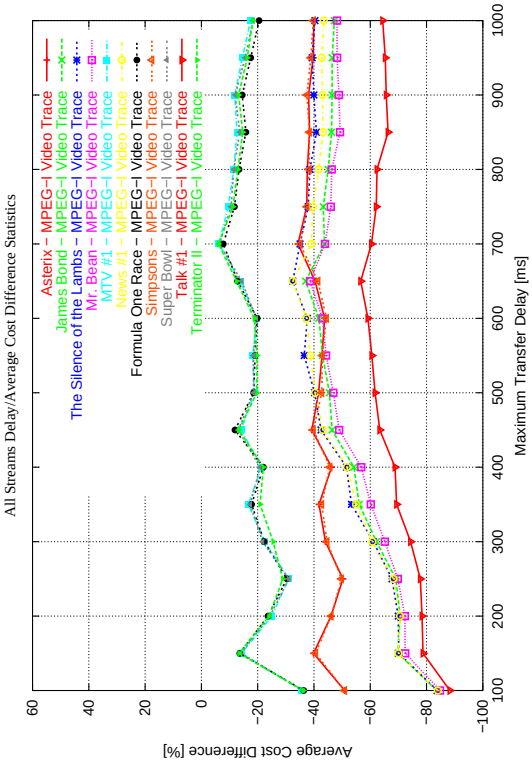


Figure C.3: Cost reduction for 50% util.

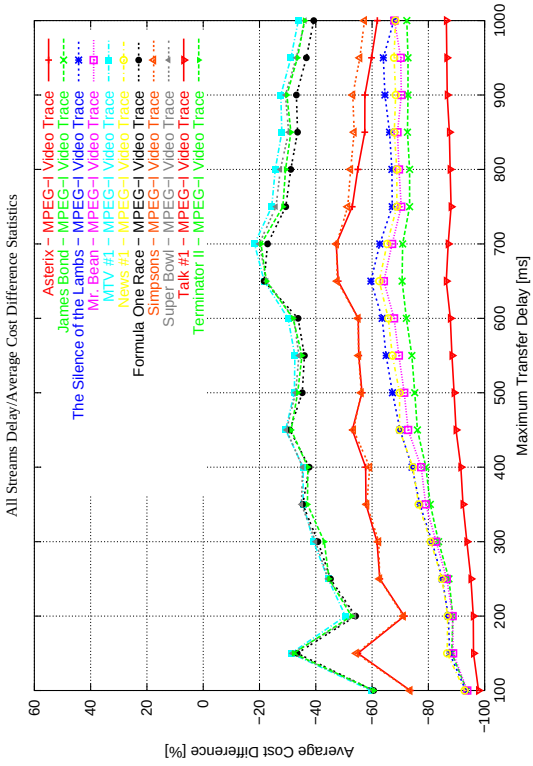


Figure C.4: Cost reduction for 25% util.

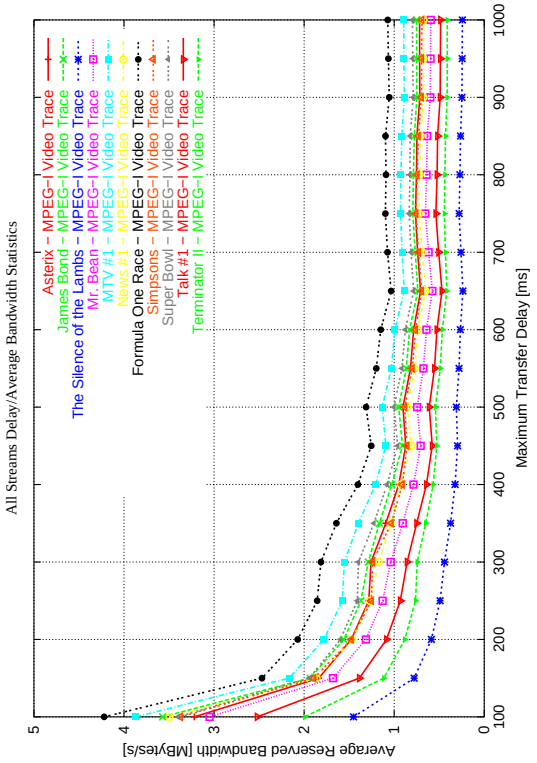


Figure C.5: MPEG-1 bandwidth, original

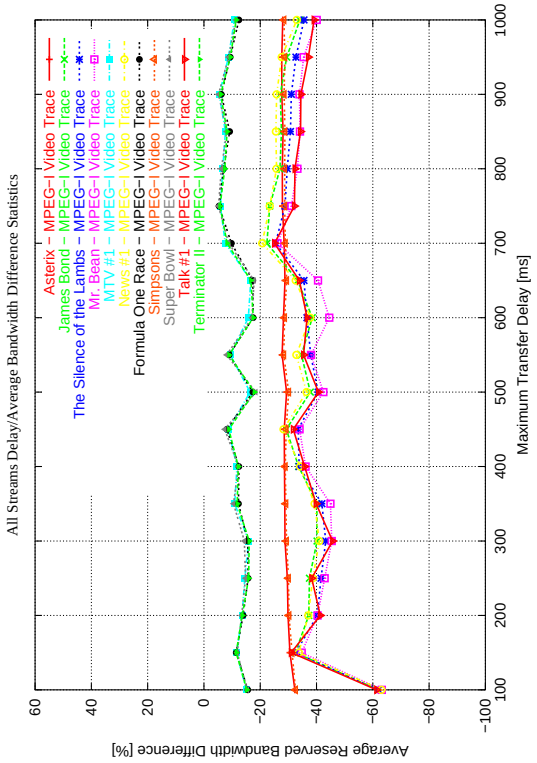


Figure C.6: Bandwidth reduction for 75% util.

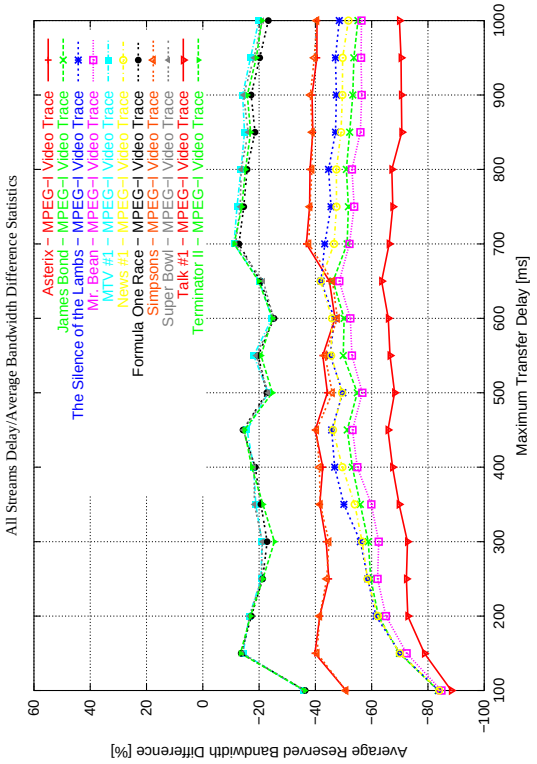


Figure C.7: Bandwidth reduction for 50% util.

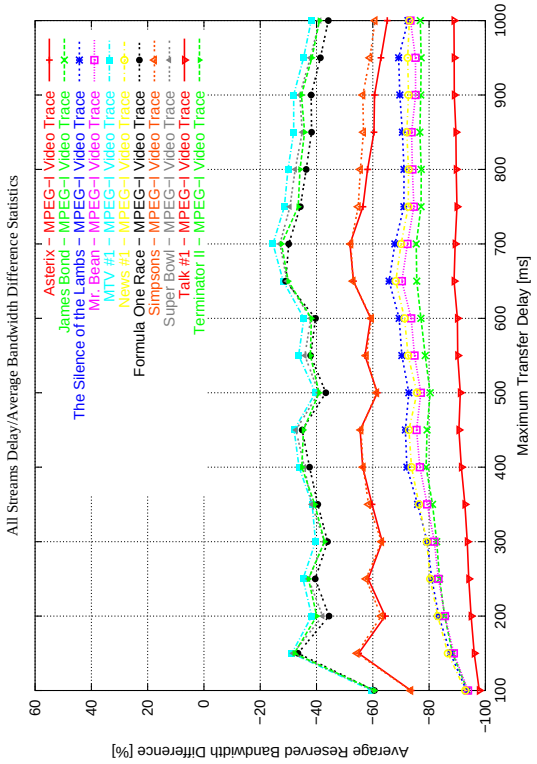


Figure C.8: Bandwidth reduction for 25% util.

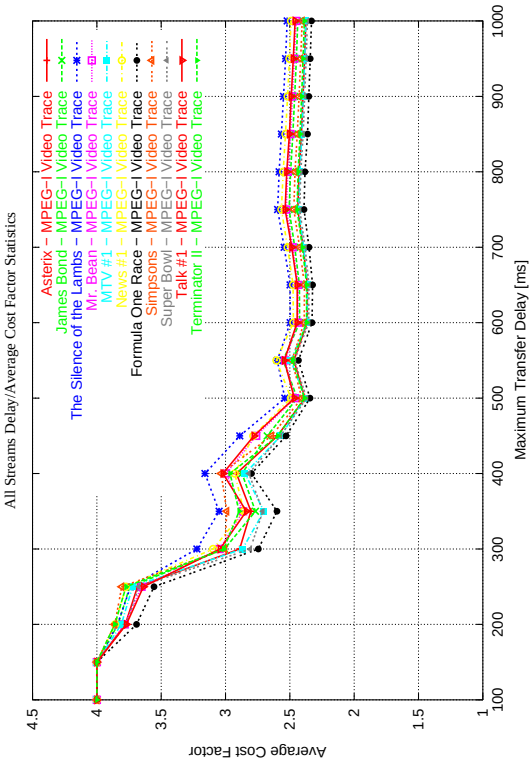


Figure C.9: MPEG-1 cost factor, original

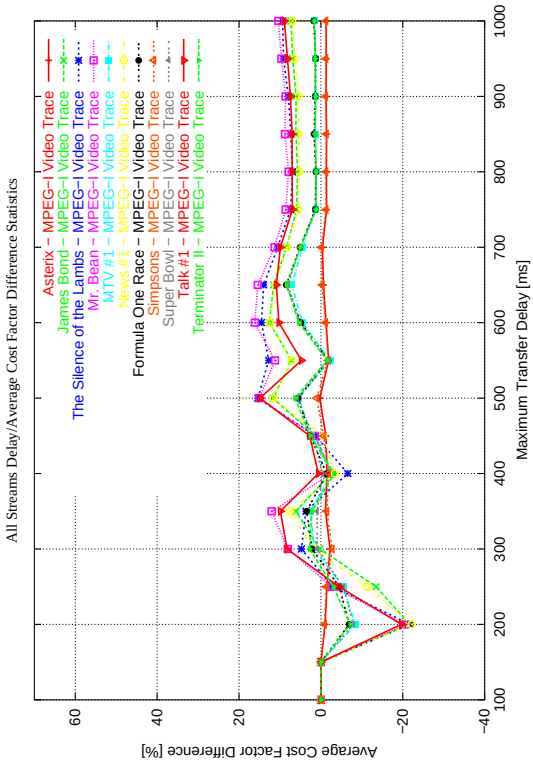


Figure C.10: Cost factor reduction for 75% util.

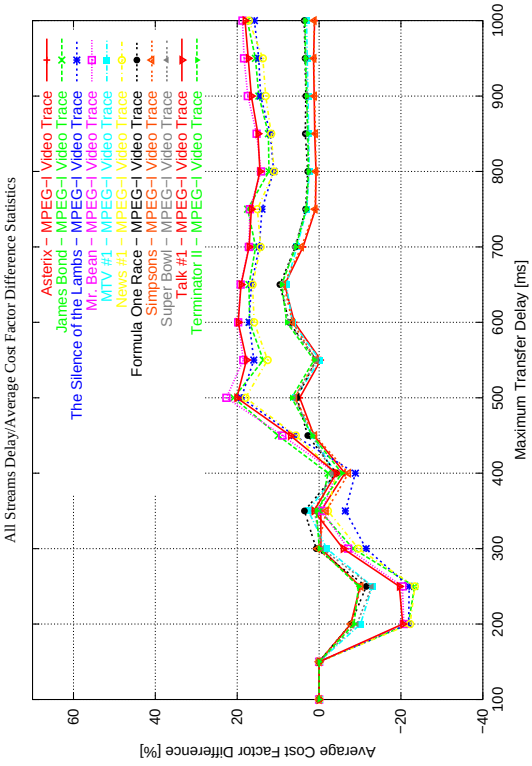


Figure C.11: Cost factor reduction for 50% util.

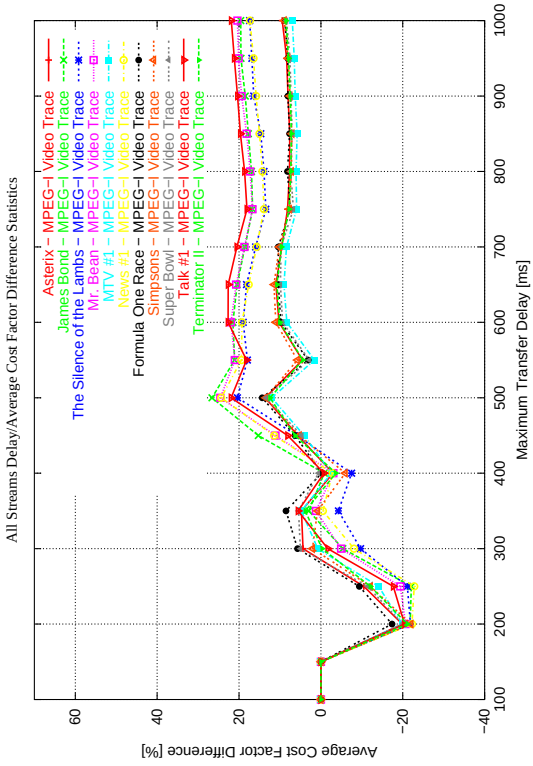


Figure C.12: Cost factor reduction for 25% util.

