The Influence of Proportional Jitter Scheduling Algorithms on Differentiated Services Networks

Vorgelegt von M. S. Thu Ngo Quynh
aus Berlin

Von der Fakultät Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
- Dr. - Ing. -

genehmigte Dissertation

Promotionsausschuss:

Vorzitzender:	**Prof. Dr. –Ing. Dr.rer.nat. Holger Boche**
Berichter:	**Prof. Dr. – Ing. Adam Wolisz**
Berichter:	**Prof. Dr. rer. nat. Martina Zitterbart**

Tag der wissenschaftliche Aussprache: 10. Juni 2003

Berlin 2003
D83

Abstract

The transformation of the Internet into an important commercial infrastructure in the recent years has led to the emergence of new service needs as it is required to carry a wide range of application information. It is widely agreed that the Internet architecture should offer some type of service differentiation, so that some traffic classes get better QoS (Quality of Service) than others. Currently, the attention of the research community has been focused on the Differentiated Service Architectures (DiffServ).

A model attracting much attention from the research communities recently is the Proportional Differentiated Service Model, which provides proportional services between different classes. There are some existing studies on mechanisms to provide the proportional service, such as Proportional Delay Service (PDDM), Proportional Loss Service, WTP, BPR, MDP, and DDTS etc. Even when such mechanisms are implemented at every router, it is not always possible to receive per-class proportional service in an end-to-end manner.

In order to overcome this issue, attention concentrates on developing a new and simple model (called Proportional Jitter Differentiated Services - PJDM) that does provide proportional jitter between different classes based on the Jitter Differentiation Parameters. Unlike other existing approaches, it is unnecessary to have complicated scheduling algorithms at every router in networks based on PJDM model. Subsequently, the issue of related packet scheduling problems is considered: four new schedulers for PJDM model are created in this work. The Relative Jitter Packet Scheduling (RJPS) and the Proportional Average Jitter (PAJ) algorithms provide long-term jitter and short-term jitter ratio proportionally between different classes. Furthermore I consider the use of variable Jitter Differentiation Parameters in RJPS and PAJ, this idea leads me to create two new mechanisms, called Adaptive RJPS and Adaptive PAJ, which are more robust than the previous RJPS/PAJ mechanisms under bursty traffic profiles.

The focus then shifts to a comparison of quality of service provided by PDDM and PJDM in terms of end-to-end delay. Results received from my simulation confirm that the topologies based on my new model PJDM achieve significantly better quality of service than the others, which derive from the old model PDDM.

Acknowledgements

Thank you to Professor Adam Wolisz and Professor Klaus Rebensburg, for their contributions in the development of my research and their guidance in my professional growth. Without their broad vision and deep insight, their valuable advice and strong encouragement, and their willingness to provide funding, this work would not have been possible. I thank them genuinely for everything I have achieved in my research so far.

The work presented in this thesis was done in the Interdepartmental Research Centre for Networking and Multimedia Technology, as well as in the Telecommunication Network Group (TKN) at the Faculty of Electrical Engineering and Computer Science, Technical University Berlin. I would like to acknowledge DAAD and all the colleagues, especially Dr. Christa Klaus, who support my 5-year staying in Germany.

Furthermore, I would like to express many thanks to the Faculty of Electronics and Telecommunications and the Personnel Department, Hanoi University of Technology, for supporting me so that I can pursue my research in Germany.

It is a great pleasure to acknowledge my friends, either German or Vietnamese, for their support and encouragements. Among them, I would like to express my earnest gratitude to my husband, Dr. Thanh Nguyen-Huu for making our many long hours at the office endurable and enjoyable. His enthusiasm, patience, work ethic and especially his love have made my thesis productive and successful. Without his support, any of my achievements would not have been possible.

I have also had many discussions with Thomas Wolfram on my research issues. Without his help, his encouragement and his friendship, I could not finish my work. I have also benefited greatly working with Dr. Holger Karl, who has given good advice to me about truly experimental system research. I thank Irina Piens for providing equipment in order to perform simulations.

Last but not least a big thank you to my family in Vietnam. Their support and encouragement made it possible for me to finish this important step in the career of a young researcher.

Contents

List of Figures

List of Tables

Acronyms

ADD: Average Drop Distance
Adaptive-PAJ or A-PAJ: Adaptive Proportional Average Jitter
API: Application Programming Interfaces
Adaptive-RJPS or A-RJPS: Adaptive Relative Jitter Packet Scheduling
ARQ: Automatic Repeat Request
ATM: Asynchronous Transfer Mode
BB: Bandwidth Broker
BPR: Backlog Proportional Rates
CBP: Complete Buffer Partitioning
CBQ: Class Based Queuing
CBR: Constant Bit Rate
CSC: Class Selector Compliant
DDP: Delay Differentiation Parameters
DDTS: Differentiated Delay and Throughput Scheduler
DiffServ: Differentiated Service
DoP: Distorsion of Playout
DS: Differentiated Service Field
DSCP: Differentiated Service Code Point
FEC: Forward Error Correction
FIFO: First In First Out
GPS: Generalized Processor Sharing
HOL: Head of Line
HTTP: HyperText Transport Protocol
HWM: High Water Mark
IETF: Internet Engineering Task Force
IntServ: Integrated Service
IP: Internet Protocol
Ipv4: Internet Protocol version 4
JDP: Jitter Differentiation Parameters
JoBS: Joint Buffer Management and Scheduling
LDP: Loss Differentiation Parameters.
LHT: Loss History Table
LPF: Lowest Priority First
LTB: Lower Target Boundary
LWM: Low Water Mark
MU: Media Unit
NEVOT: Network Voice Terminal
NN: Neural Network
NT: Network Topologies
PAJ: Proportional Average Jitter
PBS: Partial Buffer Sharing
PDD: Packet Delay Distribution

PDDM : Proportional Delay Differentiation Model
PHB: Per Hop Behaviors
PJDM or PJD: Proportional Jitter Differentiation Model
PLR: Proportional Loss Rate
PQ: Priority Queuing
PMP: Paris Metro Pricing
QoS: Quality of Service
QM: Queuing Monitoring
RED: Random Error Detection
RJPS: Relative Jitter Packet Scheduling
RSVP: Resource Reservation Protocol
RTT: Round Trip Time
SLA: Service Level Agreement
SLS: Service Level Specification
TCP: Transfer Control Protocol
TDP: Time Dependent Priority
USD: User Share Differentiation
UTB: Upper Target Boundary
VD: Virtual Delay
VLL: Virtual Leased Line
VOD: Video on Demand
WDM: Wave Division Multiplexing
WFQ: Weighted Fair Queuing
WRR: Weighted Round Robin
WTP: Waiting Time Priority
WWW: World Wide Web

Chapter 1 Introduction

This chapter presents the IP interworking environments as well as *Quality of Service* (QoS) support in such environments. It also addresses the needs of building a QoS architecture based on the *Differentiated Services* architecture and *Playout buffer delay adjustment* algorithms. Finally, it outlines problems to be solved and describes the major contribution of this thesis to the state-of-the-art.

1.1 Why IP?

Packet networks are able to transfer different types of information, such as E-mail, WWW pages, voice, music, video etc. In general, if an information unit can be digitally represented, it can also be assembled into *packets*, and can be transferred through a network. The Internet as an interconnection of many different networks is based on the Internet Protocol (IP) [Clark88]. Internet was mainly used for E-mail and File Transfer in the eighties. Later in the nineties, WWW access significantly contributed to the traffic, and recently new applications like conferencing and multimedia streaming became more and more important. This fact especially leads to relevance of the topic “QoS”.

In order to transfer packets through networks, it is necessary to have a sequence of *links* and *routers* from senders to receivers. Links are used to interconnect end users and routers with each others. Routers are nodes where packets from an input link are *forwarded* to output links. Source or destination addresses and application port numbers are stored in *packet headers* and they address a path and services which packets obtain in networks.

In an IP router, packets arrive at an *input interface* and they depart from an *output interface*. The output interface is specified by a *forwarding table*. An important feature of an IP router is the fact that packet queues are inevitable, and they can cause *delay* and *packet losses*, which are major performance-degradation factors in packet networks. Another important performance factor in packet networks is *rate*, usually called *bandwidth*, at which routers can forward and transmit packets. The operation that routers perform choosing a packet from buffer in order to put it to the free output links is called *scheduling*.

The Internet Protocol is based on connectionless mode (vs. connection oriented) and can be used to communicate across any set of interconnected networks. The major advantages of the Internet Protocol are its *flexibility* and *scalability*. There is no requirement to maintain state for individual connections. Neither a connection setup nor a teardown for packet forwarding does exist. IP and its auxiliary protocol features communicate with remote sites without a detailed

knowledge of a particular vendor's network hardware. IP is able to operate over a very wide range of underlying network technologies. In addition, IP supports a wide range of applications and institutions, from commercial use to research. The vitality and scalability of the Internet Protocol is demonstrated by the rapid expansion of the Internet every day. The most important reason why IP scales well from small local area networks to worldwide networks with millions of users lies in its connectionless nature. In case of failures the network decides which way the message will take from sender to receiver.

However, IP has also two main disadvantages. First, different quality-of-services requirements from the user can hardly be supported due to very simple forwarding mechanisms where packets are treated equal. Second, the concept of IP routing was designed more towards flexibility than rate.

1.2 Why QoS?

Quality of Service (QoS) is a generic term which takes into account several techniques and strategies that could assure application and users a predictable service from the network and other components involved, such as operating systems.

The reasons for supporting QoS models in the future are the appearance of time-sensitive applications, and the more and more ubiquitous use of Internet as work tool, congestion and uncertainties in delay and delay variation. The traditional Internet, storing and forwarding packets without guaranteed service can provide *best-effort service* only, and cannot provide acceptable performance. New real-time applications, which are less elastic and less tolerant to delay, packet losses and delay variations cannot be handled properly within the traditional data service architecture.

It is a hotly debated issue if the support of QoS is needed. One example is the *Wavelength Division Multiplexing* (WDM) that makes the future bandwidth so abundant, ubiquitous and cheap. When the link utilization is low or even moderate observed, it is unlikely that packets are lost or heavily delayed. QoS will be the same for all connections regardless of their QoS requirements. Due to large bandwidth, there will be no communication delays other than the speed of light. However, even if bandwidth would eventually become abundant and cheap, it is not going to happen soon. Moreover, bandwidth provided as a network service is not likely to become so cheap that wasting it will be the most cost-effective design principle.

In addition, different services with different price patterns are required from the Internet by the users because the users want to choose one service appropriate for

its capability. Within the same type of service requirements, there might be several service classes to be required. Some users, who are not so tolerant to packet loss and delay, need to choose a strictly guaranteed service with higher price, while others, who are tolerate to some degradation, will use a less guaranteed and cheaper service class.

Recent experience in the Internet indicates that it is ill suited to handle time and loss sensitive applications. This is due not only to the inadequate bandwidth provided by the Internet but also because the Internet does not provide the right support in the form of end-to-end protocols and adequate service to the new applications. Thus service providers have to not only provision higher link capacity but more important, they need also to introduce more sophisticated service models and architectures that can satisfy varied QoS requirements.

1.3 Why Differentiated Services (DiffServ)?

In order to provide QoS support in the Internet, two service architectures - the *Integrated Service* (IntServ) [Brad, Wrocl11] and the Differentiated Services (DiffServ) architectures - have been developed [Nich98].

The IntServ approach supports some quantified services such as minimum-service rate or a maximum tolerable end-to-end delay or loss rate for application sessions. In order to support this type of service, each router in the network has to maintain state and control information for each *flow*, which is a stream of packets belong to the same application session. This approach seems to be unfeasible for routers to perform all the above actions efficiently when there are millions of flows traversing through the network simultaneously.

The other approach, DiffServ, is newer than the IntServ approach. It proposes a coarser notion of quality of service, focusing primarily on *classes*, and intends to qualitatively differentiate services between classes rather than to provide absolute per-flow QoS guarantees. In particular, access routers process packets on the basis of finer traffic granularity such as per-flow or per-organization while routers at the core network do not maintain fine-grained state, but process traffic based on a small number of *Per Hop Behaviors* (PHBs) encoded in the packet header.

A DiffServ model that draws much attention from the research communities recently is the *Proportional Differentiated Services* Model [Dov3], which provides proportional services between different classes. There exist some studies on mechanisms to provide the proportional service, such as *Proportional Delay Service* (PDDM) and *Proportional Loss Service* [Dov2]. Waiting Time Priority WTP or Backlog Proportional Rates BPR are scheduling mechanisms designed specially for the PDDM model in [Dov3]. Even when such mechanisms are

implemented at every router, it is not always possible to receive per-class proportional service in an end-to-end manner.

1.4 Why Playout Buffer at receiver end?

Although such QoS architecture should be implemented in the Internet to guarantee QoS, the total end-to-end delay experienced by each packet is a function of variable delays due to physical media access and queuing delay. This variation of delay is considered as a big disadvantage for a stream of multimedia packets, because it influences the quality of audio-visual applications.

In order to compensate for these delay variations, a smoothing buffer (called *playout buffer*) is thus typically used at a receiver. Received packets are first queued into the playout buffer and the periodic playout of packets is delayed for some amount of time, called *playout delay*. That means: a playout buffer is responsible for holding each packet within an amount of buffer time so that the end-to-end delay is the same for every incoming packet without excessively delaying the packet. Clearly, the longer the playout delay, the more likely it is that a packet will have arrived before its scheduled playout time. Excessively long playout delay, however, can significantly impair human conversations. There is thus a critical tradeoff between the length of playout delay and the amount of loss (due to late packet arrival) that is incurred. The algorithm that controls this buffer time is called p*layout buffer delay adjustment* a*lgorithm*.

1.5 Problem Specification

This thesis deals with the issues of providing DiffServ in an IP backbone network. These issues raise some general questions as below:

1. How to create a new and simple DiffServ model that
    - is simpler than the existing DiffServ models?
    - collaborates well with the playout buffer delay adjustment algorithms implemented at receiver?
    - produces better end-to-end QoS than the existing DiffServ models?
2. Which playout buffer delay adjustment algorithm should be used at the receiver end for cooperating with the designed DiffServ model?

3. How to design different scheduling algorithms in routers in order to transport IP packets successfully through this DiffServ model?

4. How to evaluate and compare and the performance of my new model and of the existing models?

5. How to improve the disadvantages of my new model and the existing models?

I believe that the problem of network-architecture design is very complex and cannot be solved satisfactorily without dealing with other problems in networks. There are at least following components, which are intricately associated with the problem of network-architecture design:

The first question deals with the problem of network-architecture design, e.g., how to design a new architecture, which is able to simplify the components.

The second question is concerning with the existing playout buffer delay adjustment algorithms: how to choose an appropriate mechanism, which performs a good trade-off between packet delay and loss rate.

The third question implies the problem how to design different schedulers that produce proportional jitter?

For the fourth question, the objective is to establish methods in order to evaluate the quality of these models. It concerns with the choice of network topologies used in my simulations, with the setup traffic parameters, with the performance criterion for comparing the quality of my scheduling algorithms with the existing ones. Based on these methods, the quality of my new model and the existing models are evaluated.

For the last question, it is necessary to establish a combination of my new model and the existing model.

1.6 Contribution of this work

1.6.1 Innovative approaches

This work provides some innovative approaches to the issues addressed above, creating:

- First, a new model for Proportional Differentiated Services – *Proportional Jitter Differentiated Model* (PJDM). Unlike the PDDM model that performs proportional delay, this model provides proportional jitter in the network. Hence PJDM is simpler than the PDDM model because it is not necessary to implement special scheduling algorithms at every router in the network. Furthermore, by controlling the proportional jitter in the network, PJDM is also more efficient than PDDM because it can cooperate well with the playout buffer at receiver in order to produce better end-to-end quality of service. In addition, I analyse the existing playout schemes in order to choose an appropriate adaptation for PJDM. It was decided to choose *Concord mechanism* [Shiv95] for implementing at the receiver in my networks.

- Second, in contrast to existing schedulers that provide proportional delay between classes, designing four new schedulers in order to be implemented in the PJDM model, which facilitates proportional jitter. The first algorithm, which is called *Relative Jitter Packet Scheduling* (RJPS), maintains the short-term jitter and long-term jitter of different classes proportionally. This is a simple mechanism, which can be implemented easily at high-speed routers. The second algorithm, called *Proportional Average Jitter* (PAJ), is also designed for PJDM. This scheduler is simpler than RJPS, and hence easier to implement at network routers based on PJDM model. The two last algorithms (*Adaptive RJPS* and *Adaptive PAJ*, called Adaptive-RJPS and Adaptive-PAJ) are different from their original algorithms. These mechanisms use *Adaptive Jitter Differentiation Parameters* instead of fixed parameters as feedback signal for controlling the jitter ratio between different classes; hence improve the quality of jitter ratio under bursty load conditions.

- Third, focus on the methods of the performance evaluation. In order to analyse and compare different algorithms (as RJPS, PAJ, Adaptive-RJPS and Adaptive-PAJ) in different models (PDDM and PJDM), two methods are proposed. One compares the quality of my scheduling algorithms within only one hop and the other is for comparing the performance of PDDM and PJDM, which will contain different schedulers in a multi-hop network. The results from my simulations will show that two algorithms Adaptive-RJPS and Adaptive-PAJ achieve better quality in terms of jitter ratio than RJPS and PAJ, but they are also more complicated and hence, more difficult to implement in the ‘’real world’’. In addition, the simulations based on the second comparison method indicate that the new model PJDM that uses my new scheduling mechanisms achieve better end-to-end performance than the old model PDDM.

- Finally, design new network topologies based on both PJDM and PDDM models in order to overcome the disadvantages of PJDM and PDDM. I evaluate these new nework topologies and compare them with the existing topologies.

1.6.2 Results

The specific contributions of this work are the followings:

- Analysis about the relative DiffServ architecture and all the existing works on Proportional Differentiated Services Architecture.

- The development of a new model for relative DiffServ-PJDM.

- Analysis of the existing playout buffer delay adjustment algorithms. Among them, the Concord algorithm is used in my new model at the receiver, because of its good trade-off between the end-to-end delay and loss rate.

- The design of four new schedulers, which can be implemented, in my new model PJDM.

- The development of a comparative method to evaluate the quality of these four schedulers.

- Developing a comparative method to examine the performance of the new model PJDM with the existing model PDDM.

- Performance evaluation and comparison of my new four schedulers under different contexts within only one hop.

- Performance comparison of my new model PJDM with the existing model PDDM (by using different scheduling mechanisms) through a multi-hop network. Results show that my model PJDM achieves better end-to-end quality of service than the old model PDDM.

- Design new network topologies based on both PJDM and PDDM for overcoming the disadvantage of PJDM and PJDM models.

1.7 Structure of the Thesis

The overall structure of the thesis is organized as follows:

- Part I (Chapter 2, 3, 4) will present the background information and state of the art of my new research.

- Part II (Chapter 5, 6, 7) will present the new research issue and discuss it in depth.

Chapter 2 gives an overview about general issues, mechanisms and service models for providing quality of service in the Internet. Then mechanisms such as congestion control, traffic shaping, call admission control, resource reservation and service scheduling that are deployed in the service architectures are discussed. The next part of the chapter presents two service models, namely Integrated Service and Differentiated Services model. Advantages and disadvantages of the two models are also be analysed in this chapter.

Chapter 3 continues to focus on the proportional differentiation model, in terms of delay and loss. Existing works on proportional delay and proportional loss are also described. In addition, the chapter concentrates on proportional delay issues by presenting its properties.

Chapter 4 provides an overview about the importance of playout buffer delay adjustment algorithm. The taxonomy of existing adaptation schemes are presented. Furthermore, the trade-offs between playout buffer delay and loss rate are analysed. After analysing the existing algorithms, I decide to use Concord algorithm at the receiver end in my work. The properties and characterizations of the Concord algorithm are also described.

In Chapter 5, I will develop a new model for relative DiffServ, which is called Proportional Jitter Differentiation Model (PJDM). This is a new architecture that provides proportional jitter between different classes. After that I establish the performance evaluation methodology used in order to examine and compare the quality of my new schedulers in PJDM model.

Chapter 6 then describes four new scheduling algorithms are developed for this new PJDM architecture. The first one is called Relative Jitter Packet Scheduling (RJPS) algorithm, and its properties under different context as variable packet size, variable window size, and variable load distributions are also examined. The second one, which is simpler than the previous RJPS algorithm is also created and called Proportional Average Jitter (PAJ) algorithm.

The RJPS and PAJ algorithms use Jitter Differentiation Parameters as fixed variables for controlling and monitoring the jitter ratios at the output link, so that the jitter ratios between different classes stay proportionally with each other. In this chapter, I also present two new variants of these algorithms, which do not use these variables as fixed but adaptive variables. The performance of the two new variants, called Adaptive-RJPS and Adaptive-PAJ are compared to the performance of there original algorithm RJPS and PAJ.

In the last section of this chapter, the performance of the two models PDDM and PJDM using different scheduling algorithms is evaluated and compared in terms of end-to-end delay. This comparison is done based on the methodology described in the previous chapter. Finally, I propose new networks based on both PDDM and PJDM in order to overcome their disadvantages.

Finally, Chapter 7 states some conclusions about the works covered in this thesis and suggests proposals for future works in this direction.

1.7.1 Terms and Definitions

Before going to next chapters, I would like to define some of the important and most frequency used terms throughout this work in order to make it easier to read.

Accounting: The collection of resource consumption data for the purposes of capacity and trend analysis, cost allocation…Accounting management requires that resource consumption be measured, rated, assigned, and communicated between appropriate parties.

Admission control: is a set of actions taken by the network during the call setup phase to determine whether a new flow can be accepted. Call admission control is especially necessary for real-time flows.

Bandwidth: The difference between the highest and lowest frequencies of a transmission channel.

Bandwidth broker: QoS architectures are designed with agents called bandwidth brokers, that can be configured with organizational policies, keep track of the current allocation of marked traffic, and interpret new requests to mark traffic in light of the policies and current allocation

Best-effort service: This service does not make any promise of whether a packet is actually delivered to the destination, or whether the packets are delivered in order or not.

Circuit switching: A method of guiding traffic through a switching centre, from local users or from other switching centres, whereby a connection is established between the calling and called stations until the connection is released by the called or calling station.

Congestion control: is a mechanism used in a network such that the network can operate at an acceptable performance level when the demand exceeds or is near the capacity of the network resources.

Delay or latency: In a network, latency, a synonym for delay, is an expression of how much time it takes for a data packet to get from one designated point to another.

Domains: A DiffServ-capable domain; a contiguous set of nodes, which operate with a common set of service provisioning policies and PHB definitions.

Dropping: the process of discarding packets based on specified rules.

Egress node: A DiffServ boundary node in its role in handling traffic as it leaves a DiffServ domain.

Flow classification: to identify the flow to which a packet belongs.

Flow: is a stream of packets belonging to the same application sessions.

Ingress node: A DiffServ boundary node in its role in handling traffic as it enters a DiffServ domain.

Links: are direct connections between end users or routers.

Marking: A process of setting the DiffServ code point in a packet based on defined rules.

Metering: the process of measuring the temporal properties (e.g., rate) or a traffic stream selected by a classifier. The instantaneous state of this process may by used to affect the operation of a marker, shaper, dropper, and or may be used for accounting and measurement purposes.

Multiplexing (MUXing): Combining/mixing two or more information channels onto a common transmission medium**.**

Network congestion occurs when packets arrive at an output port of a router faster than they can be transmitted.

Packet classification: is the process of sorting packets based on the contents of packet headers according to defined rules.

Packet headers: Information such as the source or destination addresses and the application port numbers is stored in the packet headers and serves as the determination of the path and the service that packets receive in networks.

Packet switching networks: A switched network that transmits data in the form of packets.

Packets: In general, if an information unit can be digitally represented, it can also be assembled into packets, and transferred through a network

Per-hop behavior: a description of the externally observable forwarding treatment applied at a single node to a behavior aggregate.

Playout buffer delay: Enforced delay at the receive side for interactive real-time communication in order to smooth delay variation.

Policing: The process of delaying or discarding packets of a traffic stream (by a dropper) in accordance to the state of a corresponding meter enforcing a traffic profile.

Policy control: the process of discarding packets (by a dropper) within a traffic stream in accordance with the state of a corresponding meter enforcing a traffic profile.

Provisioning: is a policy, which defines how traffic are configured on DiffServ boundary nodes and how traffic streams are mapped to DiffServ behavior aggregates to achieve a range of service.

QoS: On the Internet and in other networks, QoS (Quality of Service) is the idea that transmission rates, error rates, and other characteristics can be measured, improved, and, to some extent, guaranteed in advance. QoS is of particular concern for the continuous transmission of high-bandwidth video and multimedia information. Transmitting this kind of content dependably is difficult in public networks using ordinary "best-effort" protocols.

Resource reservation: reserves different resources for a flow at the network node so that its quality of service is maintained.

Route pinning: Receivers or routers can request that once the reservation along a path has been set up, the path does not automatically change when a better path is available.

Routers is a physical device that joints multiple networks together.

Routing lookup: For forwarding packets, it is necessary to search and find the appropriate output interface, and this operation is called routing lookup, which is the main and most complex operation in routers today.

Routing: The process to determine and prescribe the path or method to be used for forwarding messages.

Scheduling: is the policy to choose from multiple packets, which exist in a buffer and which share a common outgoing link, the next one for service.

Service classes: Class of service is the ability of a service provider to categorize user's application into separate classes, each class correspond to one type of service requirement.

Service level agreement: A service contract between a customer and a service provider that specifies the forwarding service a customer should receive. A customer may be a user organization (source domain) or another DiffServ domain.

Service profile: the service contract between a customer and the DiffServ network or ISP is called the service profile. A service profile is usually defined in absolute bandwidth and relative loss.

Shaping: is the process of delaying packets within a traffic stream to cause it to conform to some defined traffic profile.

Signalling protocol: A protocol responsible for transfering the requirement of the application and traffic characteristics on quality of service from senders to routers along the path of this flow and to the receiver.

Speedup: in a switching fabric is the term that describes how much faster the switching fabric operates than the native line rate it supports.

Statistical multiplexing: is a method of making the most efficient use of the bandwidth available for transponder or cable transmission, whilst maintaining a quality of service across all of the multiplexed channels. It is the process of dynamically allocating bandwidth where it is needed most.

Switching fabric: Switching fabric is the combination of hardware and software that moves data coming in to a network node out by the correct port (door) to the next node in the network.

Traffic conditioning: control functions that can be applied to aggregation, application flow, or other operationally useful subset of traffic, e.g., routing updates. These may include metering, policing, shaping and packet marking. Traffic conditioning is used to enforce agreements between domains and to condition traffic to receive a differentiated service within a domain by marking packets with the appropriate code point in the DiffServ field and by monitoring and altering the temporal characteristics of the aggregate where necessary.

Traffic profile: a description of the temporal properties of a traffic stream such as rate and burst size.

WDM: Fibre-optic modulation scheme where data channels are put on different light wavelengths (or colours).

Work conserving: A property of a scheduling algorithm such that it services a packet, if one is available, at every transmission opportunity.

Chapter 2 Background

In this chapter, I firstly provide an overview of the two switching techniques, namely circuit switching and packet switching. It then presents the basic mechanisms of the Internet. The next part of the chapter presents two service models, namely Integrated Service and Differentiated Services model. Advantages and disadvantages of the two models are also analysed in this chapter. Then such mechanisms as congestion control, traffic shaping, call admission control, resource reservation and service scheduling that are deployed in the service architectures are discussed. After analysing these operations, I present some existing work in DiffServ architecture. Finally, the importance of playout buffer delay adjustment algorithm at the receiver is also discussed.