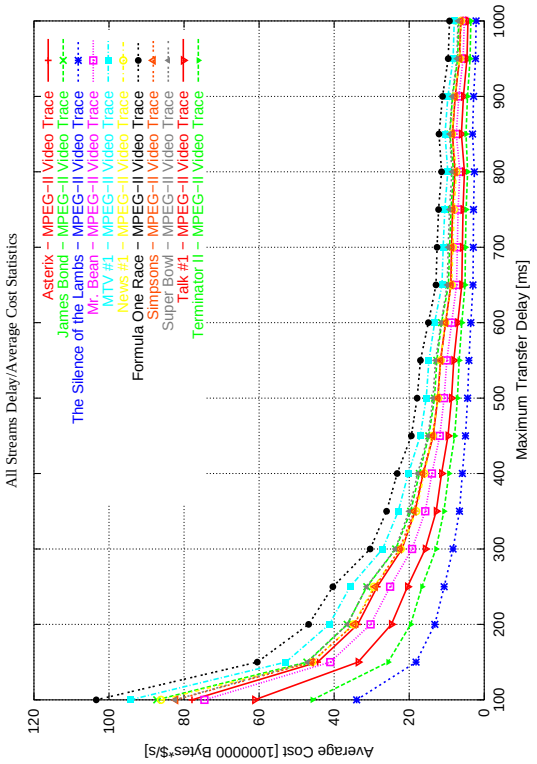


Figure C.13: MPEG-2 cost, original

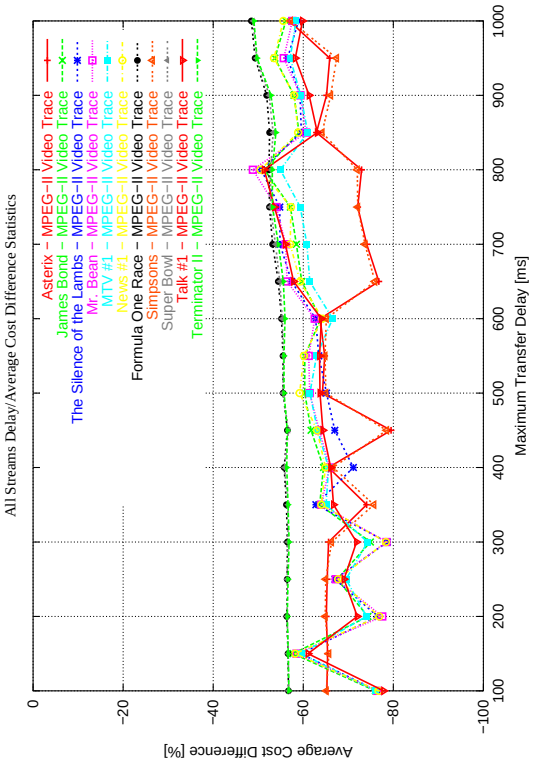


Figure C.14: Cost reduction for 75% util.

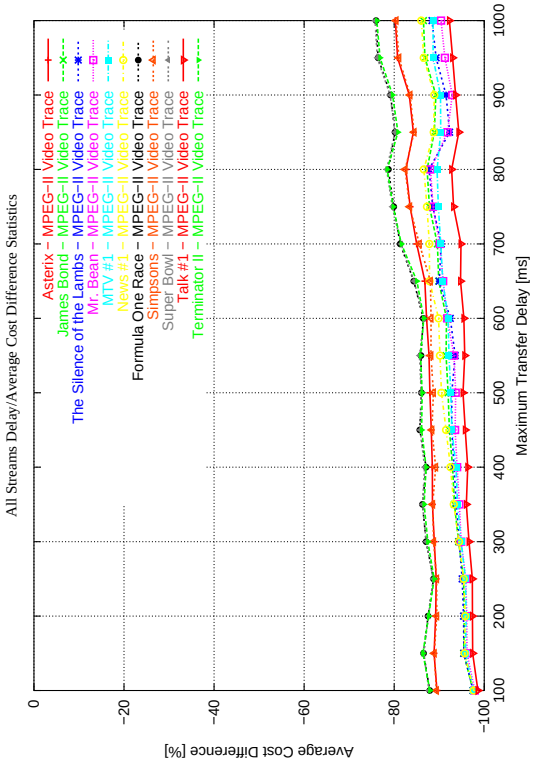


Figure C.15: Cost reduction for 50% util.

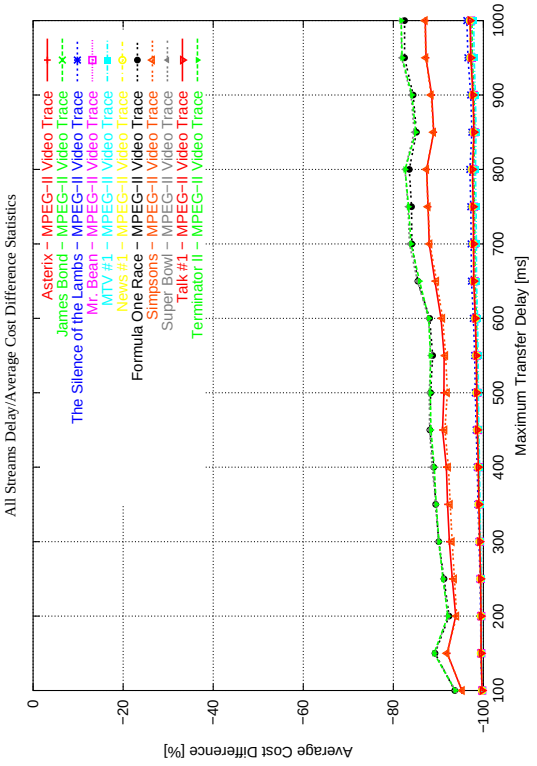


Figure C.16: Cost reduction for 25% util.

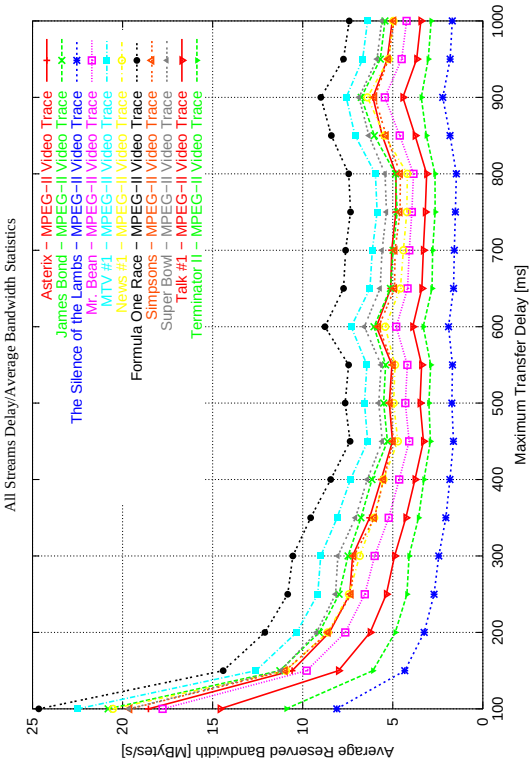


Figure C.17: MPEG-2 bandwidth, original

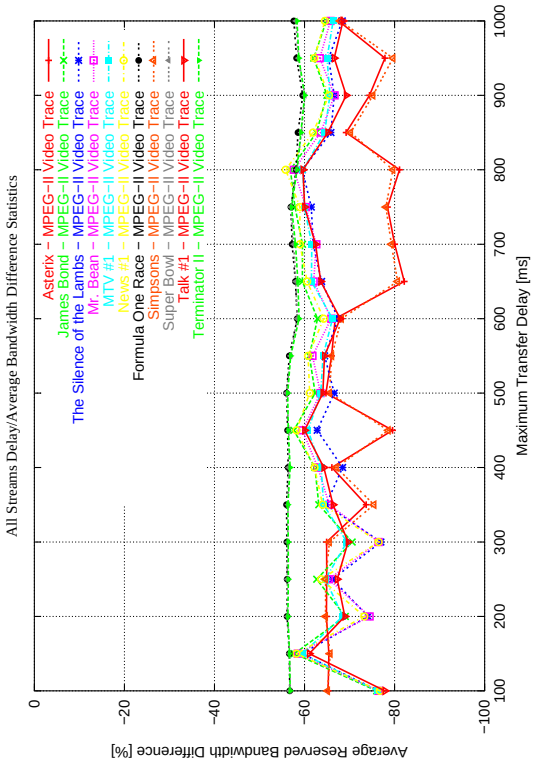


Figure C.18: Bandwidth reduction for 75% util.

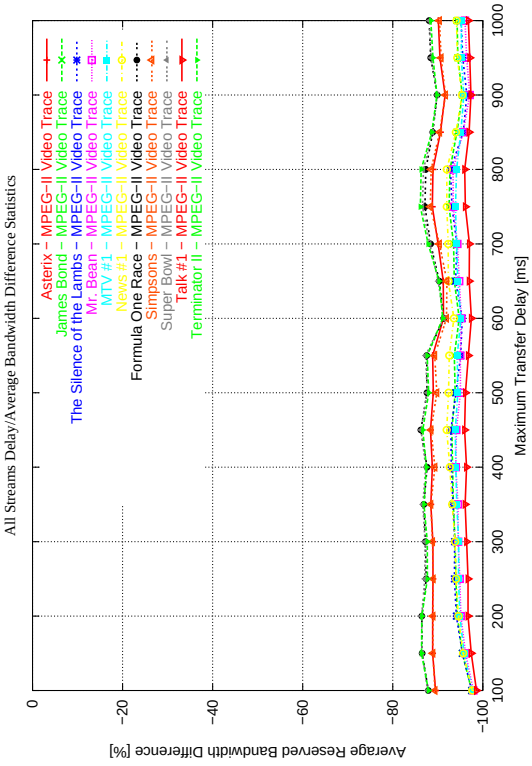


Figure C.19: Bandwidth reduction for 50% util.

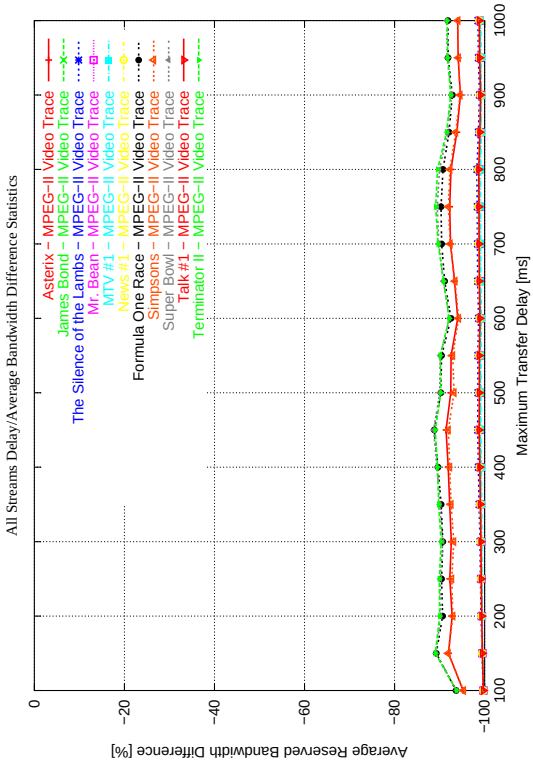


Figure C.20: Bandwidth reduction for 25% util.

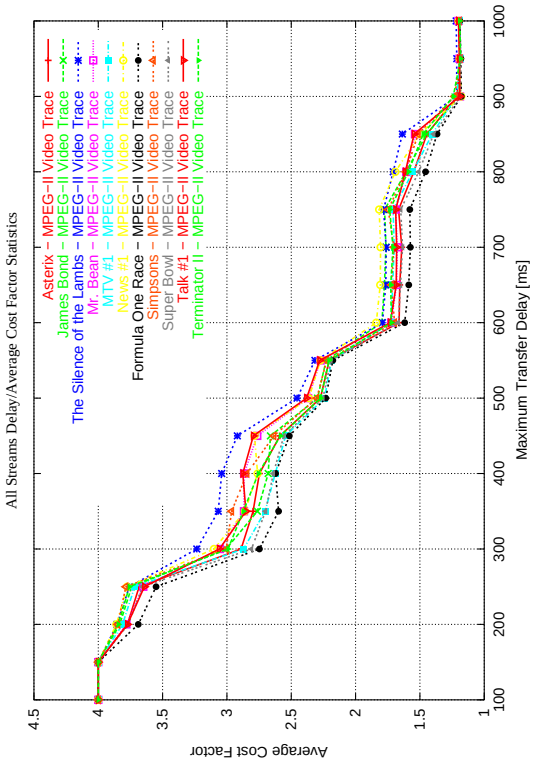


Figure C.21: MPEG-2 cost factor, original

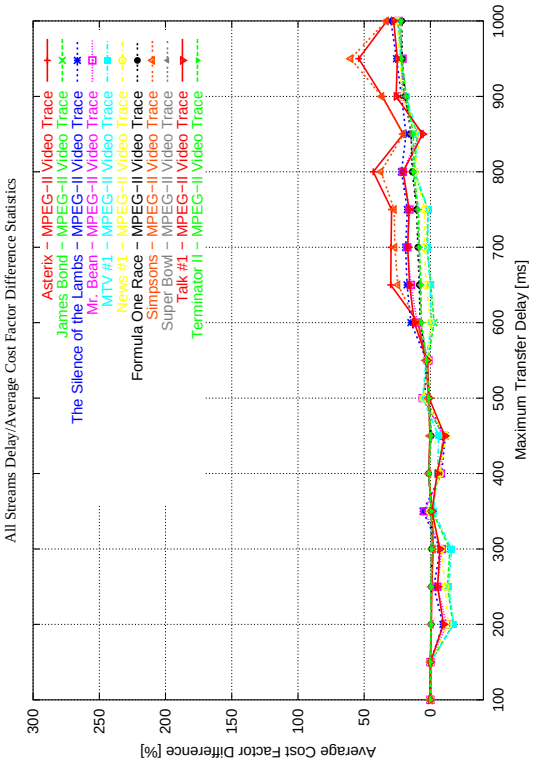


Figure C.22: Cost factor reduction for 75% util.

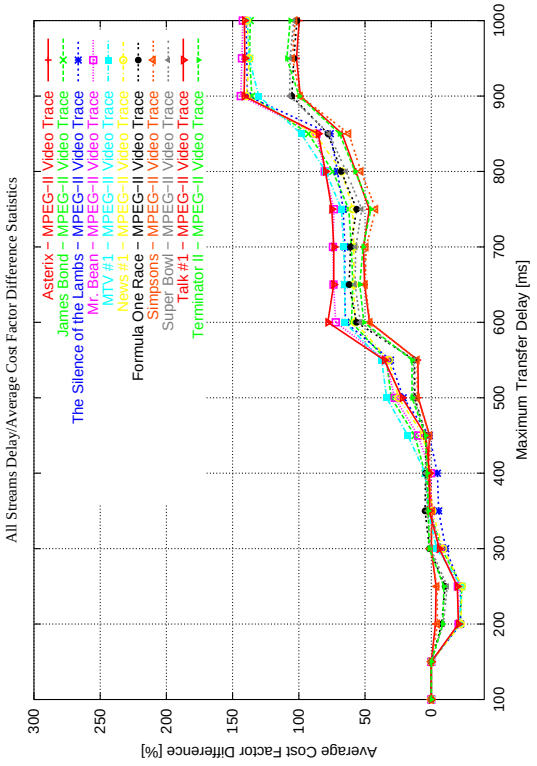


Figure C.23: Cost factor reduction for 50% util.

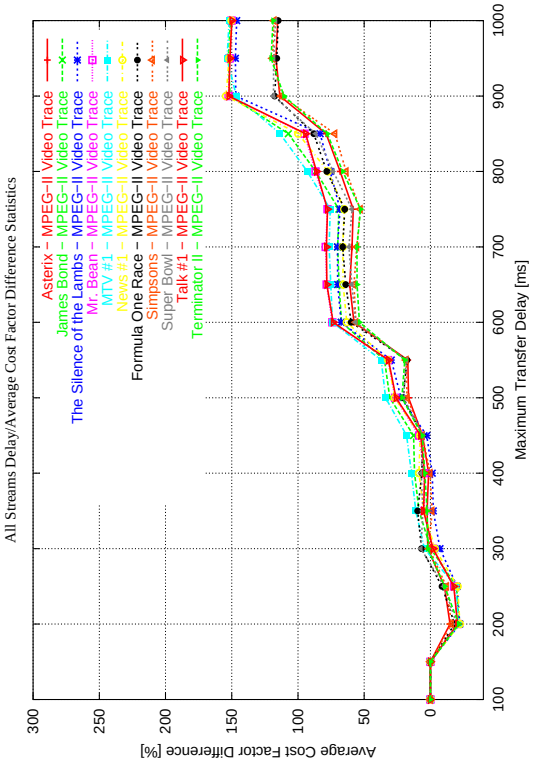


Figure C.24: Cost factor reduction for 25% util.