2.1 Circuit Switching vs. Packet Switching

Generally, there are two major important switching techniques that are used in communication networks: *packet switching* or *circuit switching*. A very good example for circuit switching is the telephone network, which was developed more than 100 years ago. In a telephone network, a circuit will be established for any two end points, which want to communicate with each other. The two end points remain connected with each other until the circuit is switched off.

In packet switching networks, information streams are transmitted in form of small packets, which are switched and transferred based on the information contained in packet headers. At receiver end, packets are put together for receiving the original information. An example of such networks is the Internet Protocol (IP) [Clark88] network.

Compared to circuit-switching network, a packet-switching network is capable of *statistical multiplexing*, which means that the active traffic sources can use any additional capacity made available by the inactive traffic sources. Statistical multiplexing can significantly increase network utilisation. More concretely, recent research has demonstrated that the ratio between peak and average rate is 3:1 for audio traffic, and as high as 15:1 for data traffic [Robert].

Unfortunately, this statistical multiplexing has its own disadvantage: *congestion*. *Network congestion* appears when packets coming from multiple inputs arrive at an output port of a router faster than they can be transmitted. In such a case, the network will have to buffer and in some cases to drop some traffic. Furthermore, the sender is required to cooperate with the network to reduce the congestion by decreasing its rate upon detection of congestion.

2.2 IP Network Model

The Internet has a hierarchical architecture where major backbone networks are connected to smaller regional networks, which then are interconnected with even smaller local networks. The most important service of today's IP network is to deliver packets between different nodes in the Internet with reasonable QoS [Gup99]. As noted before, it is the router, which makes this job available. Each router needs to have at least two interfaces to interconnect networks. In order to forward packets the router uses the destination address in the packet's header. That is why each router needs to maintain a table, called forwarding table, which maps an IP address to an interface attached to the routers. Furthermore, routers run a routing protocol based on a *distributed algorithm*. The main function of this algorithm is to enable routers to learn to know the reachability of any host in the Internet along a *good path*, or the shortest path. Hence, a packet will be transferred from its source to the destination theoretically via the shortest path.

As mentioned previously, a router has in general a set of input interface(s) and a set of output interface(s) or both. Packets will reach the router at input interfaces and leave the router at output interfaces. A switching fabric is responsible to transfer packets from inputs to outputs [Patr94]. For each switching fabric, there is a main parameter called the *speedup*. *Speedup* in a switching fabric is the term that describes how much faster the switching fabric operates than the native line rate of supported inputs/outputs.

In routers, packets could either be stored at the input interface or at the output interface or at both before going to the output link. According to the place where the packets are stored, there are three possibilities of routers, so they are classified as *input queuing, output queuing or input-output queuing*.

In an output-queuing router, packets are transferred immediately to the corresponding output when they arrive at the inputs. Packets will be queued and scheduled at the output interfaces only. Most of the analytical studies assume an output-queuing router model because of the simplicity of this model. "Simple" routers additionally need a speedup of *n*, where *n* is the number of input links. Routers have to work at this speed in cases when all inputs need to process packets simultaneously for the same output. Since inputs do not have buffers, the output must receive *n* packets from this inputs simultaneously, which I call a speedup of *n*. In high-speed networks, high speedups lead to technically costly output-queuing routers. That is why the market prefers input-output queuing types.

Input-output queuing routers use to store packets at the input, and the speedup of the switching fabric is decreased significantly. Unfortunately, this advantage at the same time increases the complexity of the router: since only the output has

complete knowledge of how packets are scheduled, complex and distributed algorithms in order to control the packet transfer from input to output have to be implemented. Furthermore this makes the routers much more complicated to be understood. To conclude, output-queuing routers are more tractable for analysis; the input and input-output queuing routers are more scalable and therefore easier to design.

Recently, researches in [Chua99, Stoika1] have proved that an input-output queuing router, which has an internal speedup of 2 only, is able to emulate a large class of scheduling algorithms of output queuing routers. That is why, at least in principle, it is possible to build scalable input-output queuing routers, which can emulate the behavior of output-queuing routers. For this reason in this work I will focus on the output-queuing router architecture only.

Figure 1 shows the architectural layout of an IP router [Kesh98], especially the queuing components. Technical parameters are the reason why packets can be delayed and lost. Queuing delay and packet loss (locally or per-hop) are substantial reasons for performance-degradation factors of each IP router. If routers have sufficient *forwarding resources* at their disposal, packet flow can receive adequate performance from routers, and on the other hand, if routers are not able to transfer packets fast enough (packets then have to stay in routers) then performance will be degraded.

Major *resource types* within a router are on the one side the *rate* at which it can forward and transmit packets (usually called *bandwidth*) and on the other side the size of packet *buffers* in the router queues. In summary, traffic performance decreases when there is *contention* for either of these resources.

A disadvantage of the IP network, which provides best-effort service only, is the fact that all packets are treated equal in terms of bandwidth, delay or loss etc., even though not every application has the same requirements from the network. A packet of an interactive voice session has the same performance considering delay as packets of the file transfer protocol application.

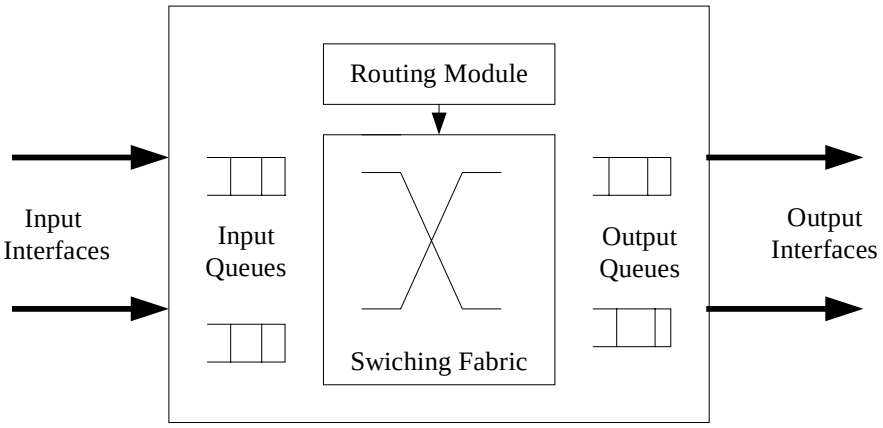


Figure 1 Architecture of a conventional IP router

Nowadays demands for more than best-effort services are coming up - towards e.g. *guaranteed services* and *differentiated services*, while today's Internet is evolving to a global architecture. *Guaranteed services* are able to ensure specific performance parameters such as bandwidth and delay on a per-flow basis. For example it should be guaranteed that all packets of a flow receive a delay smaller than a specified threshold, which implies that this specific flow receives at least a certain amount of bandwidth. This kind of service would provide support for new applications such as IP telephony, video-conferencing, and remote diagnostics. *Differentiated services* are able to provide different services, in terms of bandwidth, loss, and delay for different traffic classes. As an example it can be useful to allocate twice as much bandwidth on every link in networks to a single organization than to another organization.

Providing these services in packet-switched networks such as the Internet has been one of the major challenges in the network research over the past decade. To address this challenge, a plethora of techniques and mechanisms have been developed. Among these techniques I will examine *Integrated Services* and *Differentiated Services* architectures.

2.3 Integrated Services Architecture

There is a need for the provision of new services, which are more sophisticated than best-effort service because new applications such as IP telephony, video-conferencing and distributed games are currently growing in the Internet. These new applications have special requirements on performance parameters, such as delay and bandwidth compared to previous applications such as file transfer. For example, interactive applications demand an end-to-end delay smaller than approximately 100 ms. In global networks the propagation delay can be as much

as 100 ms. Thus, coping with these tight delay requirements is a very challenging task.

In order to support these new applications, the Internet Engineering Task Force (IETF) proposed a new service model called Integrated Service or IntServ [Brad94, Wrocl10, Whit97]. This architecture was developed within the IETF in the mid-nineties and has its roots in earlier research results [Ban96, Par93, Cruz1, Cruz2, Enw95]. IntServ provides two services besides the traditional best-effort service: *Guaranteed* and *Controlled-Load* services.

- **Guaranteed Service:** Among the services applied by IntServ and Internet, this service is the strongest semantic service so far [Shenk12] and is able to provide per-flow bandwidth and delay guarantees. Particularly, guaranteed service guarantees a minimum amount of bandwidth to a flow, even in a worst-case situation. Thus delay is bounded under a given arrival process of the flow. Hence, Guaranteed service is an ideal service for supporting real-time applications such as IP telephony, or videoconference etc.

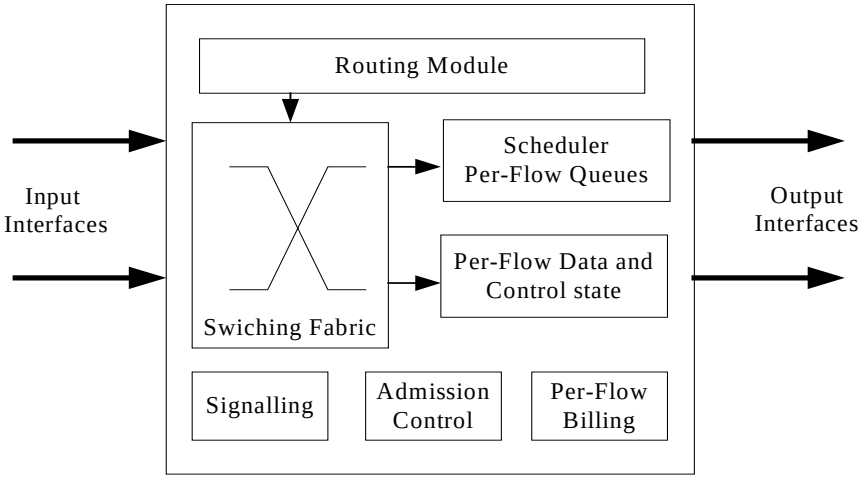


Figure 2 Architectural layout of an IntServ router

- **Controlled Load Service:** The controlled Load Service is a less strict service provided in the IntServ framework. Wroclawski has defined Controlled-Load service in [Wrocl11] as: *''tightly approximates the behavior visible to applications receiving best-effort service under unloaded conditions from the same series of network element''*. The Controlled-Load service guarantees that packet loss is not significantly larger than basic error rate of the transmission medium and the end-to-end delay experienced by a very large percentage of packets does not greatly exceed the end-to-end propagation delay. The main idea of this type of

service is to provide better support for broad class of applications that have been developed in the Internet. Some examples of such applications could be adaptive and real-time applications, such as video and audio streaming.

The services mentioned above have been standardized in the IntServ architecture and they include a maximum end-to-end delay [Shenk15], a sufficiently low loss rate [Wrocl11] and minimum end-to-end rate [Bak96]. They are based on the IntServ architectural principle: service differentiations are accomplished with *per-flow end-to-end resource reservations*. These critical issues have important implications on the strengths and the weakness of IntServ, which are discussed next.

IntServ has a *flow-centric* property. That means the service-differentiation router mechanisms are applied to individual flows, i.e., to streams of packets, which belong to the same application session. In an IntServ router (Figure 2) it is necessary to perform the following operations on packets of a flow:

- *Flow Classification*, to identify the flow in which a packet belongs to [Lak98, Srin99]

- *Scheduling*, to provide a certain delay deadline or rate to a flow [Step99, Stil98].

- *Buffer Management*: to allocate a number of buffers to a flow. [Suter98]

- *Traffic Shaping or Policing*, to control certain traffic characteristics of a flow [Chao92, Step99].

- *Admission Control* determines whether the requested QoS to a flow can be granted without affecting other flows in the network [Jam97].

- *Resource Reservation* is performed on a per-flow basis to provide guarantees to the established flows in the network [Whit97]

These operations, such as flow classification, scheduling, buffer management and traffic shaping or policing play an important role for the *data-plane* per-flow state of a router. Besides per-flow operations at the data plane described above, a router has to maintain and to process information for each flow at the *control-plane*. The *control-plane* of IntServ is responsible for *resource reservations* and *signalling operations* as well as for *per-flow accounting* [Edel99] and *policy control* [Raj99].
In addition, IntServ is an end-to-end architecture. In order to provide an end-to-end service guarantee it requires the cooperation between network providers. If

one of these networks is not able to provide this service type requested by a user then it will not be able to provide this service guarantee end-to-end.

Additionally the hierarchical architecture of the Internet significantly contributes to the difficulty to implement an IntServ model. Hence the establishment of the required multilateral agreements for enabling end-to-end services is a very difficult task in practice [Ferg98].

Resource reservation mechanisms in IntServ as a consequence lead to a certain amount of resources (bandwidth and buffer) to be allocated to a flow before the session starts and have to be maintained during the session. Resource reservations are realised by a *signalling protocol*, which is responsible to transfer the requirement of the application and traffic characteristics on quality of service from senders to routers along the path of a flow. If there is a single router on the path, which is not able to satisfy the necessary conditions then end hosts are informed about the deny of the connection by this signalling protocol. This operation is called *admission control* [Jam97]. The signalling protocol called Resource Reservation Protocol (RSVP) [Zhang2, Brad97] has been designed for this purpose. Its substantial per-flow processing at the control-plane raised concerns about the IntServ scalability.

The possibility of providing strict QoS guarantees for IntServ comes along with costs: complexity at the router, hence its deployment by network providers has been quite limited. In addition to the issues of inter-domain deployment and RSVP overhead as being discussed earlier, in IntServ there are also issues of *scalability* and *manageability* [Mank97]. As mentioned before scalability concerns are raised because the IntServ requires that routers maintain and process data and control state for every active flow. Gigabit or terabit links carry millions of simultaneously active flows, making it difficult to build IntServ-capable routers. Next-generation routers will be able to accommodate millions of flows by maintaining per-flow state at the edge routers only [Stoika1, Stoika2], adding to the term flow a more coarse granularity [Kum98], or they will use approximations of the ideal algorithms [Shree95].

Because of the conventional opinion that an IntServ network is harder to install, to debug and to operate there are raised severe manageability concerns (IntServ requires admission control, signalling, per-flow accounting, and the configuration of several router mechanisms). Additionally routes may dynamically change in an IP network. Routing changes usually do not occur in the Internet very often, but there are certain links in which this happens quite regularly [Paxs96]. If a certain end-to-end QoS is assured to a flow, the architecture should be able to either forbid routing changes for that flow (*route-pinning*) [Guer97] or to reserve the required forwarding resources in the flow's new path while the session is in

progress,. Both operations are hard to implement in practice. Similar complexities also arise when IntServ flows need fault tolerance [Dov4].

Another factor contributes to the weak deployment of IntServ – first, it requires new Application Programming Interfaces (APIs) at the end-hosts (especially for multimedia applications [Gop98]) and - second it can provide true end-to-end service guarantees only if similar resource-reservation mechanisms are deployed in the servers [Bha99] and the end-host operating systems [Yau97].

These problems were the reasons why the IETF and the research community considered simpler and more scalable service-differentiation architectures such as the technologies discussed below.

2.4 Differentiated Services

In order to overcome the disadvantages of IntServ a new architecture for the Internet has been proposed [Nich98], which is called Differentiated Services (DiffServ). Initially the purpose of DiffServ was a more scalable, manageable and easily deployable architecture for service differentiation in IP networks.

It is very important to note that in DiffServ individual flows with similar QoS requirements can be *aggregated* in larger traffic sets (or *class*). All packets in this traffic set are submitted to the same ‘’*forwarding behavior*’’ in routers. A traffic set is the minimum level of granularity in a DiffServ network in terms of service differentiation. Each traffic set uses a certain class or *Per-hop Behavior* (PHB). A PHB is identified by a short label (currently six bits) in the IPv4 header, which is called *Differentiated Services Code Point* (DSCP).

Providers and customers negotiate agreements with respect to the services to be provided at the customer/provider boundary. These agreements are commonly referred to as *Service Level Agreements* (SLA). The subject of the SLA, which provides the technical specification of the service, will be referred to as *Service Level Specification* (SLS).

Individual flows are aggregated into traffic sets at the *edges* of a DiffServ network. The *edge routers* can be the host-network interface or the router that connects a flow-aware network to a DiffServ network. The *mapping* from individual flow to traffic sets is called flow classification.

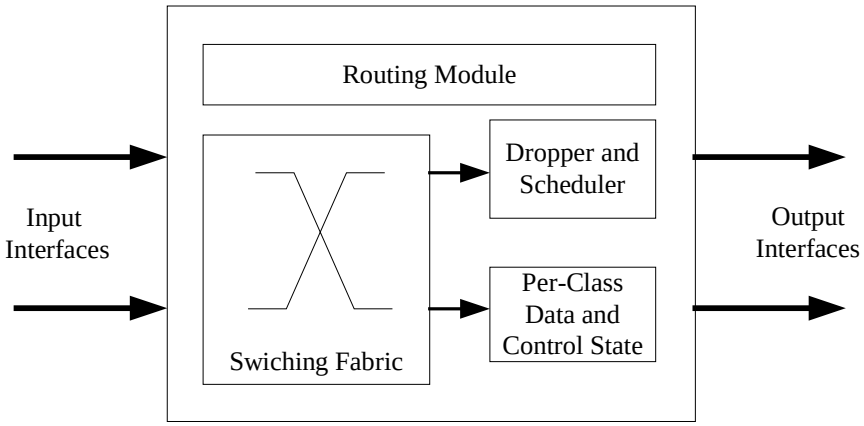


Figure 3 Architectural layout of a DiffServ router

Figure 3 shows the main architectural blocks in a DiffServ router. The aggregation of packet flows into different classes in the DiffServ architecture improves the scalability issue in IntServ architecture, since routers need to maintain only states for a few classes. Similar to IntServ, the operations for processing the packets in a DiffServ-capable router could be also scheduling, classification, buffer management, traffic shaping or policing etc., but these operations become much simpler and faster than IntServ. In addition the management of networks becomes much simpler because operators have to control and monitor just the state of a few classes rather than millions of flows. Third, network *pricing/accounting* operations are simpler, because users do not need to be billed for each session, or flow, but for the aggregated traffic. In addition, DiffServ could be broadly classified to *absolute* (or *quantitative*) and *relative* (or *qualitative*) *service differentiation*. Obviously, the fundamental drawback of aggregation is that the network cannot guarantee QoS to a flow. Obviously, it is difficult for a DiffServ network to guarantee QoS quantitatively in an end-to-end manner.

An *absolute service* model in DiffServ is a model that provides a quantitative guarantee level for each class, such as a minimum bandwidth, a maximum delay or a maximum loss rate. This model demands some form of admission control or policing for supervising users so that they do not send traffic at a higher rate than their *traffic contracts*.

Contrary to the above service model the *relative service* model guarantees that a lower class gets worse QoS than a higher class [Dov1]. There are no precise values of QoS, which are defined for each class; the values of QoS vary with load conditions and service differentiation mechanisms implemented in the network. For users, who have absolute QoS demands but make use of the relative model, it is possible to meet these requirements by dynamically adjusting the class that reaches their QoS and pricing constraints. This relative differentiation model does

not require resource reservations, admission control, signalling or fixed routing, and so it is considered as simpler to manage and deploy.

An example of the relative DiffServ is the *Proportional Differentiated Services* model, which provides *proportional*-performance metrics for bandwidth, delay or loss for different classes. I will focus on this model in Chapter 3.

2.4.1 Architecture

There is a distinction between edge and core mechanisms in a Differentiated Services network (Figure 4).

- **Edge Mechanisms**: Edge routers can be categorized into *ingress* and *egress* nodes. Traffic enters the DiffServ domain at an ingress node and leaves the domain at an egress node. In ingress nodes the operations such as *packet classifications* and *traffic conditioning* are implemented at per-flow level. The packet-classification operation identifies the class of a packet. Traffic conditioning contains *metering, marking, shaping and dropping*. Metering is an operation, which measures traffic. If this traffic is not compliant to a traffic profile then marking, shaping and/or dropping operations are invoked. In order to set the DS field of each packet to an appropriate DS code point for demoting out-of profile traffic to a different PHB, or for ensuring that only valid code points are used within a domain, the function of marking is defined. A shaping operation shapes the input traffic in a manner that the submitted traffic conforms to the agreed traffic profile. A dropping operation is necessary for dropping the out-of-profile packets.
  In egress node, it is expected that egress nodes remark, police and shape the outgoing traffic so that it conforms to an SLA of the next DiffServ domain.

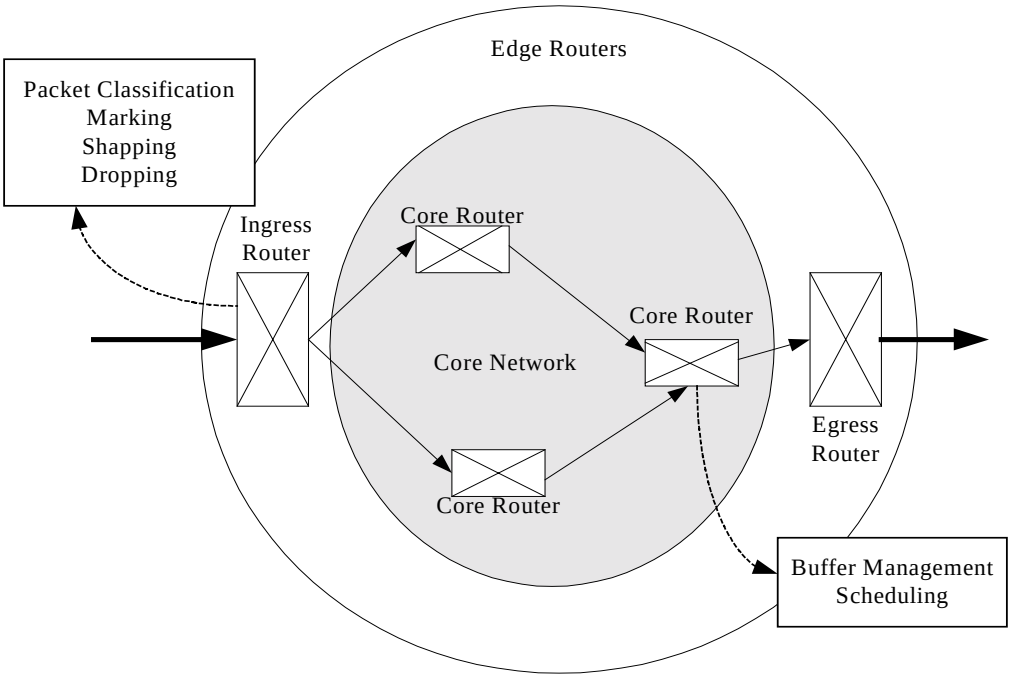


Figure 4 Architecture of a Differentiated Services network

- **Core Mechanisms**: Two mechanisms alternatively are implemented at the core router: *buffering* and *scheduling*, both in order to guarantee that service requirements offered by SLS are met. Unlike at the edge routers, these operations are done at a per-class level.

2.4.2 Complexity

As written above DiffServ routers are developed to overcome the scalability problem raised in the IntServ model. DiffServ supports a limited class of service only. Classification, traffic shaping, and other per-flow operations are done at edge routers of the DiffServ network. At the core network, where the amount of traffic is large, routers are responsible for traffic aggregations of classes of service only.

I believe that the complexity of a DiffServ router depends on different factors, such as on the lookup operation, packet classification, scheduling or on buffer management. I will now focus on the complexity of the scheduling- and buffer-management algorithms because they are very much related to my research.

2.4.2.1 Buffer Management

Buffer management is used for monitoring buffers at input or output interfaces in an IP router. Practically, buffers have limited spaces. When buffers are full, the arriving packets should be dropped. That means there is a possibility of packet loss in routers.

Key mechanisms of buffer-management algorithms are the *backlog controller*, which specifies the time instances when traffic should be dropped, and the *dropper,* which specifies the traffic to be dropped. In an IP network, the buffer management algorithms have to focus mainly on different issues, such as: avoid congestion, reduce the packet transfer delay and keeping the queue length at low levels.

Backlog Controller: Among backlog controllers for IP network, Random Early Detection [Sally2] is probably the best-known algorithm. RED was motivated by the goal to improve TCP throughput in highly loaded networks. RED operates by probabilistically dropping traffic arrivals, when the backlog at a node grows large. RIO [Clark98] and Multiclass RED [Card02] are extensions to RED, which aim at class-based service differentiation. Both schemes have different dropping thresholds for different classes in order to ensure loss differentiation.

Droppers: The simplest and most widely used dropping scheme is Drop-Tail, which discards arrivals to full buffer. For a long time, discarding arrivals was thought to be the only viable solution for high-speed routers. Recent implementation studies demonstrated that more complex dropping schemes, which discard packets which are already present in the buffer (push-out), are viable design choices at high data rates [Kron91].

Push-out techniques include mechanisms like Complete Buffer Partitioning [Lin91] and Partial Buffer Sharing [Kron91]. CBP assigns a dedicated amount of buffer space to each class and it drops traffic when this dedicated buffer is full. PBS uses a partitioning scheme similar to CBP; but the decision to drop is done after having looked at the aggregated backlog of all classes.

Simple buffer-management algorithms could be implemented by using a single queue shared by all classes. With these schemes, the routers could provide only simple network services. For more sophisticated network services, such as per-class bandwidth and delay guarantees, it is necessary to implement complex algorithms in routers, which could demand and manage separate queue for each class. Buffer management algorithms become very complicated if the router has to choose from which queue it has to drop a packet. For example, an algorithm that implements a policy that drops the packet from the longest queue has *O(log n)* complexity, where n is the number of non-empty queues. However, in practice,

this complexity can be significantly reduced by grouping the queues, which have the same size or by approximating the algorithm [Odl99].

2.4.2.2 Packet Scheduling

In routers that maintain a per-class state packet scheduling is accomplished in two steps: The first step is to select a class that has a packet to send, and the second step is to transmit a packet from the class's queue.

Normally, scheduling contains two types: *work conserving* and *non-work conserving*. The work-conserving packet-scheduling algorithm will always transfer packets to certain outputs if there is at least one packet in the system. In other words output links are busy as long as there are packets in queues. Opposed to a work-conserving scheduler, non-work conserving scheduling algorithms could keep output idle even though there are packets destined for that output.

It is also very important to know that a scheduling discipline has to be designed in order to ensure that each class gets a fair access to network resources and in order to prevent a bursty class from consuming more than its *fair share* of output port bandwidth, i.e., the bursty class has bad influence on other classes. Such mechanisms belong to the class of *fair-scheduling* algorithms.

A simple scheduling algorithm is the FIFO (First In - First Out) mechanism. In FIFO queuing all packets are treated equal by placing them into a single queue, then servicing them in the order of their arriving time.

Priority Queuing (PQ) is the basis for a class of queue-scheduling algorithms, which are designed to provide a relatively simple method of supporting Differentiated Services classes. In classic PQ [Zhang1] packets are first classified by the system and then placed into different priority queues. Packets are scheduled from the head of a given queue only if all queues of higher priority are empty. Within each of the priority queues packets are scheduled in FIFO order.

Weighted Fair Queuing (WFQ) was developed independently in 1989 [Dem90]. It supports flows with different bandwidth requirements by giving each queue a weight that assigns it a different percentage of output port bandwidth.

In Weighted Round Robin queuing WRR [Fran93], packets are first classified into various service classes (for example, real-time, interactive, and file transfer) and then they are assigned to a queue that is specifically dedicated to that service class. Each of the queues is serviced in a round-robin order. Weighted round-robin queuing is also referred to as *class-based queuing* (CBQ) or *custom queuing*.

WRR queuing supports the allocation of different amounts of bandwidth to different service class by either:

- allowing higher-bandwidth queues to send more than a single packet each time it is visited during a service round, or

- allowing each queue to send a single packet each time it is visited, but to visit higher-bandwidth queues multiple times in a single service round.