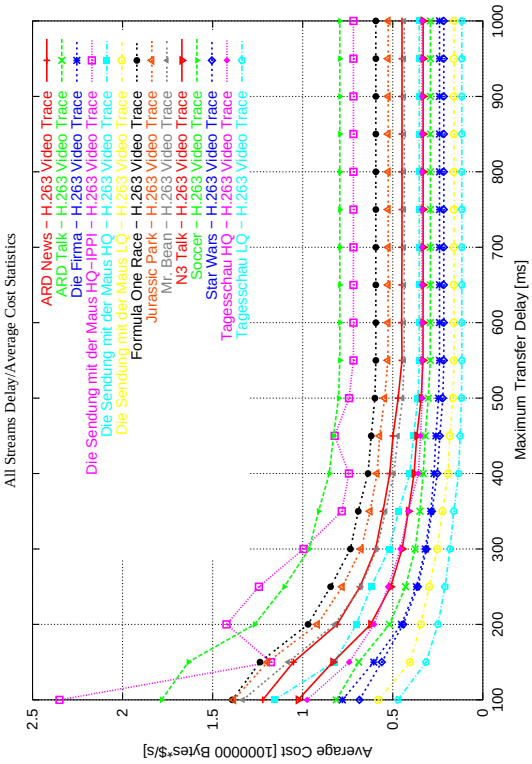


Figure C.25: H.263 cost, original

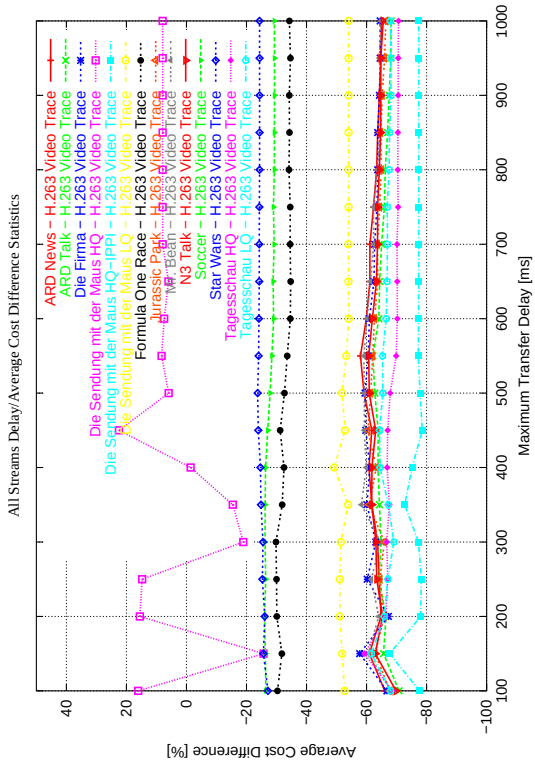


Figure C.26: Cost reduction for 75% util.

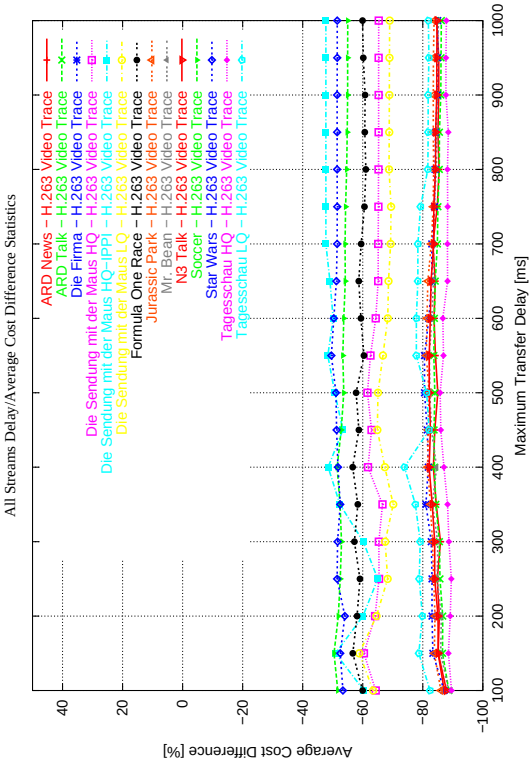


Figure C.27: Cost reduction for 50% util.

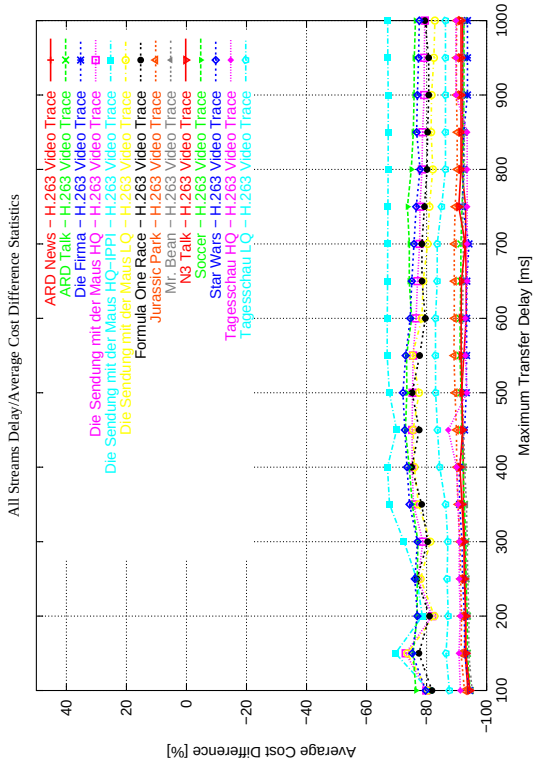


Figure C.28: Cost reduction for 25% util.

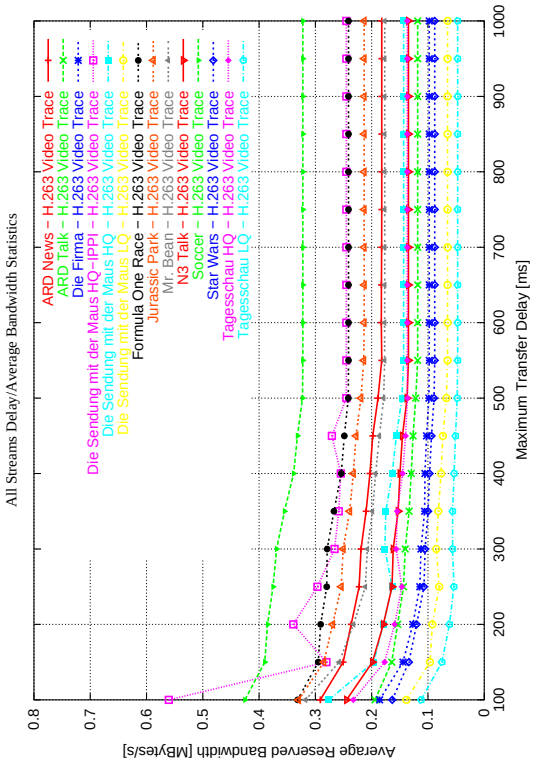


Figure C.29: H.263 bandwidth, original

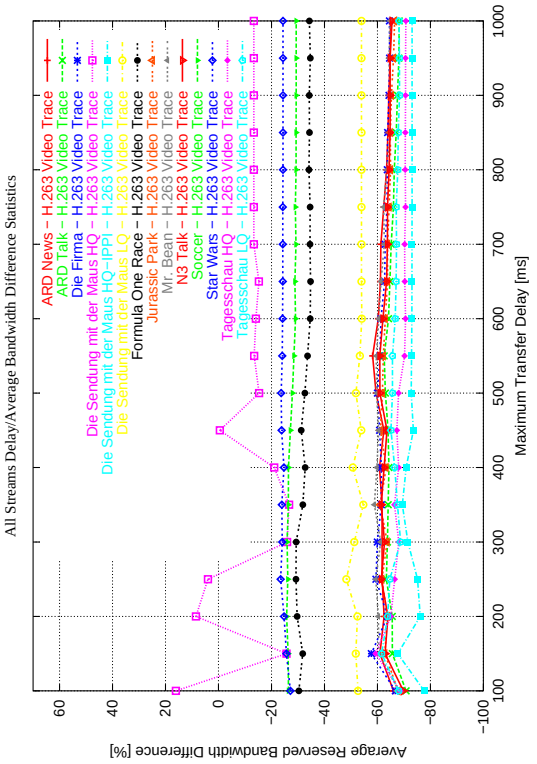


Figure C.30: Bandwidth reduction for 75% util.

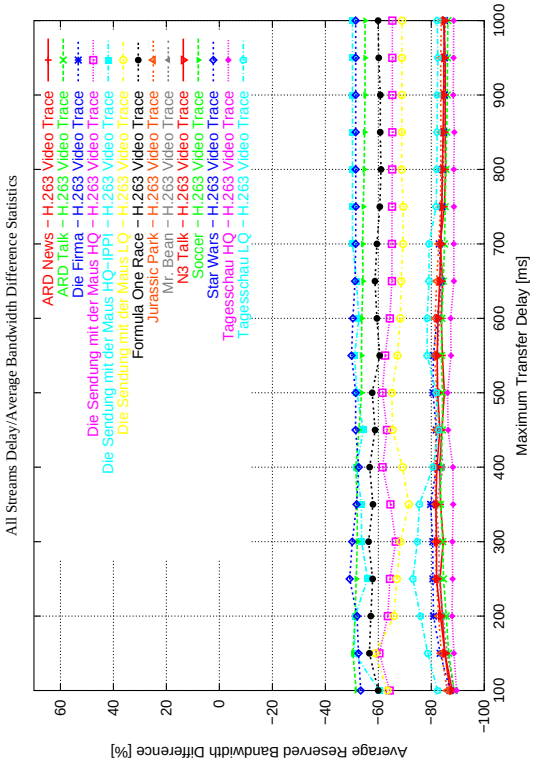


Figure C.31: Bandwidth reduction for 50% util.

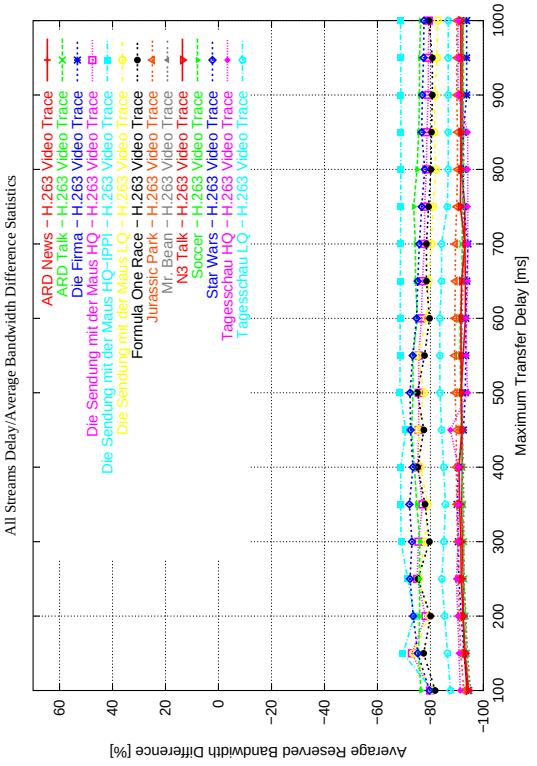


Figure C.32: Bandwidth reduction for 25% util.

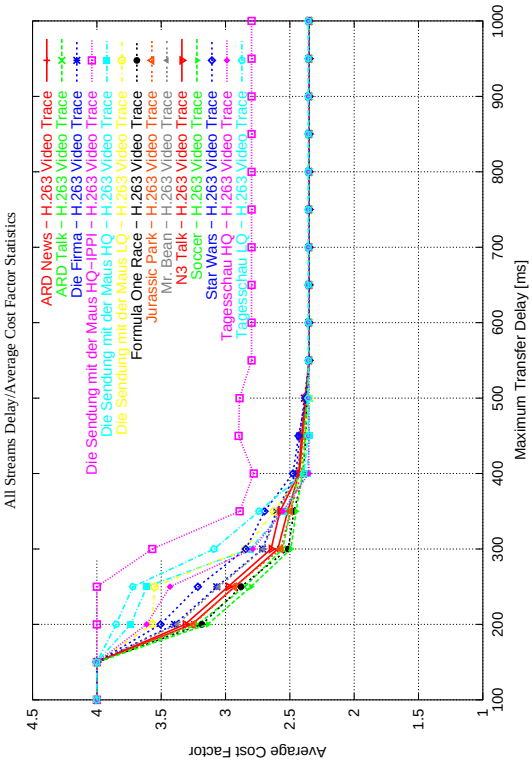


Figure C.33: H.263 cost factor, original

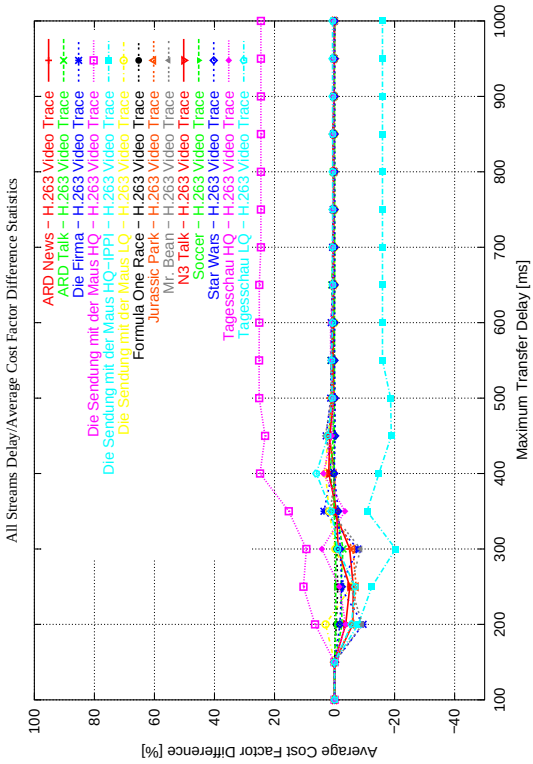


Figure C.34: Cost factor reduction for 75% util.

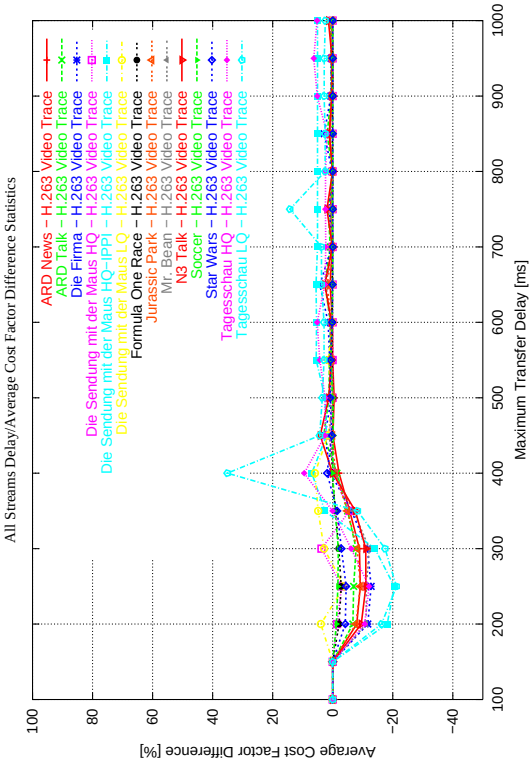


Figure C.35: Cost factor reduction for 50% util.