Many of simple algorithms as FIFO could be easily implemented by constant-time algorithms, i.e., algorithms that take *O(1)* time to process each packet. On the other hand, more complex scheduling algorithms such as Weighted Fair Queuing are not so easy to implement in routers. In general, the algorithms to implement these disciplines assign to each class a unique parameter that is used to select the class to be served. Examples of such a parameter are the class's priority, and the deadline of the packet at the head of each queue. Class selection is usually implemented by selecting the class with the largest or the smallest value. This could be accomplished by maintaining a priority queue data structure in which the time complexity of selecting a class is *O(log n)* where *n* represent the number of classes in the queue.

Non-work conserving algorithms can be more complex than work-conserving disciplines, because it is necessary to maintain the time information for each class. The goal of this time information is to determine the time when a class with a non-empty queue is eligible to send packets. The packet at the head of the queue will be transmitted if its eligible time is smaller or equal to the system time. In some cases in order to implement such scheduling algorithms it is necessary to have two elements: a rate controller, which is responsible to buffer packets until they become eligible, and a work-conserving scheduler that chooses the flow's packet to be transmitted based on the first parameter. Because the rate controller is implemented normally by constant-time algorithms, the total complexity of selecting a packet is generally dominated by the scheduling algorithm.

Once a class is selected, one of its packets is transmitted - usually the packet at the head of the queue. After that, the parameter(s) associated with the class are eventually updated.

In summary, packet classification is the most complex operation, compared to other router operations. The algorithms to solve this problem require at least *O(logn)* time and $O(n^F)$ space or, alternatively, at least $O(\log^{F-1})$ time and *O(n)* space, where *n* represents the number of classes, and *F* represents the number of fields in a filter.

In contrast to packet classification most buffer management- and packet-scheduling algorithms have *O(n)* space complexity and *O(logn)* time complexity. By trading the resource utilisation for speed the time complexity to *O(loglogn)* or even *O(1)* can be reduced.

2.4.3 Previous Works on Differentiated Services

The two DiffServ models which received major attention so far are the *Virtual Leased Line* (VLL) service and the *Assured Service*. I will analyse and present them next, discussing their advantages as well as their disadvantages.

2.4.3.1 Virtual Leased Line (VLL) Service

Van Jacobson [Jac99] proposed the model of the VLL service, or Premium Service. It was the first model, which was designed in DiffServ. VLL service focuses on guaranteeing peak-bandwidth services with marginal queuing delays and losses. Therefore this service is similar to a leased line in circuits-switched networks. At the ingress router the network controls the peak rate of VLL traffic (*R*) that is contracted between service provider and customers using traffic shaping at edge routers. The shapers need to verify whether packet bursts are delivered into networks. VLL traffic could not exceed this rate at network ingress. The rate *R* will be provisioned and reserved in network cores along its path, this traffic is classified with a highest priority. As a major benefit of the VLL service I can state that users will not experience any considerable queuing delays or losses.

Unfortunately research in [Charn00] shows that this objective of the VLL service can not always be realised: Even though an amount of VLL traffic is provisioned at network edges and reserved in network cores with highest priorities, traffic burst could exist somewhere in networks. The authors explained the reason of this problem as follows: multiplexing of VLL traffic from different input interfaces in core routers can make traffic become very bursty, and these bursts could cause queuing delays and packet losses. The size of burstiness is decided by VLL load, numbers of hops of VLL macro-flows, and shaping parameters at network ingress. Therefore in order to guarantee low queuing delays, the VLL load has to be a moderate portion of the network capacity only [Stoika4]. In certain cases, some experiments [Ferr00] of the VLL model do produce inadequate quality with very high queuing delay and losses. It does not satisfy the condition of VLL service. Latest research [Guer00] improved the performances of the VLL model with re-shaping at boundaries between network domains in order to provide constant bandwidth with low delay and low loss service. Unfortunately, re-shaping that requires some data about states of micro-flows could not yet be implemented in high-speed links due to its complexity.

The VLL service requires some form of semi-static bandwidth reservations, set up by a bandwidth broker protocol or agent in each domain [Jac99]. Between domains, reservations should also be accepted upon and harmonized with agreements of individual bandwidth brokers [Terz99]. The bandwidth broker is a centralized control point for monitoring and controlling bandwidth utilisation and reservations of links within a network. This approach poses concerns about the scalability and fault-tolerant ability of the network [Stoika4]. It is still difficult to implement a complex bandwidth-broker architecture i.e. a distributed broker. Another drawback of the VLL service is that for holding VLL traffic in links, where bandwidth reservations have been set-up, it is necessary to have some form of route pinning even though the route changes could be found in IP networks.

Despite all the disadvantages listed previously, the VLL service is one of the best DiffServ models in the Internet. There already exist some routers that are able to provide VLL service. Furthermore, there have been some experiments with the VLL in Qbone [Teit99] and in other networks.

2.4.3.2 Assured Service

This model, which is originally called Expected-Capacity framework, has been introduced by David Clark [Clark98]. The idea of the original Assure Service is quite straightforward. A sender requires a certain bandwidth from networks, let us say *R*. If the bandwidth generated by the sender is smaller than *R*, this traffic will be marked as IN traffic. Otherwise network marks it as OUT traffic. This marking is used in case of congestions in order to distinguish the treatment of the IN and OUT traffic: the IN traffic will be dropped with smaller probability than the other traffic. Accordingly, when network load is low, higher throughput than the profile R of users is accepted, but users are limited to the IN traffic if congestion occurs.

The most important idea of this model is the guarantee of no-dropping treatment for the IN traffic so that each user will get a contracted bandwidth profile. In order to realise this idea networks need to provide resources sufficiently in advance so that IN traffic will not be dropped. Network providers have to reserve the profiles of different users and different routes that IN traffic passes through. More concretely, this reservation is done for reserving an adequate bandwidth in each network link. Furthermore, the Assured Service model was studied in the context of TCP transfers [Clark98], especially on specific marking procedures for TCP traffic.

However, recent research shows some difficulties in designing provisioning algorithms, which possess a service quality with large spatial granularity and high-resource utilisation [Stoika3]. Moreover, it is also necessary to have some

forms of route-pinning for implementing Assured Services because the provision assumes fixed routing and steady load in each network link.

In conjunction with TCP transfers [Yeom00] and [Sahu00] showed that in some cases, it is impossible to provide a certain throughput of the Assured Service to a TCP connection, even though networks are well provisioned before. The Assured Service has been also analytically studied in [May99, Sahu99] with simple queuing models in order to quantify assurance levels of the provided bandwidth profiles.

2.5 Receiver

The Internet and other packet networks are used to transport audio and video streams, supporting applications such as conferencing and telephony the total delay experienced by each packet is a function of variable delays due to physical media access and queuing in routers and switches, in addition to fixed propagation delays, even though the routers implement such complicated schedulers and buffer management algorithms. Variation of delay (*jitter*) is a major disadvantage for streams of multimedia packets, because of its influence on qualities of audio-visual applications. In order to smooth these variations, there are some buffers, called *playout buffers* that are responsible of queuing and holding each packet within an amount of buffer time. The amount of buffer time, or playout-buffer delay, helps to compensate network delay variances without excessively delaying the playout.

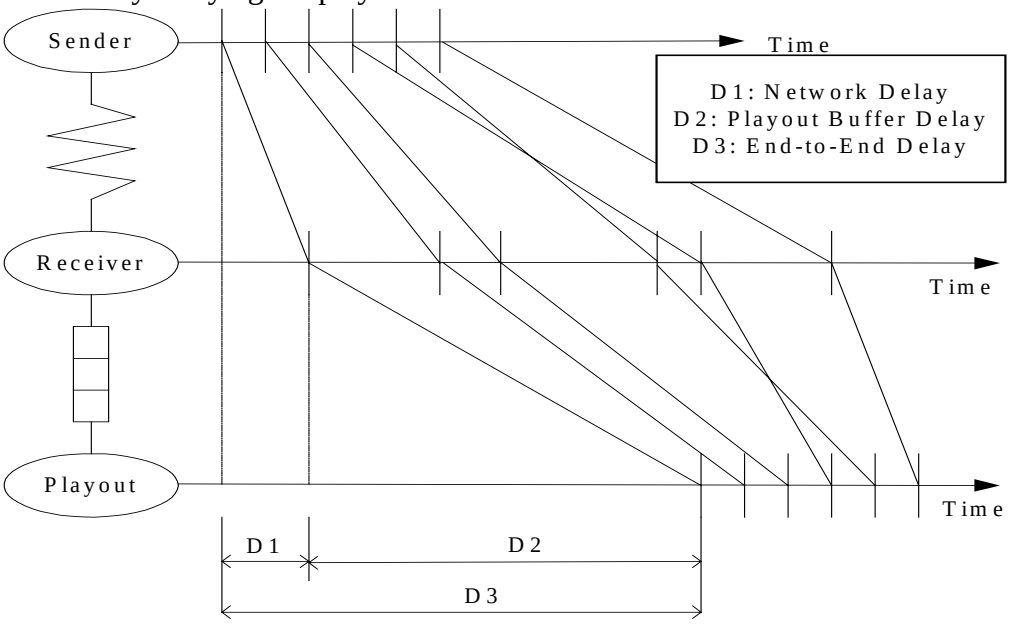


Figure 5 Playout buffer delay adjustment algorithm

If every packet is buffered such that the sum of its network delay and its buffer delay is equal to the maximum network delay, the receiver will reproduce a jitter-free play back. This is called *playout buffer delay adjustment* algorithm (Figure 5).

If the inter-packet delay exceeds the buffer time the buffer will starve and decoders will not have any packet to play. Packets that arrive too late are considered to be lost. The amount of buffer time could be determined manually, or adaptively. Clearly, the longer this delay the more packets will arrive before their scheduled playout time and the better the jitter compensation will be. A good playout scheme should be able to trade-off playout delay and packet-loss rates in order to packet voice to be successful.

Common playout-buffer delay-adjustment algorithms are classified into two broad approaches. *Fixed approaches* are the mechanisms whose range of delay is predictable and which use static buffer size and schedule. *Reactive approaches* that are generally used in the Internet will use instantaneous jitter for dynamically adjusting the buffer size and playout the packet to alleviate the lateness. Fixed playout-buffer delay-adjustment algorithms have known buffering delay but they produce potentially large packet latency while reactive mechanisms do improve the loss rate. But this improvement comes at the expense of potentially very high buffering delays.

Chapter 3 Proportional Delay and Loss

This chapter provides an overview on Proportional Differentiation Model, in terms of delay and loss. It also presents a glance at the existing works on these two models. In addition, I continue to focus on the properties of the Proportional Delay Model because it is related to my new architecture – namely the Proportional Jitter Model.

3.1 Relative Differentiation Condition

As noted in the previous chapter, there exist two distinct models in the DiffServ architecture: the absolute differentiation and the relative differentiation. The absolute model provides an absolute or quantitative performance level to each class by using some forms of admission control. In the following paragraph I focus on the relative differentiation model. The relative model provides some service level (or classes) that is relatively differentiated. In the relative differentiation model, higher class receives better performance than lower class in the expense of higher cost. The actual performance of a class could be bandwidth, delay, delay jitter or loss. These performance parameters of a class in the relative model cannot be known quantitatively in advance.

For the relative differentiation model, it is unnecessary to have admission controls, bandwidth brokers and resource reservations or signalling between users and networks. Route pinning or provisioning is also not required. Consequently, this model is simpler and easier to deploy and manage [Dov1].

The central premise of the relative differentiation is that *N* classes of service are ordered in the following sense:

> Class *i* provides better (or at least no worse) performance than class *j*, for *i*>*j*, in terms of per-hop queuing delays and packet losses.

There are eight relative differentiation classes, called Class Selector Compliant Per-Hop-Behaviors (CSC PHBs), or simply Class Selectors [Nich98], which are standardized by the IETF. Even though the use of the precedence bits has been limited in the past, there have been certain links or networks that deployed simple DiffServ schemes (such as providing higher priority to Telnet traffic) [Mills91].

For mapping packets to a certain Class Selector, several selections could be done: at application level, at operating system of end-hosts, or at edge routers. The case of mapping at the application level could be illustrated by an example that a WWW server classifies requested HTTP transactions, based on user-subscription

levels, so that non-members are mapped to the lowest class, while members get better services of a higher class [Bha99]. Or, in the case of mapping at operating systems, a policy-based classification of packets may be performed at local hosts of an academic organization, so that faculty uses the highest service class, graduate students a middle service class, while the undergraduate students the lowest class. For the last case, when the mapping could be done at end-hosts, a commercial network may classify ingress traffic at corresponding edge routers based on class prices and the maximum tariff that users are willing to pay. In order to map packets to different classes, different flow classification techniques [Beg99, Srin99] could be used.

The relative model could be used for providing absolute QoS requirements, when applications and users have some mechanisms that allow themselves to adapt dynamically to a class with an acceptable QoS level and within acceptable price range. For applications and users, which do not have requirements on absolute QoS guarantees, their classes can be fixed based on the performance versus cost trade-off.

Thus a Differentiated Services network based on the relative model will provide per-hop delay, bandwidth or delay jitter and loss differentiation. A higher class will receive better performance than lower classes. The applications, however, hope that they perform well in terms of end-to-end delay and loss rate. In this case, the receiver needs to monitor end-to-end performance and notifies the sender about current performances through a feedback channel. Based on this information, the sender will make a decision whether to stay in this class, or to move to a higher or lower class. If the user wishes to minimize the cost of sessions, the application will ask for the least expensive class that meets acceptable delays and losses.

3.2 Proportional Differentiation Model

In this section, I resume the Proportional Differentiation model as an alternative to the relative model. The meaning of the proportional differentiation here is explained as follows: the spacing between classes should follow proportional constraints on the class performance levels. Formally, the proportional differentiation model of the performance of class *i* is illustrated by the following equation:

$$\frac{\phi_i}{\phi_j} = \frac{\pi_i}{\pi_j}, 1 \le i, j \le N \qquad (3.1)$$

where π_i is the differentiation parameter of class *i*, and ϕ_i is the performance metric of the class, such as average delay, delay jitter or loss rate. Note that if lower values of ϕ_i lead to better performance, I must have that $\pi_i < \pi_j$ if $i>j$.

The class, which has lower differentiation parameters, will get better performance (delay, delay jitter or loss) than the classes that have higher differentiation parameters. These proportional differentiation parameters are used to adjust performances of classes so that they stay proportional with each other. One crucial task of the proportional differentiation model is that the service differentiation between classes must be independent class load distribution. That means network operator is responsible for providing certain spacing between classes, even though load conditions vary dynamically or are not known in advance.

The spacing between classes could be expressed as delay, delay jitter or packet losses. Because delay and loss are two major issues of the performance degradation factors in packet networks, the author in [Dov3] has applied the proportional differentiation model in terms of both queuing delays and packet losses.

Specifically, let d_i be the average queuing delay of packets served in class *i*. The proportional differentiation model in contexts of queuing delays requires that:

$$\frac{d_i}{d_j} = \frac{\delta_i}{\delta_j}, 1 \le i, j \le N \qquad (3.2)$$

Where δ_i is the *Delay Differentiation Parameter* (DDP) for class *i*. The DDPs are ordered as $\delta_1 > \delta_2 > ... > \delta_N > 0$.

Similar to the proportional delay differentiation model, the proportional differentiation model, in case of packet drops, requires that the class loss rates be spaced as:

$$\frac{\bar{l}_i}{\bar{l}_j} = \frac{\sigma_i}{\sigma_j}, 1 \le i, j \le N \qquad (3.3)$$

Where l_i is the loss rate of class *i*. The parameters σ_i are the *Loss Rate Differentiation Parameters* (LDPs).

3.3 Previous Works

In this section, I summarize the existing works on proportional delay service, and on proportional loss.

3.3.1 On Proportional Delay

In Chapter 1 I present the existing studies in DiffServ architecture. This section will provide a summary on scheduling algorithms that can produce proportional delay between different classes. Figure 6 shows the structure of a Proportional Delay Scheduler. Each delay class is served by a FIFO packet queue. All flows with the same class specification share the same FIFO queue at the router. At each time when the output link is free, the scheduler should decide which packet will be scheduled next if there are packets at the input queues.

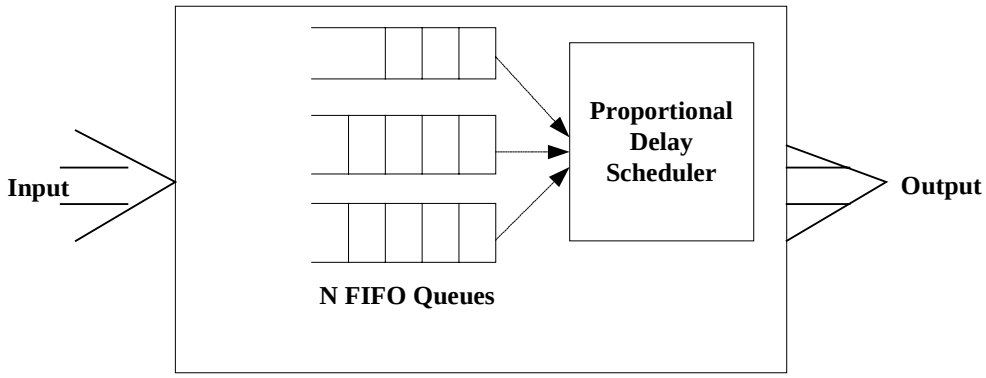


Figure 6 Proportional delay scheduler

The Asymmetric Best-effort described in [Bou99] is a model, which contains two service classes: one for real-time applications (such as IP telephony), and another for applications, which require some levels of throughput, such as data transfer.

In case of relative Differentiated Services model, the User Share Differentiation (USD) [Wang98] is a very good example, which guarantees that per-hop distribution of bandwidth is scheduled proportionally to profiles that user demands. A similar of relative service model has been studied for USD, too.

In addition, it is necessary to say about the Paris Metro Pricing (PMP) model [Odl99], which is a variant of relative Differentiated Services model. The fundamental idea of this model is to provide differentiation based on pricing,

instead of special router forwarding mechanisms. For implementing this model, the class, which pays more, receives higher loads, or better performance. However, the idea of differentiation based on pricing is only useful over relatively long timescales, in particular when class tariffs cannot be frequently modified. If higher classes become overloaded (because for example many rich users become active at the same time) they will offer worse performance than lower classes. This would be an instance of inconsistent or unpredictable relative differentiation, as I discussed.

Unlike the Paris Metro Pricing model, whose differentiation is based on bandwidth, Constatinos Dovrolis [Dov3] created another model, called Proportional Delay Differentiation Model, whose differentiation is based on queuing delay. In order to implement this type of service, there are two possibilities: WTP and BPR schedulers [Dov3], which are proposed by the same author. I discuss now the advantages and disadvantages of WTP and BPR schedulers and how they can support various service profiles.

WTP is a scheduling algorithm, which calculates priority of a packet proportionally with its waiting time. The priority of a Head of Line (HOL) packet in queue *i* at time *t* is defined as follows:

$$p_i(t) = w(t).\frac{1}{\delta_i} \tag{3.4}$$

Where $w(t)$ is the waiting time of Head of Line packet, and δ_i is the Delay Differentiation Parameter of class *i*. The most important advantage of WTP scheduler is the consistent when approximating the proportional delay differentiation model, in particular when load condition and traffic patterns vary. However, WTP decouples delay from service rate, and that is why it is not so easy to realize different service performance types, as throughput differentiation and delay differentiation… at the same time without using multi-level scheduling architecture.

Usually, if a connection receives less delay than other connections, it will imply that more bandwidth is allotted to these connections, but this property is not always true. This problem is illustrated by the experiments realized in [Nguy01]. Suppose that I have four users, which use TCP service. These four users send data and the traffic produced by these four users passes through a node, which uses WTP scheduler. For each user, I assign a class of service, from class 1 to class 4, respectively. Accordingly, the four Delay Differentiation Parameters of these classes are $\delta_1 = 1$, $\delta_2 = 1/2$, $\delta_3 = 1/3$, $\delta_4 = 1/4$. The user, which uses the highest priority class, hopes that he will receive more bandwidth. But results shown in two tested scenarios lead to a conclusion that throughput performance is quite

different between the both scenarios: while the bandwidth performance in scenario 1 is what I expected, the fourth user in scenario 2 got actually less bandwidth than other users. That means WTP could provide delay differentiation, but this scheduling algorithm is not stable for providing bandwidth differentiation. This is a reason that leads to a new scheduling algorithm, called Differentiated Delay and Throughput Scheduler (DDTS) in [Nguy01]. This is a scheduling algorithm, which provides simultaneously delay and throughput differentiation and link sharing. The scheduling policies are integrated in a single service discipline. Under certain cases, the delay service discipline in DDTS provides better performance than the WTP scheduler.

Unlike the WTP, BPR is originally derivate from the Generalized Processor Sharing- GPS [Par93] scheduling. This GPS system schedules the packet based on the rate allotted to the packet's session and the queue backlog of that session at the time packet arrives. The scheduling mechanism of BPR is based on the reallocation of rates to its classes of service proportionally to their backlog. Suppose that $r_i(t)$ be the service rate assigned to queue at time *t*, $q_i(t)$ be the queue *i* backlog at time *t*. For two backlogged queues *i* and *j*, the service rate allocation in BPR satisfies the proportional constraints:

$$\frac{r_i(t)}{r_j(t)} = \frac{s_i}{s_j}\frac{q_i(t)}{q_j(t)} \tag{3.5}$$

Where s_i are the Scheduler Differentiation Parameters, that are related directely to the Delay Differentiation Parameters. Because BPR algorithm is done by the rate-allocation, one can think about the possibility of integration of link sharing policies and throughput differentiation policies into BPR. However, the performance of BPR in terms of proportional delay differentiation is noteworthy worse than WTP, especially when the load distribution between classes of service is not symmetric.

A novel algorithm for buffer management and packet scheduling is proposed in [Lieb01], called JoBS (Joint Buffer Management and Scheduling). This new proposition could provide loss and delay differentiation simultaneously for traffic classes at a network router. The model of network using JoBS requires no admission control and policing. The novel property of the JoBS algorithm is that scheduling and buffer management decisions are done in a single step. Furthermore, both relative and absolute QoS requirements of classes are supported.

In order to evaluate and compare the performance of our new model PJDM with the existing model PDDM, I decide to choose WTP as the scheduling mechanism

implemented in PDDM since it is the simple algorithm that can maintain proportional delay ratio stably [Lieb01].

3.3.2 On Proportional Loss

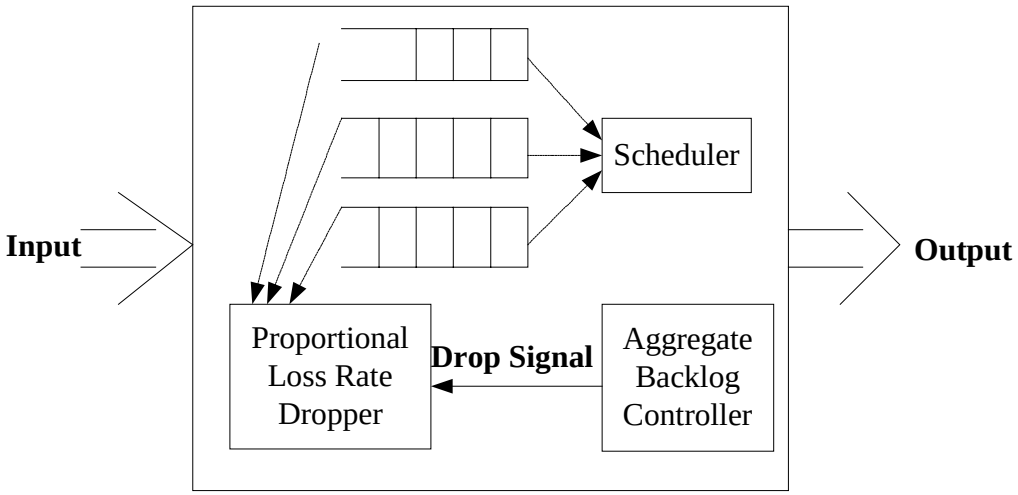


Figure 7 Proportional loss rate dropper

Figure 7 shows the architecture of a proportional loss rate dropper, similar to proportional delay module. Suppose that there are N classes of service, the model of proportional loss differentiation is described as follows. For each FIFO queue, a logical queue is created. When a packet arrives, it will be queued to the appropriated queue based on class marking. The scheduler module is responsible for determining which packet will be scheduled when outgoing links are available for achieving the necessary delay or delay jitter differentiation between classes. Another important module, which is called backlog controller, controls backlog of these logical queues, and take a decisions whether a packet should be dropped or not, and which packet should be dropped.

The most straightforward example of this backlog controller is Drop-Tail algorithm. This simple mechanism drops packets when the buffer space is higher than a threshold, or when there is no more available space. There exist also other complex buffer management schemes, such as RED or its variants.

When it is necessary to discard a packet, a backlogged queue will be selected by the packet dropper module, a packet from this backlogged queue will be dropped. That means the backlog controller will monitor aggregate backlog in forwarding engine, and the dropper module is responsible for controlling the loss rate differentiation between classes. This dropper module could remove a packet from the tail, from the head of line position…of queue.

3.3.2.1 Proportional Loss Rate (PLR) Droppers [Dov2]

For achieving the model of proportional differentiation in terms of loss, the normalized loss rate l_i / σ_i should be equal for all classes. PLR dropper maintains a running estimated l_i of loss rate in each class. When there is a need to discard a packet, PLR dropper will select a backlogged class with the minimum l_i / σ_i ratio. In other words, PLR dropper will throw up a packet from a backlogged class j with $j = \arg\min_i \frac{l_i}{\sigma_i}$. This operation is explained as discarding a packet from class j will reduces the difference of l_i / σ_i from the normalized loss rates of the other classes, and tending to equalize them.

3.3.2.2 Proportional Loss Rate Dropper with infinite memory PLR(∞) [Dov2]

This dropper is a version of the PLR dropper, it uses the loss rate estimate l_i, which is the long-term fraction of packets from class i that have been removed. This loss rate counter can be calculated using a single parameter for the arrivals and drops in each class. This element is called a dropper with infinite memory, because the loss rate is calculated from the entire history of previous arrivals and drops. The complete algorithm is shown as follows:

B(t): Set of backlogged classes at time *t*
A_i : Counter of packet arrivals in class *i*
D_i : Counter of packet drops from class *i*

1. Packet arrival in class *i*:
$A_i ++$

2. Packet drop from class *j* at time *t:*
$j = \arg\min_{i \in B(t)} \frac{D_i}{A_i \sigma_i}$
$D_j ++$

Figure 8 Description of the PLR(∞) dropper

The length of two parameters A_i and D_i of the PLR(∞) dropper is an important point, especially when counters overflows. One solution for this overflow

problem is to reset all counters, and enter a cold start phase, when any of arrival counters overflows. For example, if counters are 32-bit, the counter A_i will overflow after at least four billion packet arrivals. It is anyway advantageous when the counter is reset over shorter intervals, so that the dropper is more adaptive to changing class distributions.

For implementing the algorithm in reality, it is necessary to realise *N* multiplications and *N* divisions every time a packet needs to be dropped. This operation could be generally in floating-point arithmetic. An optimisation that avoids the multiplications would be to increase A_i by σ_i every time a packet arrives in class *i*. If a single-precision division take 40 cycles, the calculation of the normalized loss rates for 8 classes would take 320 cycles, and with a 700MHz CPU the selection of the drop-target class would require about 0.5micros.

3.3.2.3 PLR(M): Proportional Loss Rate Dropper with memory M [Dov2]

The estimation of class loss rate based on long history of packet arrivals is the most important disadvantage of the PLR(∞) dropper, which makes it less adaptive to changing class load distributions.