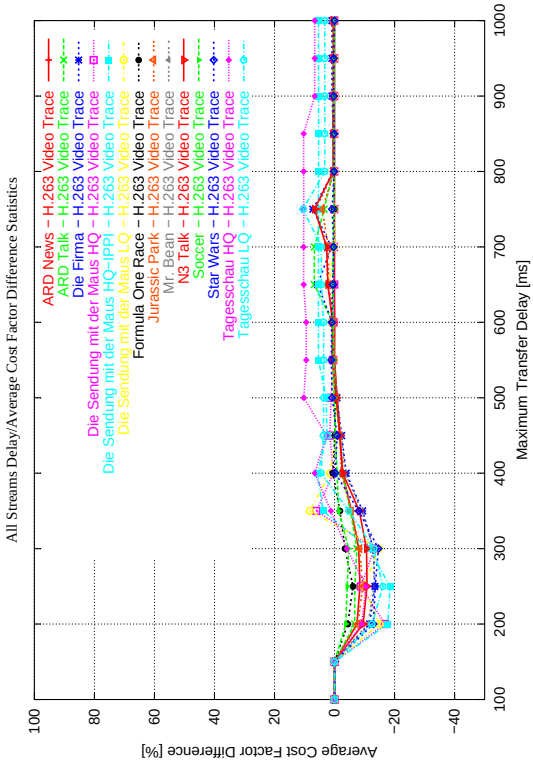


Figure C.36: Cost factor reduction for 25% util.