In contrast to PLR(∞) dropper, this algorithm defines the loss rate of class *i* as the fraction of dropped packets from class *i* in the last M arrivals. For doing this, a table, called Loss History Table LHT, records the class index $(LHT[i].class \in \{1,...,N\})$ and the drop status $(LHT[i].drop \in \{0,1\})$ of each packet in the last *M* arrivals.

Using counters in this table LHT, the dropper maintains a knowledge about number of arrivals $A_i(M)$ and number of drops $D_i(M)$ from class i in the last M arrived packets.

This scheme is different from the PLR(∞) in the way of counting packets: it resets counters after every *M* arrivals and tries to achieve the proportional loss rate differentiation model in every time window of *M* arrivals while PLR(∞) tries to achieve proportional loss rate differentiations in successive time windows.

In order to implement PLR(M), it is necessary to have internally in routers a packet tag, which are rewritten when packets are dropped. This element is the main implementation complexity of PLR(M) compared to PLR(∞). Furthermore, it has to be noted that such temporary packets tags, attached to packets in routers while packets are being forwarded, are not uncommon in switch and router designs.

Another important issue concerning to designs problem is the length of window *M*. This parameter should be large enough so that a dropped packet is always one of the last *M* arrived packets; otherwise, it will cause deviations of the LHT-based loss rate from its actual value. This constraint is met in practice, even for small values of *M*, when the dropped packets are removed from the tail of the queues. A more tight constraint on *M* is that it should be large enough so that the specified LDPs are feasible, with a given current load distribution and aggregate loss rate. Intuitively, if *M* is not large enough, the PLR(M) dropper cannot remember enough the previous arrivals and drops in order to adjust the loss rates based on proportional constraints.

3.3.2.4 Average Delay Distance [Bod01]

Unlike the previous algorithms, ADD is a mechanism, which uses an estimator on average drop distance (ADD) for controlling loss-rate. For each drop precedence level, an ADD covers a history, whose length is defined in number of drops. This makes the ADD algorithm more adaptive to history lengths at each level to changing load distributions. The history covered by this estimator can be set short without risking estimated loss-rates to be zero for some traffic loads. By using this type of estimators, the history length simply determines how fast the changing traffic load conditions are detected. Hence, the ADD estimators do not have the same trade-off in choosing history length as the LHT estimator.

3.4 Proportional Delay Differentiation Model

The Proportional Delay Differentiation (PDDM) model intends to space the average queuing delays so that the ratios of these average queuing delay stay proportional with each other, based on Delay Differentiation Parameters (DDPs) $\{\delta_i, i = 1,..., N\}$.

In detail, suppose that $\bar{d}_i$ is the average queuing delay, or simply average delay of the class *i* packets. The PDDM model, which requires that the ratio of average delays between two classes i and j fixed to the ratio of the corresponding DDSs, is formally illustrated by the following question:

$$\frac{\bar{d}_i}{\bar{d}_j} = \frac{\delta_i}{\delta_j}, 1 \le i, j \le N \tag{3.6}$$

The model of PDDM has two objectives. First, this model should produce consistent service differentiation between classes. That means a class with higher advertised quality should consistently outperform a class with lower advertised quality. In addition, it should allow the possibility of adjusting quality spacing between classes, based on pricing and other criteria. Furthermore, these two goals should be met even for sharing over short timescales.

The scheduling algorithms as WTP or BPR, proposed in [Dov3], are created specially for this PDDM model. The first approach, WTP, is based on Kleinrock's Time-Dependent-Priorities algorithm [Leung00], and is shown to be highly effective in [Dov3]. Because WTP is derived from TDP algorithm in [Leung00], I summarize some related works of TDP on feasibility conditions for achieving proportional delay.

3.4.1 Time Dependent Priority Scheduler

This section will outline some results for TDP scheduling. The necessary and sufficient conditions for a given delay spacing to be feasible under TDP for two traffic classes are analysed here. These characterizations are later extended for *N* classes.

TDP is a packet scheduling algorithm that uses a set of control variables b_i, $1 \le i \le N$ where $0 \le b_1 \le b_2 \le ... \le b_N$ (in the PDDM model, these variables are called $\left\{\frac{1}{\delta_i}, i = 1,..., N\right\}$. These parameters are the dynamic priorities of class *i* packets. Formally, if the *k-th* packet of the class *i* arrives at its queue at time τ_k, then its priority at time *t* (for $t \ge \tau_k$), denoted by $q_i^k(t)$, is:

$$q_i^k(t) = (t - \tau_k)b_i$$

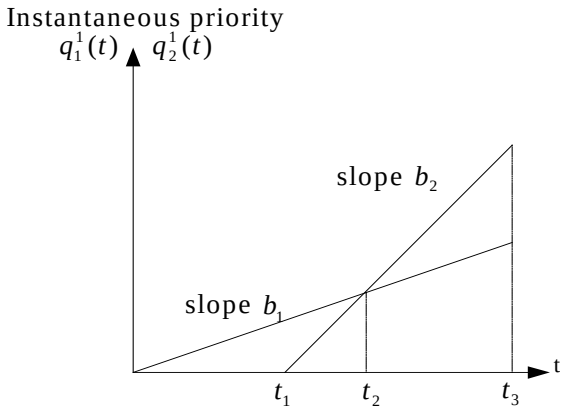


Figure 9 Two class TDP where *b1 < b2*

In Figure 9, I have a TDP scheduler with only two classes. The first packet of class 1 arrives in system at time 0, and the class 2 has the first packet with arriving time t_1. These two packets will reside in system until time t_3. From the time 0 to t_1, there is only one packet of class 1 in systems. Within the time interval $(t_1, t_2]$, the first packet of class 1 has a larger priority than the first packet of class 2. But after time $t > t_2$, the control parameter b_2 becomes higher than b_1, that means higher priority.

Suppose that $N_i(t)$ is the number of packets of class i in the queue at time *t.* The conception of TDP scheduler is to select the HOL packet with highest priority, when the outgoing link is able to transmit a packet. It is necessary to note that for a same class the HOL packet is chosen, because the earlier arrival packets always have higher priorities than the later arrival packets.

Formally, the TDP scheduler will choose a packet from class *i** satisfied the following conditions, when server is ready to transmit a packet:

$$i^*(t) = \arg_{i=1\ldots N, N_i(t)>0} \max\{q_i(t)\}$$

where $q_i(t)$ is defined as priority of the packet at the head of the class *i* queue.

By serving the packet that has been waiting the longest in system, the average delay of this class is reduced. After this the server stays idle and it will react when there is new arriving packets. It is very important to note that in the TDP scheduler, a class packet increases in priority at a faster rate (b_i) than packets of any class $j<i$.

The long-term waiting time of each class of traffic under the TDP scheduling algorithm could be also analysed. Suppose that each class *i* traffic has Poisson arrival process with an average rate of λ_i. In addition, the services time of class *i* packets follows general distribution with the first and second moments, designed respectively as $\bar{x}_i$ and $\bar{x}_i^2$. The TDP system utilization, named as ρ, will be the sum of ρ_i ($\sum_{i=1}^{N}\rho_i$), where $\rho_i = \lambda_i \bar{x}_i$. A closed form expression for the average long-term waiting time for class *i* packets is derived in [Leung00] as the following equation:

$$\bar{d}_i = \frac{[\bar{d}_0/(1-\rho)] - \sum_{k=1}^{i-1}\rho_k \bar{d}_k[1-(b_k/b_i)]}{1-\sum_{k=i+1}^{N}\rho_k[1-(b_i/b_k)]} \quad i = 1,\ldots,N \qquad (3.7)$$

Where $\bar{d}_0 = \frac{1}{2}\sum_{i=1}^{N}\lambda_i \bar{x}_i^2$ is the expected residual service time. The above equation was derived by assuming that packet service times are exponentially distributed and is also valid for any general service time distribution (see [Nett79]).

An important advantage of the TDP scheduler is that it is able to maintain certain proportional differentiations of waiting times between different traffic classes. For doing this, it is only necessary to control the control parameters b_i so that the average delay ratio achieves the desired values. Let $r_{1,2}^t$ be the target long-term average waiting time ratio between class *i* and class *j* traffic, and $r_{1,2}^a$ be the achieved long-term average waiting time ratio between class *i* and class *j* traffic (i.e., $r_{i,j}^a = \frac{\bar{d}_i}{\bar{d}_j}$). The goal of proportional delay Differentiated Services is to make the achieved long-term waiting time ratio equal to the target long-term waiting time ratio, that means $r_{i,j}^a = r_{i,j}^t$.

3.4.1.1 2 Classes-Case

Theorem 1: Suppose that $r_{1,2}^t$ is the target ratio of the average waiting time of class 1 traffic to that of class 2 traffic. The equation $r_{1,2}^a = r_{1,2}^t$ is feasible if and only if the system utilization ρ satisfies:

$$1-\frac{1}{r_{1,2}^{t}}<\rho<1$$

Proof: At first, if $\rho<1$, then the system does not stay stable. That means $\rho<1$ is the required condition so that the stable situation of system is maintained.

Now, the *only if part* will be demonstrated. From the equation (3.7), the average waiting times of these two classes are received as:

$$\bar{d}_1=\frac{[\bar{d}_0/(1-\rho)]}{1-\rho_2[1-(b_1/b_2)}$$

$$\bar{d}_2=[\bar{d}_0/(1-\rho)]-\rho_1\bar{d}_1[1-(b_1/b_2)]$$

If submit $\bar{d}_1$ into the equation of $\bar{d}_1$ into $\bar{d}_2$, $\bar{d}_2$ could be written as the following equation:

$$\bar{d}_2=[\bar{d}_0/(1-\rho)]\frac{1-\rho[1-(b_1/b_2)]}{1-\rho_2[1-(b_1/b_2)]}$$

The achieved long-term average waiting time ratio between class 1 and class 2 is described as:

$$r_{1,2}^{a}=\frac{\bar{d}_1}{\bar{d}_2}=\frac{1}{1-\rho[1-(b_1/b_2)]}$$

If the target ratio is achieved, that is, $r_{1,2}^{a}=r_{1,2}^{t}$, then:

$$r_{1,2}^{t}=\frac{1}{1-\rho[1-(b_1/b_2)]}$$

After rearranging the above equation, the following relation is gained:

$$\rho=\frac{b_2}{b_2-b_1}(1-\frac{1}{r_{1,2}^{t}})$$

Since $0<b_1<b_2$, this implies that $\frac{b_2}{b_2-b_1}>1$. Therefore $\rho>1-\frac{1}{r_{1,2}^{t}}$.

Next, the *if part* will be proved (i.e., if $1-\frac{1}{r_{1,2}^t} < \rho < 1$), then $r_{1,2}^a = r_{1,2}^t$. If $\rho > 1-\frac{1}{r_{1,2}^t}$, then ρ could be described as follows:

$$\rho = \frac{b_2}{b_2 - b_1}(1-\frac{1}{r_{1,2}^t})$$

Here, the parameters b_1 and b_2 are some constants such that $0 < b_1 < b_2$. Buy submitting it into equation (3.7), the equation $r_{1,2}^a = r_{1,2}^t$ is obtained.

Remark: This theorem shows that for achieving the target ratio, it is necessary to have sufficient amount of traffic and enough packets, which are backlogged in the system. For example, if the requirement is 10, then the system has to be at least 90% utilized so as to achieve the desired waiting time spacing. In other words, if the system utilization is less than 90%, then the target ratio 10 could not be achieved.

Corollary 1: If $b_1 = 1$, $b_2 = \rho/(\rho - 1 + \frac{1}{r_{1,2}^t})$, and $1-\frac{1}{r_{1,2}^t} < \rho < 1$, then $r_{1,2}^a = r_{1,2}^t$.

Proof: By replacing $b_1 = 1$, $b_2 = \rho/(\rho - 1 + \frac{1}{r_{1,2}^t})$ into equation (3.7), the relation $r_{1,2}^a = r_{1,2}^t$ could be achieved.

Corollary 2: If $1-\frac{1}{r_{1,2}^t} < \rho < 1$, then any b_1 and b_2 such that $b_1/b_2 = (\rho - 1 + \frac{1}{r_{1,2}^t})/\rho$ can achieve $r_{1,2}^a = r_{1,2}^t$.

Proof: First, the author in TDP scheduling algorithm states that for two TDP systems ***A*** and ***B*** wherein the control parameters for system ***A*** are b_i and the control parameter for system ***B*** are b_i'. If the following liaison is kept:

$$\frac{b_i}{b_{i+1}} = \frac{b_i'}{b_{i+1}'} \text{ for } i=1,2,\ldots,N$$

Then W_i in ***A*** will be equal to W_1' in ***B***. In other words, the average waiting time of TDP system depends *not on the exact value* of the control parameters b_i but rather depends on *the ratios* of b_i .

If $1-\frac{1}{r_{1,2}^t} < \rho < 1$, from the corollary 1, $b_1 = 1$ and $b_2 = \rho/(\rho - 1 + \frac{1}{r_{1,2}^t})$ can achieve $r_{1,2}^a = r_{1,2}^t$. As TDP system depends on the ratios of b_i's only, any b_1 and b_2 such that:

$$\frac{b_1'}{b_2'} = \frac{1}{\rho/(\rho - 1 + \frac{1}{r_{1,2}^t})} = \frac{\rho - 1 + \frac{1}{r_{1,2}^t}}{\rho}$$

can achieve $r_{1,2}^a = r_{1,2}^t$.

3.4.1.2 *N* Classes-Case

Generally, when the TDP system server has more than 2 classes, for example *N* classes, the problem becomes very complicated to solve, because to answer the question how the values of the control parameters b_i should be for satisfying the equation (3.3), it is necessary to solve a system of *N* non-linear equations. On the other side, when the configuration of the system (ρ_i and $\bar{d}_0$) is known before, it is possible to determine $\bar{d}_i$ by using the conservation law principle.