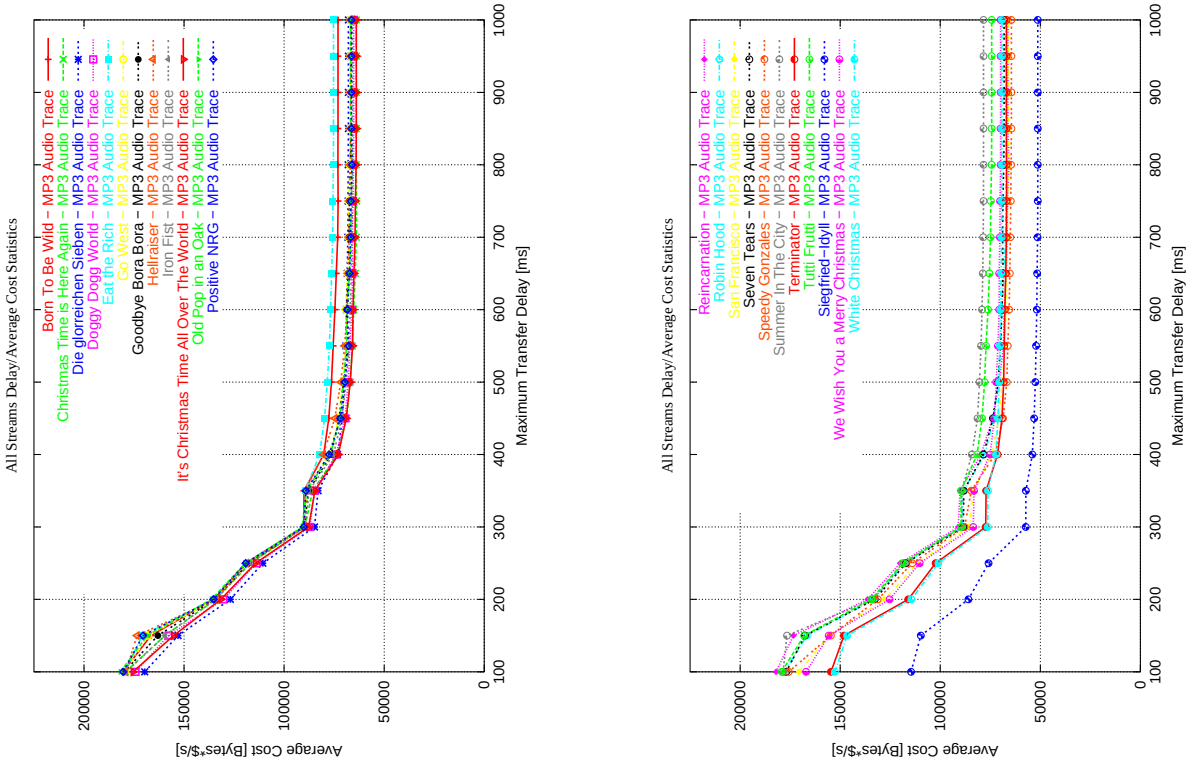


Figure C.37: MP3 cost, original

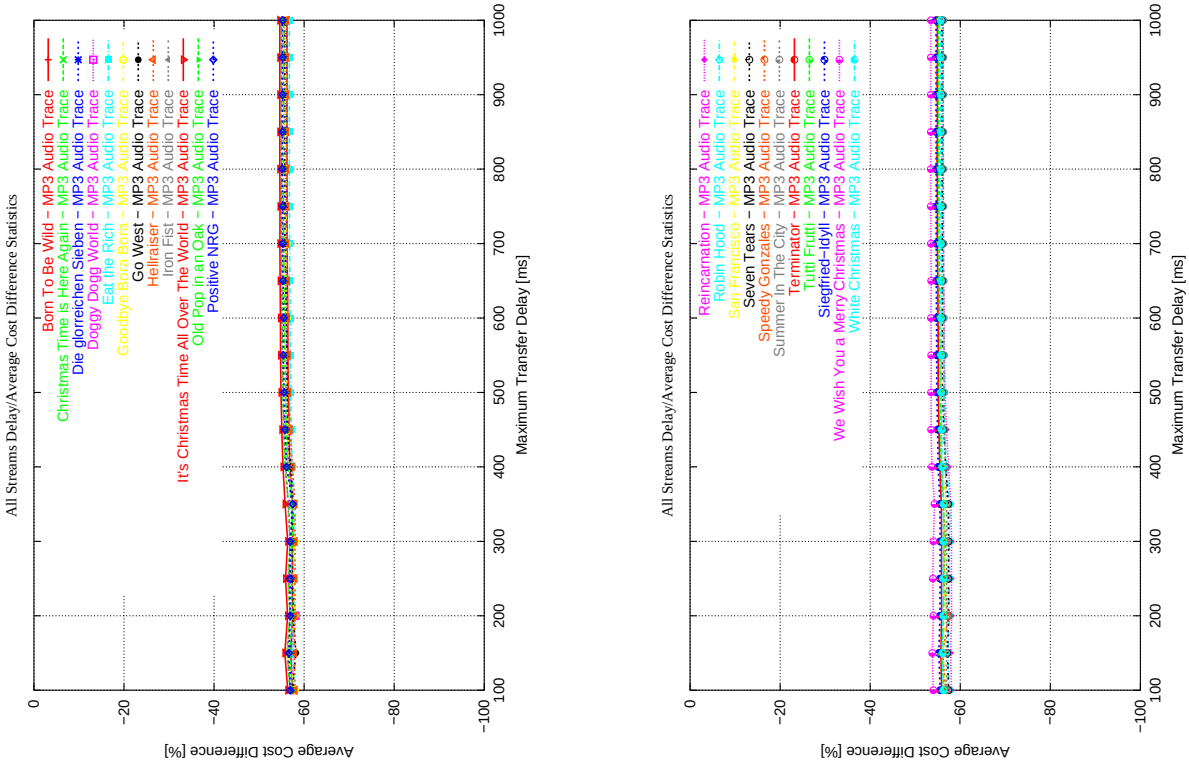


Figure C.38: Cost reduction for 75% utilization

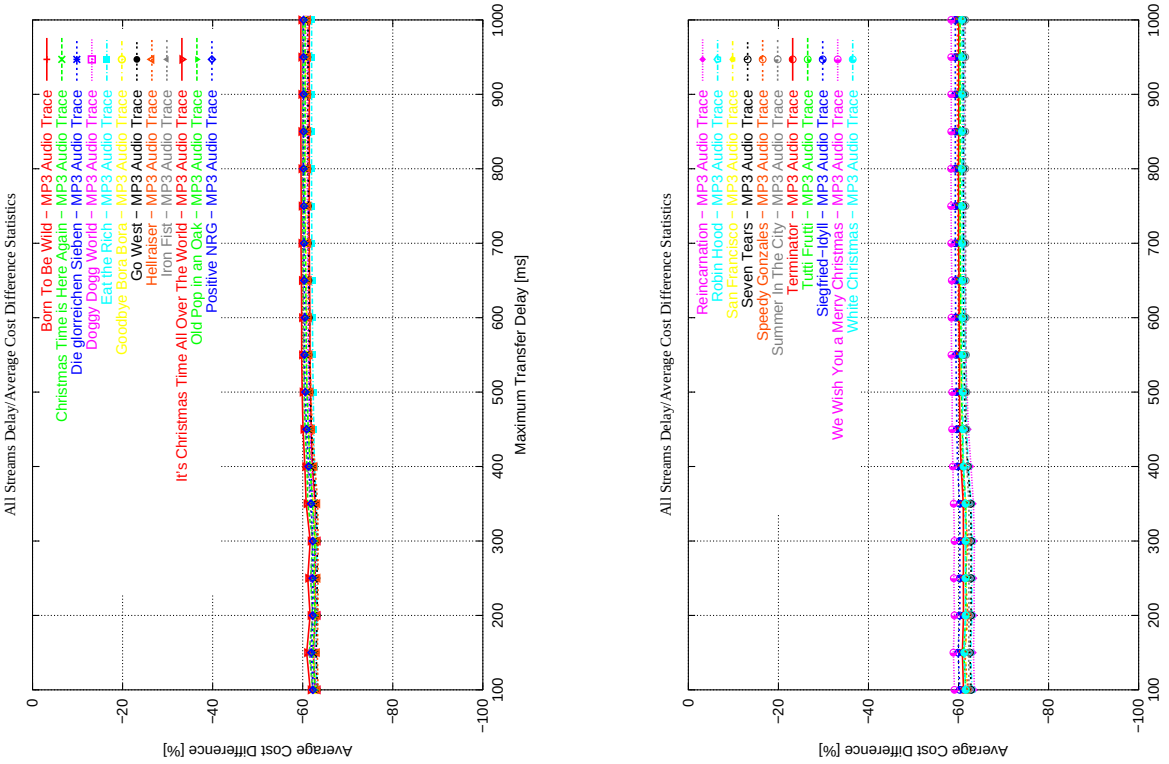


Figure C.39: Cost reduction for 50% utilization

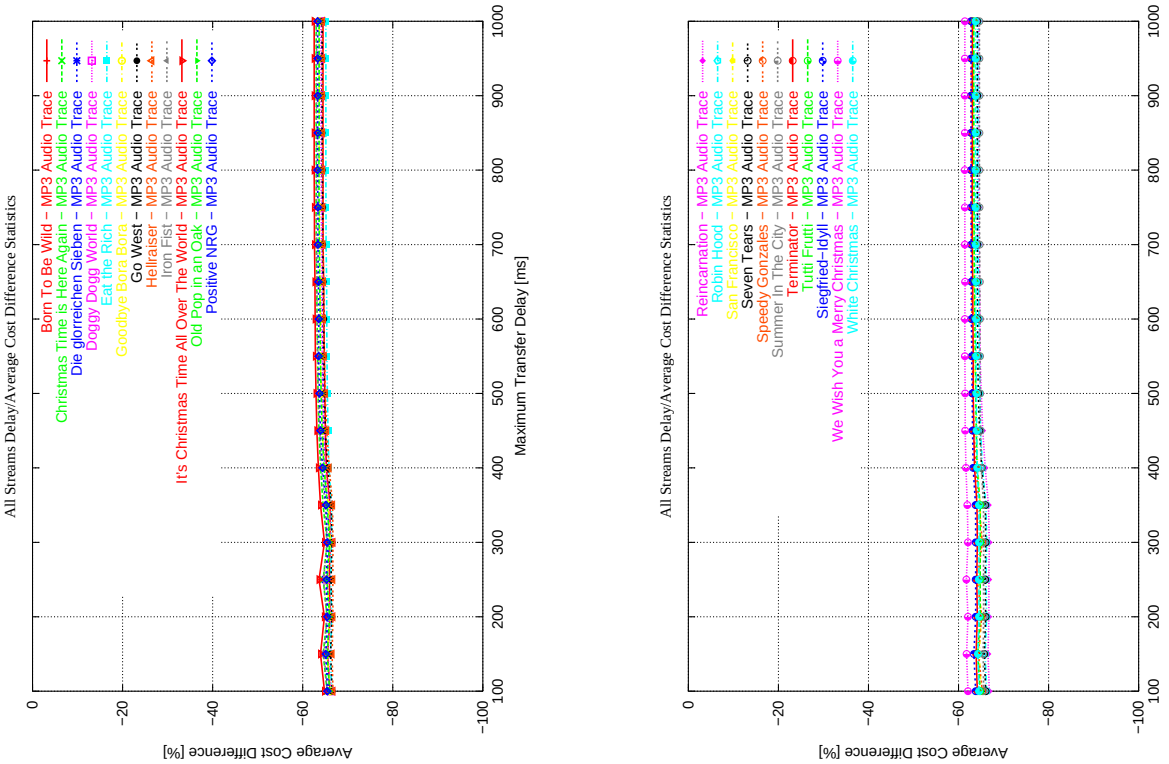


Figure C.40: Cost reduction for 25% utilization

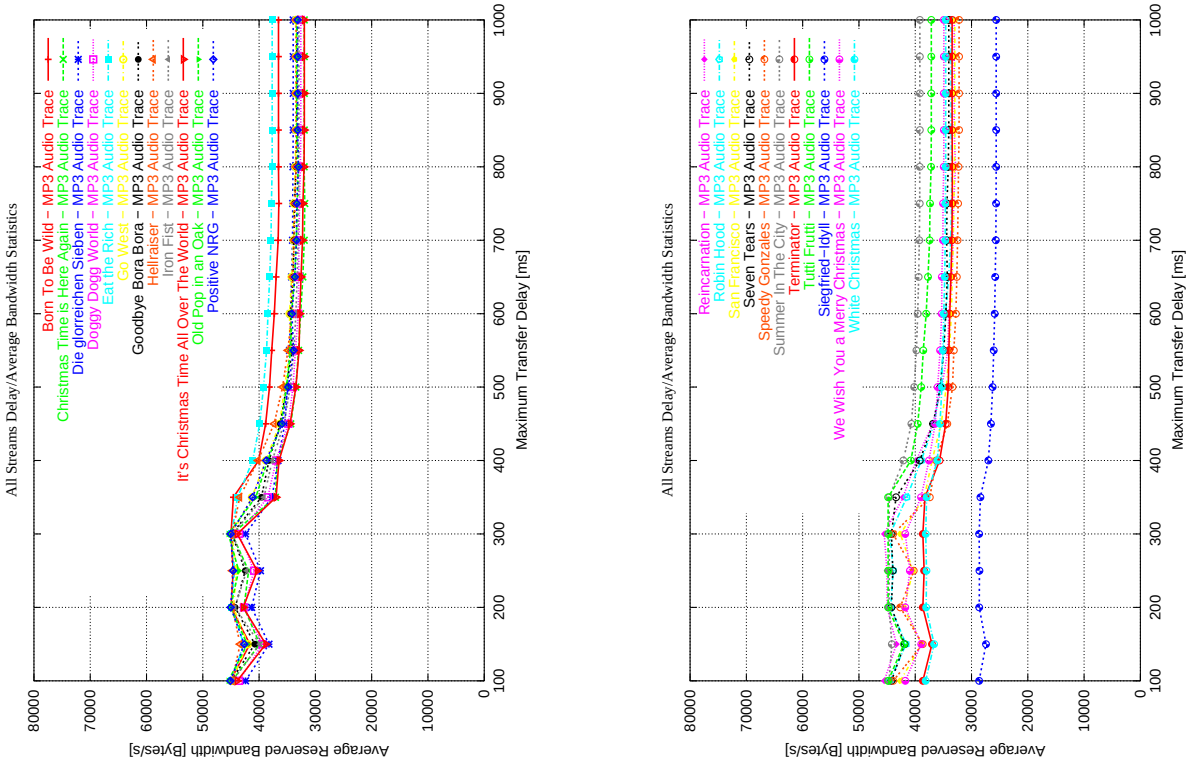


Figure C.41: MP3 bandwidth, original

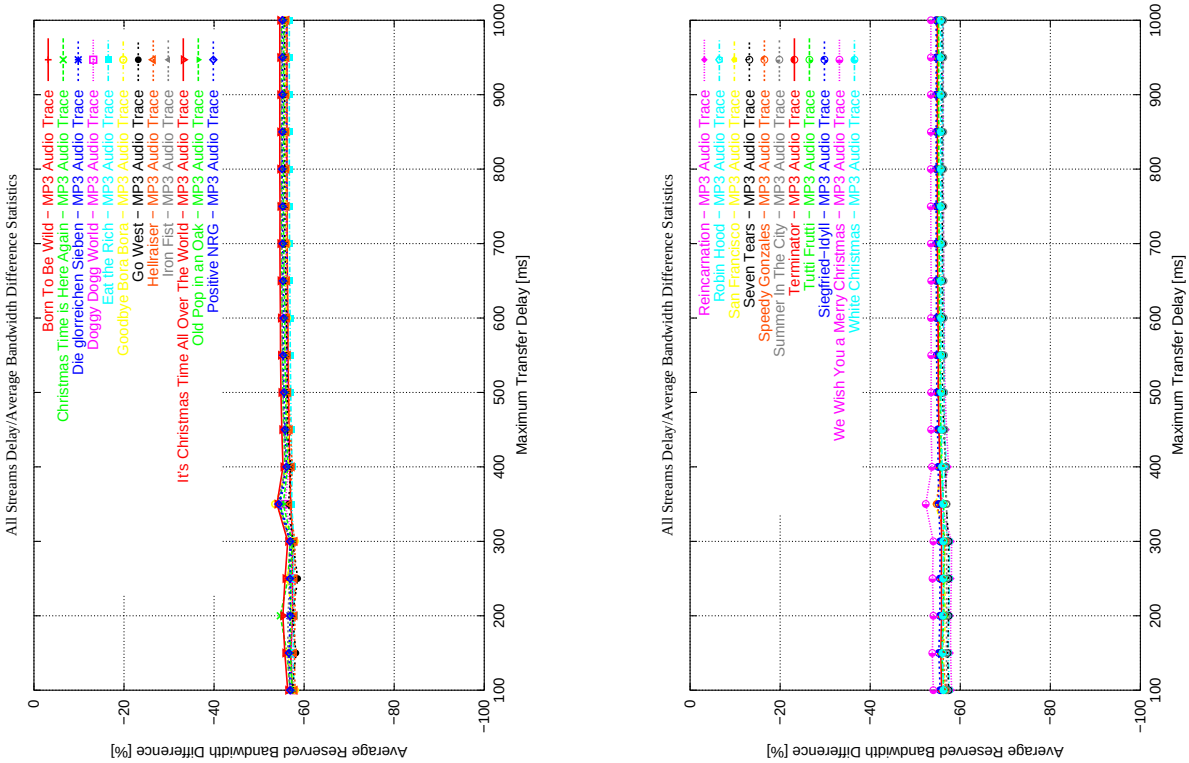


Figure C.42: Bandwidth reduction for 75% utilization

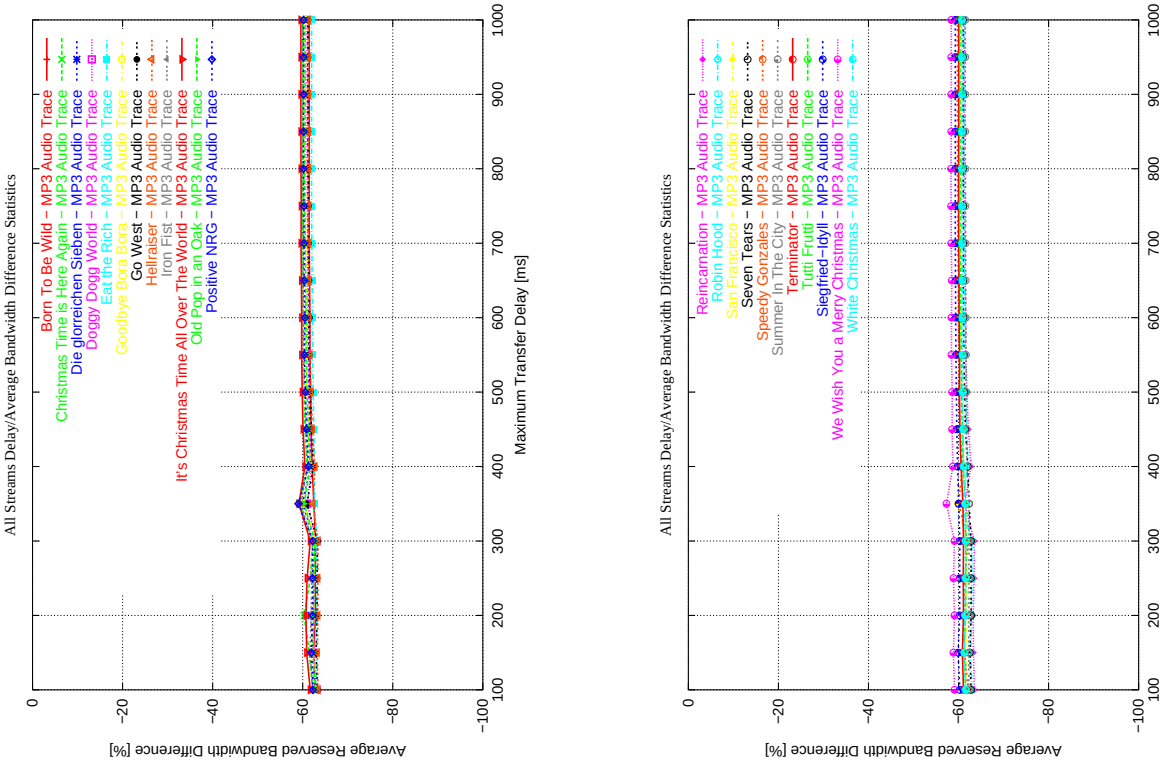


Figure C.43: Bandwidth reduction for 50% utilization

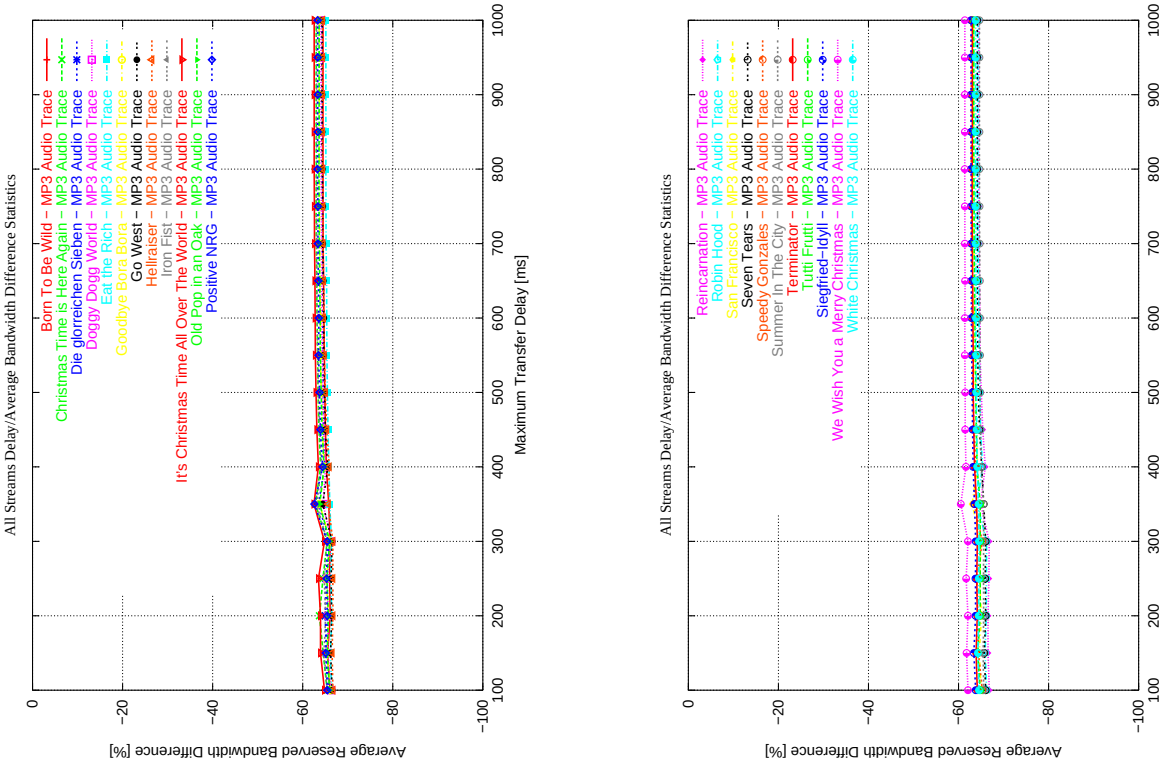


Figure C.44: Bandwidth reduction for 25% utilization

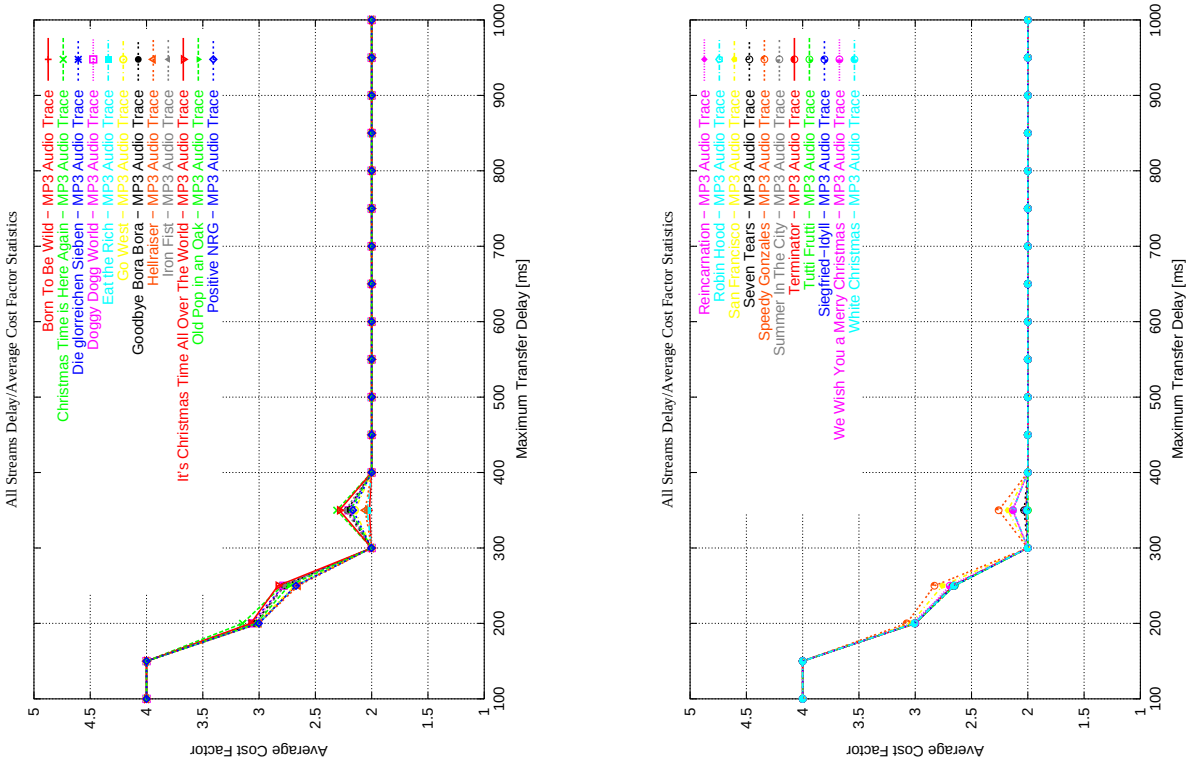


Figure C.45: MP3 cost factor, original

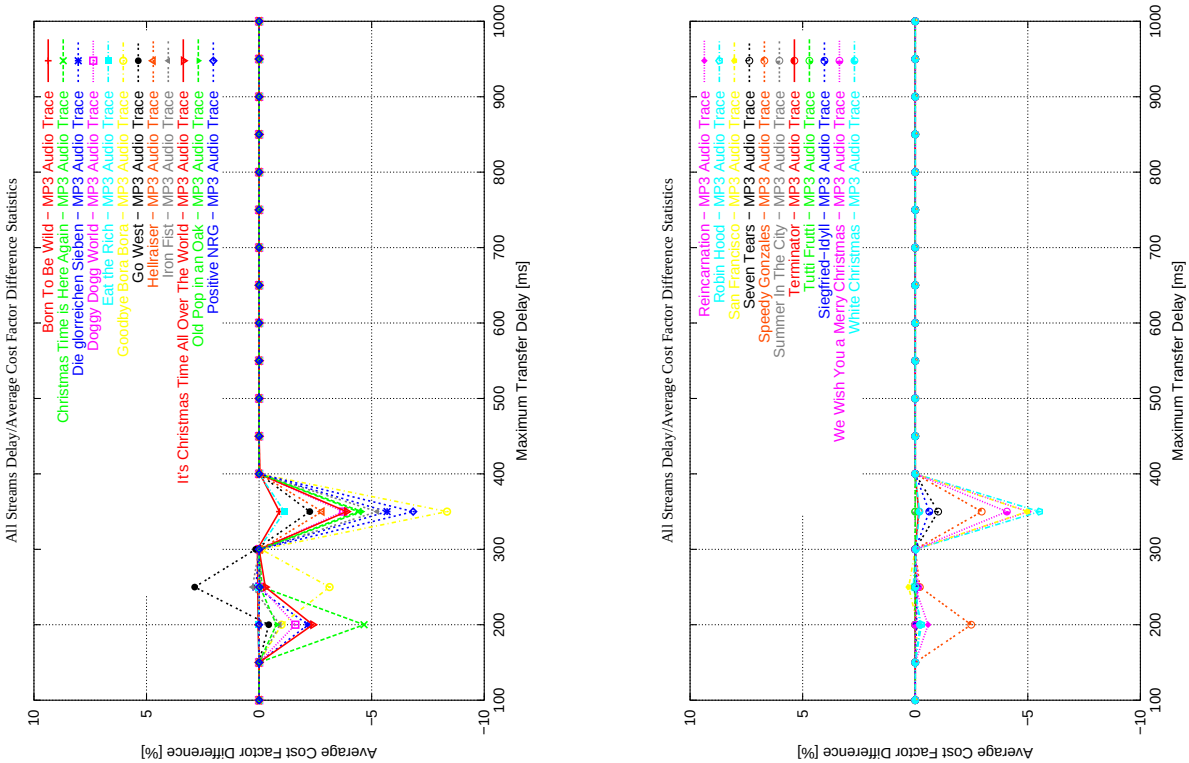


Figure C.46: Cost factor reduction for 75% utilization

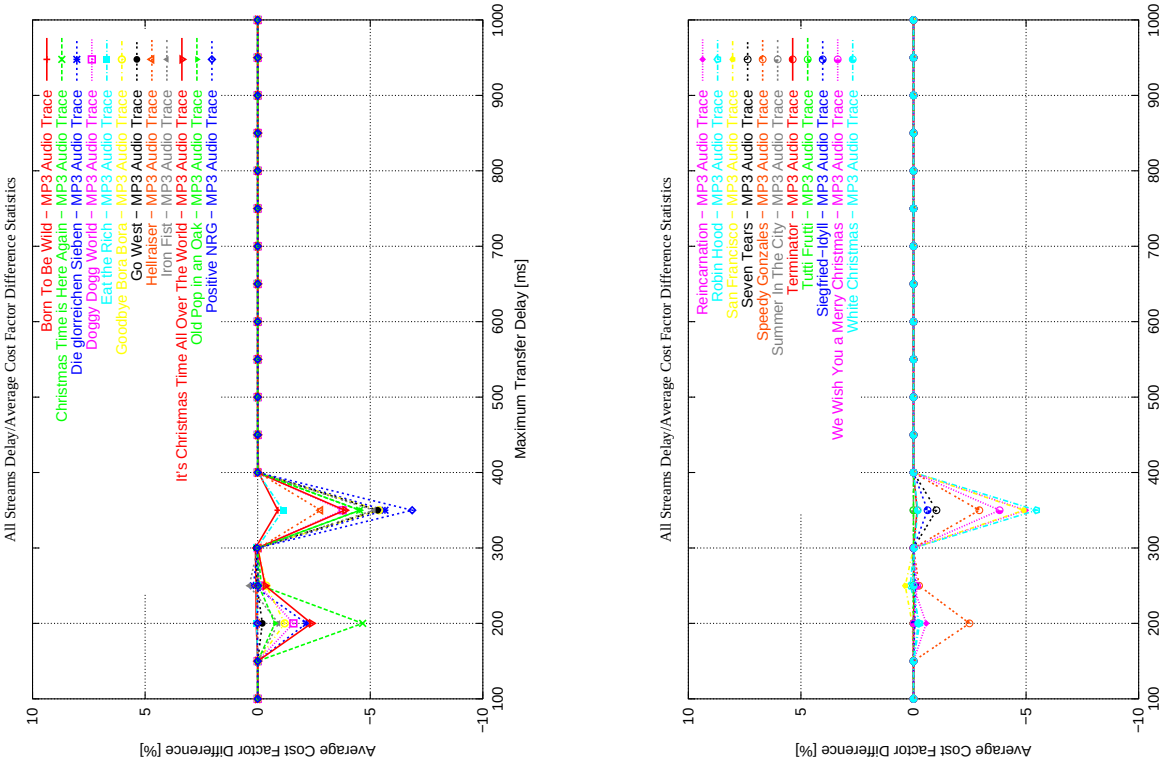


Figure C.47: Cost factor reduction for 50% utilization

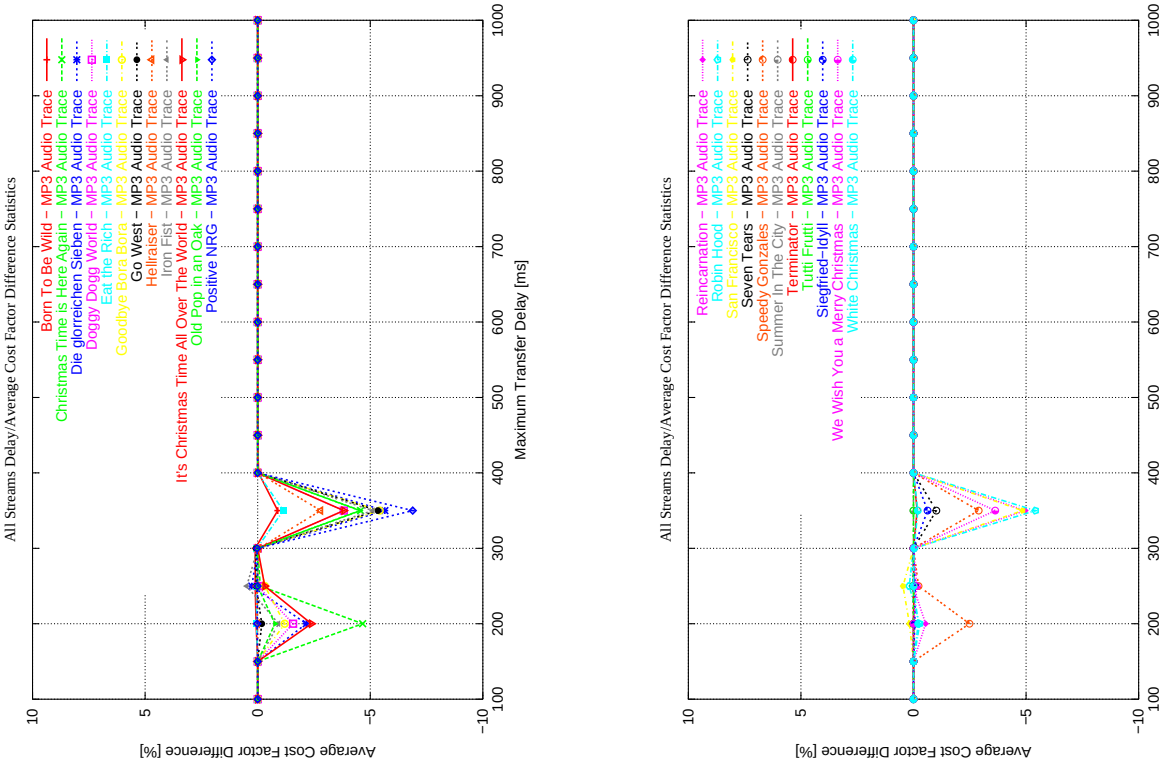


Figure C.48: Cost factor reduction for 25% utilization

Appendix D

Abbrevations Index

4CIF 4× CIF resolution: 704×576 pixels

16CIF 16× CIF resolution: 1408×1152 pixels

AF Assured Forwarding (see section 2.2.3)

AoD Audio on Demand (see chaper 2)

ASRMD1 Apporoximation algorithm #1 for the SRMD problem (see section 2.5.4 and [LS98])

ATM Asynchronous Transfer Mode

BB Bandwidth Broker (see section 3.1 and [Sel01])

BE Best Effort (see section 2.2.3)

CBR Constant Bit-Rate

CD Compact Disc

CDDA Compact Disc Digital Audio

CIF Common Intermediate Format (352×288 pixels)

CORAL COmmunication Protocols for Real-Time Access to Digital Libraries (see section 3.1)

CPU Central Processing Unit

CSRC Contributing SouRCe (see section 2.1.5)

DCT Discrete Cosine Transformation (see section 2.6.1)

DFT Discrete Fourier Transformation

DSCP DiffServ Code Point (see section 2.2.3)

DiffServ Differentiated Services (see section 2.2.3)

DVQ Digital Video Quality, video quality metric (see section 2.6.1).

ECM Endpoint Congestion Management (see section 3.1 and [Kar01])

EF Expedited Forwarding (see section 2.2.3)

FDDI Fibre Distributed Data Interface (see [Tan96])

FTP File Transfer Protocol

GoP Group of Pictures (see section 2.6.1)

H.263 ITU-T Recommendation H.263, video standard (see section 2.6.3)

HDTV High-Definition TV

HTTP Hyper-Text Transport Protocol

ICMP Internet Control Message Protocol (see section 2.1.4)

ICMPv4 Internet Control Message Protocol, Version 4 (see section 2.1.4 and IPv4)

ICMPv6 Internet Control Message Protocol, Version 6 (see section 2.1.4 and IPv6)

IETF Internet Engineering Task Force, `http://www.ietf.org`

IntServ Integrated Services (see section 2.2.2).

IP Internet Protocol (see section 2.1.2)

IPv4 Internet Protocol, Version 4 (see section 2.1.2)

IPv6 Internet Protocol, Version 6 (see section 2.1.2)

ITU International Telecommunication Union, `http://www.itu.int`

MP3 MPEG-1/2 Layer 3, audio standard (see section 2.6.4)

MPEG Motion Pictures Experts Group, video standard (see section 2.6.1 and 2.6.2)

MPQM Moving Pictures Quality Metric, video quality metric (see section 2.6.1).

MRSD Multiple Resource Single QoS Dimension (see section 2.5.4)

MRMD Multiple Resource Multiple QoS Dimension (see section 2.5.4)

PHB Per-Hop Behavoir (see section 2.2.3)

RGB Red Green Blue, color model (see section 2.6.1)

PSNR Peak Signal Noise Ratio, video quality metric (see section 2.6.1)

RSVP Resource ReSerVation Protocol (see section 2.2.2)

RTCP RTP Control Protocol (see section 2.1.5 and RTP)

RTP Real-Time Transport Protocol (see section 2.1.5)

RTT Round Trip Time (see section 2.1.4 and 3.1)

RU Resource/Utilization (see section 2.5)

SDES Source DEScription (see section 2.1.5)

SDTV Standard-Definition TV

SLA Service Level Agreement (see section 2.2.3)

SMTP Simple Mail Transport Protocol

SSRC Synchronization SouRCe (see section 2.1.5)

SQCIF $\frac{1}{16}\times$ CIF resolution: 88×72 pixels

SRSD Single Resource Single QoS Dimension (see section 2.5.4)

SRMD Single Resource Multiple QoS Dimension (see section 2.5.4)

TCP Transmission Control Protocol (see section 2.1.3)

TTL Time To Live, field of IPv4 header (see section 2.1.2)

TOS Type of Service, field of IPv4 header (see section 2.1.2)

TV TeleVision

PEAQ Perceptual Evaluation of Audio Quality, audio quality metric (see section 2.6.4)

QCIF $\frac{1}{4}\times$ CIF resolution: 176×144 pixels

QoS Quality of Service (see section 2.2).

UDP User Datagram Protocol (see section 2.1.3)

UMTS Universal Mobile Transmission System

VBR Variable Bit-Rate

VoD Video on Demand (see chaper 2)

WWW World-Wide Web

YUV Luminance Y, Chrominance U, V, color model (= YC_rC_b, see section 2.6.1)

Bibliography

[ACH95] Aurrecoechea/Compbell/Hauw, 1994
A Survey of Quality of Service Architectures
Lancaster University Technical Report
Lancaster/England, 1994
http://www.bigfoot.com/~dreibholz/diplom/ACH95.ps.gz

[AKM+00] Albrecht/Köster/Martini/Frank, 2000
End-to-end QoS Management for Delay-sensitive Scalable Multimedia Streams over DiffServ
25th Annual IEEE Conference on Local Computer Networks (LCN)
Tampa/U.S.A., 2000
http://www.bigfoot.com/~dreibholz/diplom/AKM+00.pdf

[AFK+95] Apteker/Fisher/Kisimov/Neishlos, 1995
Video Acceptability and Frame Rate
IEEE Multimedia
Johannesburg/South Africa, 1995

[Ber00] H.263 Traces Archive
Technical University of Berlin, Dept. of Computer Science
Berlin/Germany
http://www-tkn.ee.tu-berlin.de/~fitzek/TRACE/ltvt.html

[BCC+99] Bocheck/Campbell/Chuang/Liao, 1999
Utility-based Network Adaption for MPEG-4 Systems
Proceedings of the 9th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)
Basking Ridge/U.S.A., 1999
http://www.bigfoot.com/~dreibholz/diplom/BCC+99.pdf

[BDT+96] Basso/Dalgić/Tobagi/van den Branden, 1996
Study of MPEG-2 Coding Performance based on a Perceptual Quality Metric
Proceedings of the Picture Coding Symposium (PCS)
Melbourne/Australia, 1996
http://www.bigfoot.com/~dreibholz/diplom/BDT+96.ps.gz

[Bra99] Brandenburg, 1999
MP3 and AAC Explained
17th International Conference on High Quality Audio Coding
Florence/Italy
http://www.bigfoot.com/~dreibholz/diplom/Bra99.ps.gz

[CCH94a] Campbell/Coulson/Hutchinson, 1994
A Quality of Service Architecture
ACM Computer Communication Review
`http://www.bigfoot.com/~dreibholz/diplom/CCH94a.ps.gz`

[CCH94b] Campbell/Coulson/Hutchinson, 1994
Flow Management in a Quality of Service Architecture
5th International IFIP Conference on High Performance Computing
Grenoble/France, 1994
`http://www.bigfoot.com/~dreibholz/diplom/CCH94b.ps.gz`

[CKS99] Courcoubetis/Kelly/Siris/Weber, 1999
A study of simple usage-based charging schemes for broadband networks (Rev. 2)
Proceedings of the IFIP International Conference of Broadband Communication (BC)
Stuttgart/Germany, 1998
`http://www.bigfoot.com/~dreibholz/diplom/CKS99.ps.gz`

[CZ99] Cao/Zegura, 1999
Utility Max-Min: An Application-Oriented Bandwidth Allocaton Scheme
IEEE Infocom Conference on Computer Communications
New York/U.S.A., 1999
`http://www.bigfoot.com/~dreibholz/diplom/CZ99.ps.gz`

[DSV00] Dreibholz/Selzer/Vey, 2000
Echtzeit-Audioübertragung mit QoS-Management in einem DiffServ-Szenario
University of Bonn, Dept. of Computer Science IV
Bonn/Germany, 2000
`http://www.bigfoot.com/~dreibholz/rn/Bericht.ps.gz`

[FJ93] Floyd/Jacobson, 1993
Random Early Detection Gateways for Congestion Avoidance
IEEE/ACM Transactions on Networking, 1993
`http://www.bigfoot.com/~dreibholz/diplom/FJ93.ps.gz`

[Gum98] Gumbrich, 1998
Effiziente Datenübertragung für interaktive Videoanwendungen bei Einsatz der dynamischen Bandbreitereservierung in ATM-Netzwerken
Dissertation Universität Bonn, GCA-Verlag
Herdecke/Germany, 1999

[H.263] International Telecommunication Union (ITU)
ITU-T Recommendation H.263

[IETF] Internet Engineering Task Force (IETF)
`http://www.ietf.org`

[Kar01] Karl, 2001
Aggregierte Ende-zu-Ende Bandbreitenregelung zur TCP-freundlichen Übertragung von Echtzeitströmen in Best-Effort-Netzen
University of Bonn, Dept. of Computer Science IV
Bonn/Germany, 2000

[KWL+95] Knightly/Wrege/Liebeherr/Zhang, 1995
Fundamental Limits and Tradeoff of Providing Deterministic Guarantees to VBR Video Traffic
Proceedings of the Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)
Ottawa/Canada, 1995
`http://www.bigfoot.com/~dreibholz/diplom/KWL+95.ps.gz`

[KZ97] Knightly/Zhang, 1997
D-BIND: An Accurate Traffic Model for Providing QoS Guarantees to VBR Traffic
IEEE/ACM Transactions on Networking, 1997
`http://www.bigfoot.com/~dreibholz/diplom/KZ97.ps.gz`

[LAME] The LAME MP3 Encoder
`http://www.mp3dev.org/mp3`

[LinuxDS] Differentiated Services on Linux
`http://icawww1.epfl.ch/linux-diffserv`

[LLR+99] Lee/Lehoczky/Rajkumar/Siewiorek, 2000
On Quality of Service Optimization with Discrete QoS Options
5th Real-Time Technology Symposium (RTAS),
Vancouver/Canada, 1999
`http://www.bigfoot.com/~dreibholz/diplom/LLR+99.pdf`

[LS98] Lee/Siewiorek, 1998
An Approach for Quality of Service Management
Carnegie Mellon University Technical Report
Pittsburgh/U.S.A., 1998
`http://www.bigfoot.com/~dreibholz/diplom/LS98.ps.gz`

[LW96] Liebeherr/Wrege, 1997
An Efficient Solution to Traffic Characterization of VBR Video in Quality-of-Service Networks
IEEE Infocom Conference on Computer Communications
San Francisco/U.S.A., 1996
`http://www.bigfoot.com/~dreibholz/diplom/LW96.ps.gz`

[MM00] Steinmetz 2000,
Multimedia-Technologie
3. Auflage
Springer-Verlag

[MPEG] `http://www.mpeg.org`

[Napster] Napster
Homepage: `http://www.napster.com`
Unix/KDE client 'knapster': `http://sourceforge.net/project/?group_id=1456`

[NS96] Nahrstedt/Smith, 1996
Design, Implementation and Experiences of the OMEGA End-Point Architecture
Urbana-Champaign/U.S.A., 1996
`http://www.bigfoot.com/~dreibholz/diplom/NS96.ps.gz`

[Rog98] Glynn Rogers, 1998
Mapping User Level QoS from a Single Parameter
Proceedings of the 2nd International Conference on Multimedia Networks and Services (MMNS)
Versailles/France, 1998
`http://www.bigfoot.com/~dreibholz/diplom/Rog98.ps.gz`

[Rad01] Radi, 2001
Entwurf und simulative Bewertung der kostenoptimalen Übertragung von Multimedia-Daten über Bandwidth-Broker-gesteuerte DiffServ-Netzwerke
University of Bonn, Dept. of Computer Science IV
Bonn/Germany, 2000

[NW-IDS] Network Wizards Internet Domain Survey
http://www.nw.com `http://www.nw.com`

[RFC 761] Defense Advanced Research Projects Agency, January 1980
RFC 761: Transmission Control Protocol (TCP)
`ftp://ftp.isi.edu/in-notes/rfc761.txt`

[RFC 764] Postel/Reynolds, October 1985
RFC 959: File Transfer Protocol (FTP)
`ftp://ftp.isi.edu/in-notes/rfc959.txt`

[RFC 768] Postel, August 1980
RFC 768: User Datagram Protocol (UDP)
`ftp://ftp.isi.edu/in-notes/rfc768.txt`

[RFC 791] Defense Advanced Research Projects Agency, September 1981
RFC 791: Internet Protocol, Version 4 (IPv4)
`ftp://ftp.isi.edu/in-notes/rfc791.txt`

[RFC 792] Postel, September 1981
RFC 792: Internet Control Message Protocol (ICMPv4)
`ftp://ftp.isi.edu/in-notes/rfc792.txt`

[RFC 821] Postel, August 1982
RFC 821: Simple Mail Transfer Protocol
`ftp://ftp.isi.edu/in-notes/rfc821.txt`

[RFC 959] Postel, June 1980
RFC 764: Telnet Protocol Specification
`ftp://ftp.isi.edu/in-notes/rfc764.txt`

[RFC 1071] Braden/Borman/Partridge, September 1988
RFC 1071: Computing the Internet Checksum
`ftp://ftp.isi.edu/in-notes/rfc1071.txt`

[RFC 1305] Mills, March 1992
RFC 1305: Network Time Protocol (Version 3) Specification, Implementation and Analysis
`ftp://ftp.isi.edu/in-notes/rfc1305.txt`

[RFC 1323] Jacobson/Braden/Borman, May 1992
RFC 1323: TCP Extensions for High Performance
`ftp://ftp.isi.edu/in-notes/rfc1323.txt`

[RFC 1700] Reynolds/Postel, October 1994
RFC 1700: Assigned Numbers
`ftp://ftp.isi.edu/in-notes/rfc1700.txt`

[RFC 1715] Huitema, November 1994
The H Ratio for Address Assignment Efficiency
`ftp://ftp.isi.edu/in-notes/rfc1715.txt`

[RFC 1884] Hinden/Deering, December 1995
RFC 1884: IP Version 6 Addressing Architecture
`ftp://ftp.isi.edu/in-notes/rfc1884.txt`

[RFC 1889] Schulzrinne/Casner/Frederik/Jacobson, January 1996
RFC 1889: A Transport Protocol for Real-Time Applications (RTP)
`ftp://ftp.isi.edu/in-notes/rfc1889.txt`

[RFC 1890] Schulzrinne, January 1996
RFC 1890: RTP Profile for Audio and Video Conferences with Minimal Control
`ftp://ftp.isi.edu/in-notes/rfc1890.txt`

[RFC 2068] Fielding/Irvine/Gettys/Mogul/Frystyk/Berners-Lee, January 1997
RFC 2068: Hypertext Transfer Protocol - HTTP/1.1
`ftp://ftp.isi.edu/in-notes/rfc2068.txt`

[RFC 2205] Braden/Zhang/Berson/Herzog/Jamin, September 1997
RFC 2205: Resource ReSerVation Protocol (RSVP) - Functional Specification
`ftp://ftp.isi.edu/in-notes/rfc2205.txt`

[RFC 2208] Dell/Romanow/Weinrib/Zhang, September 1997
RFC 2208: Resource ReSerVation Protocol (RSVP) - Applicability Statement
`ftp://ftp.isi.edu/in-notes/rfc2208.txt`

[RFC 2235] Zakon, November 1997
RFC 2235: Hobbes' Internet Timeline
`ftp://ftp.isi.edu/in-notes/rfc2235.txt`

[RFC 2460] Deering/Hinden, December 1998
RFC 2460: Internet Protocol, Version 6 (IPv6)
`ftp://ftp.isi.edu/in-notes/rfc2460.txt`

[RFC 2462] Thomson/Narten, December 1998
RFC 2462: IPv6 Stateless Address Autoconfiguration
`ftp://ftp.isi.edu/in-notes/rfc2462.txt`

[RFC 2464] Conta/Deering, December 1998
RFC 2463: Internet Control Message Protocol (ICMPv6)
`ftp://ftp.isi.edu/in-notes/rfc2235.txt`

[RFC 2474] Vaudreuil/Parsons, September 1998
RFC 2474: Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers
`ftp://ftp.isi.edu/in-notes/rfc2474.txt`

[RFC 2597] Heinanen/Baker/Weiss/Wroclawski, June 1999
Assured Forwarding PHB Group
`ftp://ftp.isi.edu/in-notes/rfc2597.txt`

[RFC 2598] Jacobson/Nichols/Poduri, June 1999
Expedited Forwarding PHB
`ftp://ftp.isi.edu/in-notes/rfc2598.txt`

[RN98] Martini, 1998/1999
Vorlesung Rechnernetze I/II
University of Bonn
Bonn/Germany, 1998/1999

[RTLinux] RTLINUX - The Realtime LINUX
`http://www.rtlinux.org`

[RTP Audio] The RTP AUDIO Homepage
`http://www.bigfoot.com/~dreibholz/rn`

[Sel01] Selzer, 2001
Domain-interne Verteilung von QoS-Ressourcen durch einen Bandwidth-Broker in einem DiffServ- Szenario
University of Bonn, Dept. of Computer Science IV
Bonn/Germany, 2000

[TTB+98] Thiede/Treurniet/Bitto/Sporer/Brandenburg/Schmidmer/ Keyhl/Beerends/ Colomes/Stoll/Feiten, 1998
PEAQ - der künftige ITU-Standard zur objektiven Messung der wahrgenommenen Audioqualität
20. Tonmeistertagung, Karlsruhe 1998
`http://www.bigfoot.com/~dreibholz/diplom/TTB+98.ps.gz`

[Vey01] Vey, 2001
Entwurf und Implementierung eines QoS-Managements zur Übertragung gelayerter und skalierbarer Multimedia-Ströme mit variabler Datenrate über DiffServ
University of Bonn, Dept. of Computer Science IV
Bonn/Germany, 2000

[Tan96] Tanenbaum, 1996
Computer Networks
Prentice Hall PTR

[Wat98] Watson, 1998
Towards a perceptual video quality metric
NASA Ames Research Center
Moffett Field/U.S.A.
`http://www.bigfoot.com/~dreibholz/diplom/Wat98.ps.gz`

[WCH96] Waddington/Coulson/Hutchinson, 1996
Specifying QoS for Multimedia Communications within Distributed Programming Environments
Proceedings of the 3rd International COST237 Workshop, 1996
`http://www.bigfoot.com/~dreibholz/diplom/WCH96.ps.gz`

[Wür95] MPEG-1 Traces Archive
University of Würzburg, Dept. of Computer Science
Würzburg/Germany
`http://www-info3.informatik.uni-wuerzburg.de/MPEG`

[ZL96] Zeng/Liu, 1996
Rate shaping by block dropping for transmission of MPEG precoded video over channels of dynamic bandwidth
4th ACM Multimedia Conference
Boston/U.S.A., 1996
`http://www.bigfoot.com/~dreibholz/diplom/ZL96.ps.gz`

List of Algorithms

1 Optimal calculation of remapping intervals from [Gum98] 17
2 The ASRMD1 algorithm from [LS98] 26
3 Calculation of a given bandwidth's resource/utilization point 54
4 The media-dependent resource/utilization list calculation part for MPEG 55
5 Recursive resource/utilization list calculation 57
6 Conversion from payload to raw bandwidth 63
7 Conversion from raw to payload bandwidth 64
8 Calculation of session's resource/utilization multipoints. 70
9 The complete remapping 71
10 An allocation trial for a resource/utilization multipoint 72
11 An allocation trial for a resource/utilization point 73
12 The partial remapping 74

List of Tables

2.1 The OSI Reference Model . . . 3
2.2 The TCP/IP Reference Model . . . 3
2.3 The IPv4 header . . . 4
2.4 The IPv6 header . . . 4
2.5 The IPv6 extension header . . . 4
2.6 The RTP header . . . 7
2.7 The *Traffic Class*/*Type of Service* field . . . 10
2.8 The example's empirical envelope and bandwidth to be reserved . . . 13
2.9 Traffic models and their traffic constraint functions . . . 15

3.1 An example session for a virtual shopping mall . . . 35

4.1 25 frames of the MPEG-1 video "*Terminator II*" . . . 43
4.2 Example of the space optimization . . . 48
4.3 The result of the space optimization . . . 48

5.1 A resource/utilization point . . . 53
5.2 Using the trival calculation . . . 56
5.3 Using the recursive algorithm . . . 56
5.4 Recursion levels for the resource/utilization list example . . . 58
5.5 An example resource/utilization list of an MPEG-1 video . . . 58

6.1 A stream description . . . 61
6.2 A session description . . . 62
6.3 Example traffic constraint for layer #1 . . . 65
6.4 Example traffic constraint for layer #2 . . . 65
6.5 DiffServ class delays for the layer to class mapping example . . . 66
6.6 Cost for layer #1 for an original bandwidth of 1500 KBytes/s . . . 66
6.7 Cost for layer #2 for an original bandwidth of 600 KBytes/s . . . 66
6.8 Sorting values for different priorities and fairness settings . . . 68
6.9 SLAs for the bandwidth management example . . . 73

7.1 An MPEG-2 trace example with frame sizes in bytes . . . 79
7.2 The results of [AFK+95] . . . 81
7.3 Used genre mappings for the traces . . . 81
7.4 User satisfaction for different frame rate and frame size requirements . . . 83
7.5 Used jitter and loss rate limits for each media type . . . 85
7.6 Standard class settings for the buffering, layering and weighting simulations . . . 86
7.7 The DiffServ scenario's SLA . . . 88

8.1 The DiffServ class settings of the network quality example . . . 104
8.2 The DiffServ class settings of the priority example . . . 107

8.3 The session and stream priority example scenario inclusive resulting utilizations . . . 107
8.4 The system load scenario . . . 109
8.5 System load results . . . 109
8.6 Class settings for the partial remapping simulations . . . 113
8.7 The real network scenario . . . 114
8.8 Total number of buffer flushes for different system delay tolerances . . . 114
8.9 Average durations of stream description initializations . . . 115

A.1 MPEG-1 trace statistics . . . 121
A.2 MPEG-2 trace statistics . . . 122
A.3 H.263 trace statistics . . . 123
A.4 MP3 trace statistics . . . 124

List of Figures

2.1 An example for DiffServ domains . . . 10
2.2 The RED dropping probability . . . 12
2.3 A leaky bucket and its canonical comparision . . . 12
2.4 An example stream consisting of 12 frames . . . 13
2.5 An empirical envelope . . . 14
2.6 Bandwidth to be reserved . . . 14
2.7 An example for long-term and short-term burstiness . . . 14
2.8 A traffic model comparision . . . 16
2.9 Graphical explaination of the remapping intervals calculation . . . 17
2.10 The layered QoS model . . . 18
2.11 Utilization points => interpolated and approximated utility function . . . 19
2.12 The utility function $u(x) = a * \ln(b * x + c)$ . . . 20
2.13 The utility function $u(x) = 1 - e^{a*x+b}$ . . . 21
2.14 The utility function composition $U(x) = \sum_{i=1}^{n} \omega_i * u_i(x_i)$ from [LS98] . . . 21
2.15 The utility function composition $U(x) = \frac{n}{\sum_{i=1}^{n} \frac{1}{u_i(x_i)}}$ from [Rog98] . . . 22
2.16 The resource management based on [LS98] . . . 23
2.17 An example for the ASRMD1 algorithm . . . 27
2.18 The MPEG codec . . . 28
2.19 An MPEG example (“*The Silence of the Lambs*”, GoP: IBBPBBPBBPBB) . . . 30
2.20 Base layer only . . . 32
2.21 Base and enhancement layer . . . 32
2.22 The MP3 codec . . . 33
2.23 An MP3 Example (Go West) . . . 33

3.1 The CORAL concept . . . 36
3.2 The stream hierarchy of the CORAL concept . . . 36
3.3 The RTP AUDIO system . . . 37
3.4 The offline part . . . 39
3.5 The online part . . . 39

4.1 The comparision of overlapping and non-overlapping intervals . . . 42
4.2 Per-layer intervals . . . 44
4.3 Per-stream intervals . . . 44
4.4 Remapping intervals calculated using unweighted and weighted cost . . . 44
4.5 A frame rate scalability example . . . 45
4.6 A frame size scalability example . . . 46
4.7 The runtime-optimized remapping interval calculation . . . 47
4.8 The space-optimized remapping interval calculation . . . 49

5.1 Remapping intervals for different frame rates . . . 51
5.2 Example for lower frame rate but higher bandwidth requirement . . . 54

5.3 An example utility function for the resource/utilization list calculation 55
5.4 The recursive resource/utilization list calculation algorithm 56

6.1 Empirical envelopes and bandwidth to reserve for layer #1 and #2 65
6.2 The example's resulting multipoint list 71
6.3 Some concave functions 76

7.1 Utility functions for video frame rate and size 84
7.2 Utility function for audio frame size 84
7.3 An overview of the layering, RTP transport and traffic shaping implementation . . . 85
7.4 The real DiffServ scenario 87

8.1 Buffering gain and DiffServ class usage for layer #1 (I-frames) 90
8.2 Buffering gain and DiffServ class usage for layer #2 (P-frames) 90
8.3 Buffering gain and DiffServ class usage for layer #3 (B-frames) 90
8.4 Reserved bandwidth for MPEG-1 video "*The Silence of the Lambs*" 91
8.5 Unweighted remapping intervals 92
8.6 Weighted remapping intervals: 4/3/2 92
8.7 MPEG-1 average cost/delay comparision 94
8.8 MPEG-1 average bandwidth/delay comparision 94
8.9 MPEG-1 average cost factor/delay comparision 94
8.10 MPEG-2 average cost/delay comparision 96
8.11 MPEG-2 average bandwidth/delay comparision 96
8.12 MPEG-2 average cost factor/delay comparision 96
8.13 H.263 average cost/delay comparision 97
8.14 H.263 average bandwidth/delay comparision 97
8.15 H.263 average cost factor/delay comparision 97
8.16 MP3 average cost/delay comparision 98
8.17 MP3 average bandwidth/delay comparision 98
8.18 MP3 average cost factor/delay comparision 98
8.19 MPEG-1 cost for 100% utilization 100
8.20 MPEG-1 cost for 75% utilization 100
8.21 MPEG-1 cost for 50% utilization 100
8.22 MPEG-1 cost for 25% utilization 100
8.23 MPEG-2 cost for 100% utilization 102
8.24 MPEG-2 cost for 75% utilization 102
8.25 MPEG-2 cost for 50% utilization 102
8.26 MPEG-2 cost for 25% utilization 102
8.27 H.263 cost for 100% utilization 103
8.28 H.263 cost for 75% utilization 103
8.29 H.263 cost for 50% utilization 103
8.30 H.263 cost for 25% utilization 103
8.31 MP3 cost for different utilizations 104
8.32 Total cost comparision without (left) and with (right) network quality changes 105
8.33 DiffServ class for layer #1 (I) for both network quality simulations 106
8.34 DiffServ class for layer #2 (P) for both network quality simulations 106
8.35 DiffServ class for layer #3 (B) for both network quality simulations 106
8.36 Total reserved bandwidth and cost of the first system load simulation 111
8.37 Reduction of bandwidth and cost, compared to figure 8.36 111
8.38 Average cost and bandwidth for different maximum complete remapping interval settings 112

8.39 Average utilization for different maximum complete remapping interval settings . . . 112
8.40 Remappings for different maximum complete remapping interval settings 112
8.41 Total number of buffer flushes for different system delay tolerances 115

B.1 MPEG-1 cost/delay comparision 126
B.2 MPEG-1 bandwidth/delay comparision 127
B.3 MPEG-1 average cost factor/delay comparision 128
B.4 MPEG-2 cost/delay comparision 129
B.5 MPEG-2 bandwidth/delay comparision 130
B.6 MPEG-2 average cost factor/delay comparision 131
B.7 H.263 cost/delay comparision 132
B.8 H.263 bandwidth/delay comparision 133
B.9 H.263 average cost factor/delay comparision 134
B.10 MP3 cost/delay comparision 135
B.11 MP3 bandwidth/delay comparision 135
B.12 MP3 average cost factor/delay comparision 135

C.1 MPEG-1 cost, original 138
C.2 Cost reduction for 75% util. 138
C.3 Cost reduction for 50% util. 138
C.4 Cost reduction for 25% util. 138
C.5 MPEG-1 bandwidth, original 139
C.6 Bandwidth reduction for 75% util. 139
C.7 Bandwidth reduction for 50% util. 139
C.8 Bandwidth reduction for 25% util. 139
C.9 MPEG-1 cost factor, original 140
C.10 Cost factor reduction for 75% util. 140
C.11 Cost factor reduction for 50% util. 140
C.12 Cost factor reduction for 25% util. 140
C.13 MPEG-2 cost, original 141
C.14 Cost reduction for 75% util. 141
C.15 Cost reduction for 50% util. 141
C.16 Cost reduction for 25% util. 141
C.17 MPEG-2 bandwidth, original 142
C.18 Bandwidth reduction for 75% util. 142
C.19 Bandwidth reduction for 50% util. 142
C.20 Bandwidth reduction for 25% util. 142
C.21 MPEG-2 cost factor, original 143
C.22 Cost factor reduction for 75% util. 143
C.23 Cost factor reduction for 50% util. 143
C.24 Cost factor reduction for 25% util. 143
C.25 H.263 cost, original 144
C.26 Cost reduction for 75% util. 144
C.27 Cost reduction for 50% util. 144
C.28 Cost reduction for 25% util. 144
C.29 H.263 bandwidth, original 145
C.30 Bandwidth reduction for 75% util. 145
C.31 Bandwidth reduction for 50% util. 145
C.32 Bandwidth reduction for 25% util. 145
C.33 H.263 cost factor, original 146

C.34 Cost factor reduction for 75% util. 146
C.35 Cost factor reduction for 50% util. 146
C.36 Cost factor reduction for 25% util. 146
C.37 MP3 cost, original 147
C.38 Cost reduction for 75% utilization 147
C.39 Cost reduction for 50% utilization 148
C.40 Cost reduction for 25% utilization 148
C.41 MP3 bandwidth, original 149
C.42 Bandwidth reduction for 75% utilization 149
C.43 Bandwidth reduction for 50% utilization 150
C.44 Bandwidth reduction for 25% utilization 150
C.45 MP3 cost factor, original 151
C.46 Cost factor reduction for 75% utilization 151
C.47 Cost factor reduction for 50% utilization 152
C.48 Cost factor reduction for 25% utilization 152

Index

A priori, 38, 41, 62
Additional depth, 57
AF, 11
Amstel, background traffic, 88
Andechs, client, 87
APP, RTCP, 9
Application profile, 23
Application utilities, 24
ASRMD, 25, 68
Assured forwarding, 11
ATM, 4

B-frame, 29
Bandwidth broker, 120
Bandwidth broker, CORAL, 36, 74, 87, 93, 108
Bandwidth threshold, 56
Bandwidth, charges, 37
Bandwidth, payload, 63
Bandwidth, pricing, 76
Bandwidth, raw, 63
Bandwidth, remapping, 16, 70
Best effort, 9, 32, 35, 37, 87
Bitrate, MP3, 33
Block dropping, MPEG, 30
Block, MPEG, 29
Bonn, 1, 86
Border router, 11
Buffer delay, 12, 64
Buffer flush, 12, 87
Buffer overflow, 87
Buffering gain, 65
Burstiness, 14
BYE, RTCP, 9

CBR, 12
Checksum, IP, 5
Chrominance, 29
Classifier, DiffServ, 11
CNAME, RTP, 8
Coefficient elimination, MPEG, 30
Color subsampling, 29
Color transformation, 29
Complete remapping, 72
Composition, utility functions, 20
Compression, audio, 32
Compression, video, 28
Congestion Management, CORAL, 37
Congestion, AF, 11
Congestion, TCP, 6
Connection-less, 6
Connection-oriented, 6
Conslusions, 119
Constant bitrate, 12
Contributing source, RTP, 8
Convex hull, ASRMD, 25, 76
Convex, traffic constraint, 15, 16
CORAL, 35
CORAL, concept, 35
CORAL, layered transmission, 35
CORAL, RTP Audio, 37
Core router, 10
Corona, server, 87

D-BIND, traffic model, 16
Data partitioning, MPEG-2, 32
DCT, 28, 29
Detmolder, background traffic, 88
Differentiated Services, 10
DiffServ, 10
DiffServ code point, 10
DiffServ on Linux, 88
Discrete cosine transformation, 28, 29
Dreibholz, Thomas, i
Drop precedence, AF, 11
Dropper, DiffServ, 11
DSCP, 10
DVQ, Digital Video Quality, 31

Echo reply, ICMP, 7
Echo request, ICMP, 7
ECM, 37
EF, 11
Elastic traffic, 6
Empirical envelope, 14
Endpoint identification, 6
Error picture, 29

Ethernet, 4, 63
Expedited forwarding, 11

Fair sharing, 24
Fairness, 75
Fairness, sessions, 70
Fairness, streams, 67
FDDI, 4, 63
Flowlabel, IPv6, 6, 10
Flush, buffer, 12
Forwarding, assured, 11
Forwarding, expedited, 11
Fragmentation, IP, 5
Frame, 27
Frame count, 63
Frame dropping, MPEG, 30
Frame position, 51
Frame rate, 27
Frame rate calability, 45
Frame size calability, 46
Frame type, I/P/B, 29
Frame type, PB, 32

Gaffel, background traffic, 88
Genre, video, 81
GoP, 30
Grolsch, router, 87
Group of pictures, 30

H.263, video, 32
HDTV, 31
Header, IPv4, 4
Header, IPv6, 4
Header, IPv6 Extension, 4
Header, packet, 62
Header, RTP, 7
Hierarchy, protocol, 3
Holsten, router, 87
Hop limit, 5
Human ear, 32
Human visual system, 28

I-frame, 29
ICMP, protocol, 6
ICMPv4, protocol, 6
ICMPv6, protocol, 6
IETF, 3
Inaccuracy, scheduling, 86
Initialization, session description, 68
Initialization, stream description, 62
Integrated services, 9
Interarrival jitter, RTP, 8
Interface, 3
Internet, 1, 5
Internet Engineering Task Force (IETF), 3
Internet service provider, 10
Interpolation, 29
Intervals, per-layer, 44
Intervals, per-stream, 44
IntServ, 9
IP, address, 5
IP, header, 4
IP, protocol, 4
IPv4, 4
IPv6, 4
ISP, 10
ITU, 34

Jitter, RTP, 8
Joint stereo, 33

LAME, MP3 encoder, 33, 80
Layer, application, 4
Layer, base, MPEG-2, 31
Layer, enhancement, MPEG-2, 31
Layer, host-to-network, 4
Layer, Internet, 4
Layer, protocol, 3
Layer, transport, 4
Layered transmission, 35
Layering control, CORAL, 35
Layering, H.263, 43
Layering, MP3, 43
Layering, MPEG-1, 42
Layering, MPEG-2, 43
Leaky bucket, 11
Linux, 2, 87
Los Angeles, 86
Luminance, 29

Macroblock, MPEG, 29
Marker, DiffServ, 11
Measurement, 89
Meter, DiffServ, 11
Model, perceptual, 18
Model, QoS, 17
Motion compensation, 28, 29
Motion vector, 29
MP3, audio, 32
MPEG, decoding, 29
MPEG, encoding, 28
MPEG, error tolerance, 30

MPEG, MPEG-2, 31
MPEG, scalability, 30
MPEG, video, 27
MPQM, Moving Pictures Quality Metric, 31
MRMD, 25
MRSD, 25
Multimedia, 2, 6, 7, 9, 12, 18, 35

Napster, MP3 downloads, 80
NMR, 33
Non-convex, D-BIND, 16
Notation, colon hexadecimal, 5
Notation, compressed colon hexadecimal, 5
Notation, dotted decimal, 5
NP-complete, 25

Offline, 38
Online, 39
Optical fibre, 86
Over-provisioning, 16

P-frame, 29
Parallelization, 68
Partial remapping, 74
Payload data, 62
PB-frame, 32
PEAQ, 34
Peer, 3
Per-hop behaviour, 11
Perceptual model, 32
PHB, 11
Picture type, MPEG, 29
Ping, 87
Policing, traffic, 12
Port number, TCP, 6
Port number, UDP, 6
Position, 51
Premium service, 11
Pricing, bandwidth, 76
Priority factor, 67
Priority, session, 68
Priority, stream, 67
Profile, application, 23
Profile, resource, 24
Profile, task, 23
Profile, user, 23
Protocol, 3

QoS constraint, 24
QoS manager, CORAL, 36
QoS model, 17
QoS optimization problem, 24
QoS, dimensions, 17
QoS, requirements, 9
Quality metric, 18
Quality metric, MPEG, 31
Quality of service, 9
Quality space, 23
Quality, perceptual, 17
Quantization noise, 29

Random realy drop, 11
Receiver report, RTCP, 8, 38, 64, 88
RED, 11
Reference model, OSI, 3
Reference model, TCP/IP, 3
Remapping, bandwidth, 16
Remapping, complete, 72
Remapping, interval, 17
Remapping, partial, 74
Report, RTCP, 8, 64, 88
Requantization, MPEG, 31
Reservation module, CORAL, 36
Resource profile, 24
Resource/utilization list, 24, 55, 56
Resource/utilization list calculation, 56
Resource/utilization list example, 58
Resource/utilization multipoint, 68
Resource/utilization point, 24, 53, 55, 68
RGB, 28
Round trip time, 7
RTCP, 38, 64, 88
RTCP, protocol, 7, 8
RTP Audio, 37, 38, 85
RTP, CNAME, 8
RTP, protocol, 7

Satellite link, 87
Scalability, frame rate, 45
Scalability, frame size, 46
Scalable coding extensions, MPEG-2, 31
Scale factor alpha, 46
Scale factor theta, 52
Sender report, RTCP, 8
Sequence number, RTP, 7, 88
Service level agreement, 10
Session Description, 68
Session description, 62
Session priority, 68
Session, CORAL, 35
Session, RTP, 8
Shaper, DiffServ, 11

Shaping, traffic, 11
Sharing, utility-fair, 24
Sigma-Rho, traffic model, 15
Signal speed, 86
Simulation, 87, 89
Size ratio, MPEG frame types, 30
SLA, 10
Slice, MPEG, 29
SNR scalability, MPEG-2, 32
Socket, 6
Sorting value, 67
Source description, RTCP, 8
Spatial scalability, MPEG-2, 31
Speed adjustment approach, 87
SRMD, 25
SRSD, 25
Statistics, traces, 121
Stream Description, 62
Stream description, 61
Stream priority, 35, 67
Stream-oriented, 6
Subadditivity, 13
Substream, 35
Synchronization source, RTP, 8
System utility, 24

Task, 23
Task profile, 23
TCP, protocol, 6
TCP-friendly, 37
Temporal component, video watchability, 81
Temporal scalability, MPEG-2, 31
Threshold, bandwidth, 56
Threshold, utilization, 56
Time stamp, RTP, 8
Time to live, 5
Time-invariance, 13
TOS, 6
Trace statistics, 121
Traces, 79
Traffic class, 6
Traffic constraint function, 13, 14
Traffic description, 14
Traffic model, 15
Traffic policing, 12
Traffic shaping, 11
Transfer delay calculation, CORAL, 38
Transmission, layered, 35
Transmission, reliable, 6
Transmission, unreliable, 6

TTL, 5
Type of service, 6

UDP, protocol, 6
Unprioritized sorting value, 67
User profile, 23
User ratings, 81
User satisfaction, 17
Utility function, 18
Utility, application, 24
Utility, system, 24
Utility-fair sharing, 24
Utilization, 17
Utilization threshold, 56

Vacuum light speed, 86
Variability, transfer delay, 64
Variable bitrate, 12
VBR, 12
Virtual shopping mall, 35
Visual component, video watchability, 81

Watchability, 81

YUV, 28