Recall that according to the *conservation law* [Klein76], if a scheduling discipline is independent of the service time of jobs, then the weighted average of waiting times of all classes are invariant, and it is equal to the average waiting time of an *M/G/1* system. Formally, the conservation law is described by the following equation:

$$\sum_{i=1}^{N} \frac{\rho_i}{\rho} \bar{d}_i = \frac{\bar{d}_0}{1-\rho} \tag{3.8}$$

If s_i is defined as $s_i = r^t_{i,i+1} r^t_{i+1,i+2} ... r^t_{N-1,N}$, and if the equation $r^a_{1,2} = r^t_{1,2}$ could be achieved, then $s_i = \frac{\bar{d}_i}{\bar{d}_N}$. That means it is possible to express all $\bar{d}_i$'s by the relationship with $\bar{d}_N$ and s_i: $\bar{d}_i = s_i \bar{d}_N$ for all *i=1,2,…N*.

By replacing the parameters $\bar{d}_i$'s from equations (3.8) in the conservation law, the following equation is obtained:

$$\frac{\bar{d}_0}{1-\rho} = \frac{1}{\rho} \sum_{i=1}^{N} \rho_i s_i \bar{d}_N$$

$\bar{d}_i$ can be expressed in terms of s_i, ρ_i and $\bar{d}_0$ as:

$$\bar{d}_i = s_i \frac{\rho \bar{d}_0}{1-\rho} \left(\sum_{i=1}^{N} \rho_i s_i\right)^{-1} \text{ for } i=1,2,\ldots,N$$

From the above equation, if $r^a_{1,2} = r^t_{1,2}$ is achieved, the only unknown in equation (3.7) is the vector $b = [b_1, b_2, ..., b_N]$. Now, setting all b_i's in equation (3.7) on the left hand side, the following equation is obtained:

$$b_i \left(\bar{d}_i \sum_{k=i+1}^{N} \frac{\rho_k}{b_k}\right) - \frac{1}{b_i} \left(\sum_{k=1}^{i-1} \rho_k \bar{d}_k b_k\right) = \frac{\bar{d}_0}{1-\rho} - \sum_{k=1}^{i-1} \rho_k \bar{d}_k - \left(1 - \sum_{k=i+1}^{N} \rho_k\right) \bar{d}_i$$

Suppose that:

$$A(i) = \bar{d}_i \sum_{k=i+1}^{N} \frac{\rho_k}{b_k}; \; B(i) = \sum_{k=i}^{i-1} \rho_k \bar{d}_k b_k \; ;$$

$$R(i) = \frac{\bar{d}_0}{1-\rho} - \sum_{k=1}^{i-1} \rho_k \bar{d}_k - \left(1 - \sum_{k=i+1}^{N} \rho_k\right) \bar{d}_i \; ,$$

Then:

$$A(i) b_i - \frac{B(i)}{b_i} = R(i) \quad i=1,2,\ldots,N$$

This is a system of non-linear equations for solving the b_i. Because all the values of b_i should be positive, the parameters of ρ_i and s_i must satisfy a condition.

Theorem: For receiving the positive values of all the b_i, the necessary condition is that *R(1)>0* and *R(N)<0*.

Proof: For receiving the positive values of all the b_i, the necessary condition is that *R(1)>0* and *R(N)<0*.

Case 1: For *i=1*, I have *B(1)=0*, which implies that:

$$b_i = \frac{R(1)}{A(1)}$$

Since $b_1 > 0$, this in turn implies that *R(1)>0*

Case 2: For *1<i<N*, the result from equation (3.20) is used and:

$$A(i)b_i^2 - R(i)b_i - B(i) = 0$$

Because the parameters $\{b_i\}$'s should be positive, its will receive the following values:

$$b_i = \frac{R(i) + \sqrt{R(i)^2 + 4A(i)B(i)}}{2A(i)}$$

Because $R(i)^2 + 4A(i)B(i) > R(i)^2$, therefore $|R(i)| < \sqrt{R(i)^2 + 4A(i)B(i)}$. Hence:

$$b_i > \frac{R(i) + |R(i)|}{2A(i)} \geq 0$$

In summary, for *1<i<N*, b_i is always greater than zero even when *R(i)* is negative.

Case 3. For *i=N*, I have *A(N)=0*, which implies that:

$$b_N = -\frac{B(N)}{R(N)}$$

Since $b_N > 0$ and *B(N)>0*, I conclude *R(N)<0*.

Remarks: The implication of the above theorem is: a necessary condition for a feasible region (e.g., a region wherein a positive solution of b_i's exist) is *R(1) >0* and *R(N)<0*. If the system configuration (ρ_i and s_i) falls outside of this region, it is possible that there exist no positive values of the b_i's for which the TDP scheduler can obtain the target waiting time ratios.

The first condition *R(1)>0* implies:

$$\frac{\bar{d}_0/(1-\rho)}{\bar{d}_1} > 1 - \sum_{i=2}^{N} \rho_i \tag{3.9}$$

where $\bar{d}_0/(1-\rho)$ is the average waiting time of the aggregate traffic. If I want a large waiting time differentiation, $\bar{d}_1$ has to be larger than $\bar{d}_i$, *i=1,2…N*. Since $\bar{d}_1 \geq \bar{d}_2 \geq ... \geq \bar{d}_N$, this implies the fraction on the left hand side of equation (3.9) to be small. Thus, $\sum_{i=2}^{N} \rho_i$ should be close to one to make the inequality hold. The physical meaning is that to have a large waiting time differentiation, there should be a sufficient amount of higher class packets to keep the system busy so that the lower class packets are delayed adequately.

The second condition is $R(N) < 0$, which implies:

$$\frac{\bar{d}_0}{1-\rho} < \sum_{i=1}^{N-1} \rho_i \bar{d}_i + \bar{d}_N \tag{3.10}$$

By the conservation law, $\bar{d}_0/(1-\rho) = \sum_{i=1}^{N} \frac{\rho_i}{\rho} \bar{d}_i$. If I put it back into equation (3.10), I have:

$$\sum_{i=1}^{N} \frac{\rho_i}{\rho} \bar{d}_i < \sum_{i=1}^{N-1} \rho_i \bar{d}_i + \bar{d}_N$$

$$0 < \sum_{i=1}^{N-1} \rho_i (\bar{d}_N - \bar{d}_i (1-\rho)) \tag{3.11}$$

Since $\bar{d}_1 \geq \bar{d}_2 \geq ... \geq \bar{d}_N$, to make the right hand side of equation (3.11) positive, one way is for ρ to be large. If ρ tends to *1*, the value of the left hand side in equation (3.10) will be large. To make the inequalities hold, the value of the right

hand side in equation (3.9) should be larger. Since $\bar{d}_1 \geq \bar{d}_2 \geq ... \geq \bar{d}_N$ and the major part $\sum_{i=1}^{N-1} \rho_i \bar{d}_i + \bar{d}_N$ is the weighted average of the mean waiting time of the first *N-1* classes, to attain a large value, ρ_i should be large, especially for the lower traffic classes. The physical meaning is that in order to have a large waiting time differentiation, the server has to delay packets of the lower classes so as to have large waiting times $\bar{d}_i$, *i=1,2…N-1*. If their traffic loading is high, many of them will be backlogged and their waiting time will increase. Last but not least, another important implication of the above necessary conditions is that even though the system utilization ρ remains unchanged, it is still possible that certain distributions of ρ_i's will not lead to a positive solution of b_i's. In such case, the system cannot achieve the target waiting time ratios.

3.4.2 Per-class Average Delays in the PDDM Model

In given load conditions, the *N-1* ratios of the PDDM model specify uniquely the average delays of the *N* classes. The key additional relation in mappings from delay ratios to class delays is the conservation law, which constrains the average class delays in any work-conserving scheduler ***S***. The conservation law holds under arbitrary distributions for the packet inter arrivals and packet sizes, as long as the first moment of these distributions and the second moment of the packet size distribution exist, and the packet scheduling discipline ***S*** is independent of the packet sizes.

3.4.3 Delay Dynamics in the PDDM model

Property 1: *Increasing the input rate of a class, increase (in the wide sense) the average delay of all classes.*

In the other words, there is always a link or relationship between classes that is expected due to the relative differentiation nature of the model. When the input rate of a class increases, the load of this class is added, and the delays of all classes will also encounter an increase.

Property 2: *Increasing the rate of a higher class causes a larger increase in the average class delays than increasing the rate of a lower class.*

In the simplest case, when there are only 2 classes. Assume that the following two cases occur:

- $\lambda_1' = \lambda_1 + \varepsilon$ and $\lambda_2' = \lambda_2$
- $\lambda_1'' = \lambda_1$ and $\lambda_2'' = \lambda_2 + \varepsilon$

with a condition that ε>0. According to the conservation law, the weighted average of the class delays is the same in both cases:

$$\lambda_1' W_1' + \lambda_2' W_2' = \lambda_1'' W_1'' + \lambda_2'' W_2''$$

From the above equations, it is clearly to see that because of the PDDM constraints, the class average delays in the second case are larger, i. e. $W_1' > W_1''$ and $W_2'' > W_2'$.

This property shows that higher classes cost more, in terms of queuing delay, than lower classes.

Property 3: *Decreasing the delay differentiation parameters of a class increases (in the wide sense) the average delay of all other classes, and decreases (in the wide sense) the average delay of that class.*

That means if I diminish the delay of a class by minimizing its Delay Differentiation Parameters, it will imply that the delay of all other classes will increase.

Now, assume that the class load distribution changes from $\{\lambda_n\}$ to $\{\lambda_n'\}$, with $\lambda_i' = \lambda_i - \varepsilon$; $\lambda_j' = \lambda_j + \varepsilon$, and $\lambda_k' = \lambda_k$ for all $k \neq i, j$ ($\varepsilon > 0$). W_n' is the average delay in class *n* when the class load distribution is $\{\lambda_n'\}$. The following properties are important in case of Dynamic Class Selection (when a class chooses a higher or lower class for achieving end-to-end guarantees):

Property 4: *If i>j then* $W_n^{'} \leq W_n$ *for all n=1…N. Similarly, if i<j then* $W_n^{'} \geq W_n$.

Property 5: *If i>j then* $W_j^{'} \leq W_i$. *Similarly if i<j then* $W_j^{'} \leq W_i$.

Property 6: *Delay has accumulated property.*

It is easy to see that delay has accumulated property. That means the delay of a class or flow through a network is the sum of the queuing delays at each router and the propagation time, which is considered small compared to the queuing delays. Hence there is a need to implement proportional delay scheduling algorithms at every router in a network based on PDDM model for receiving proportional delay between different classes. However, in the case of the

existence of proportional delay scheduling schemes at every router, each packet transfers through network along different paths that contain different hops numbers. That is why the proportional property of queuing delay is just maintained for only one local hop, but the sum of queuing delay of one class does not stay proportional with other classes any more. In other words, such networks can not guarantee the proportional property of delay between different classes. This is a big disadvantage of the PDDM model.

Chapter 4 Playout Buffer Delay Adjustment Algorithm

Chapter 4 provides background information about the importance of the playout-buffer adjustment algorithm. It gives an overview about the existing playout schemes and also emphasizes the trade-offs between playout buffer delay and loss rate. After analysing the existing algorithms, I have decided to use Concord algorithm at the receiver end in my work. Properties and characterizations of the Concord algorithm are also analysed.

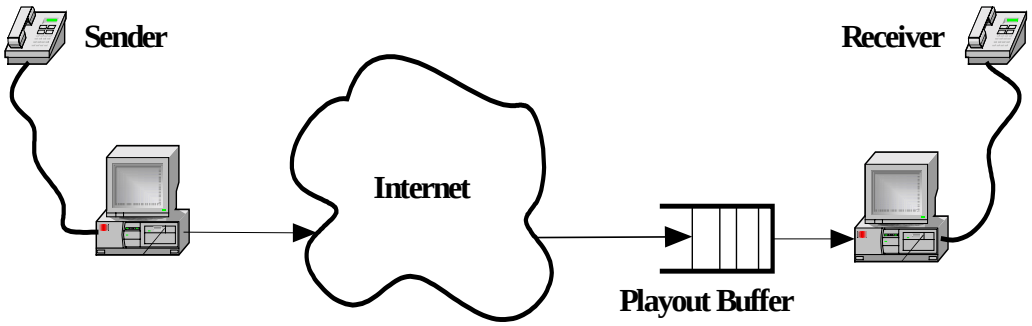


Figure 10 Packet voice with receiver jitter compensation

Usually it is required to have a playout buffer at the receiver end for audio or video signal in order to smooth the jitter produced by different networks because this jitter can degrade the quality of audio and video stream heavily.

Figure 11 illustrates how the playout buffer stores the arriving packets and calculates appropriate playout-delay time so that the total end-to-end delay of packets is smoothed:

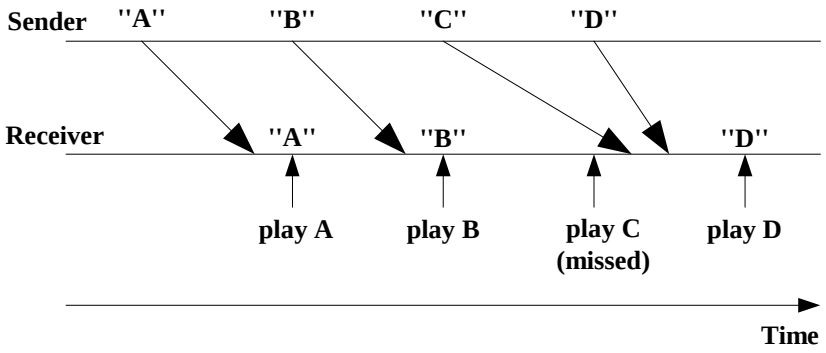


Figure 11 Different playout buffer time

As indicated in the above figures, different packets arrive at playout buffer each at a different time. Depending on the algorithm used at this receiver, the playout buffer delays these packets by different playout delays so that the jitter of the stream is reduced.

4.1 End-to-End Delay Characteristics

A lot of previous studies [Bol93] demonstrated the presence of “spikes” within the problem of end-to-end Internet delays. A spike is a sudden, large increase in end-to-end network delays, followed by a series of packets arriving almost simultaneously, leading to the completion of the spike.

With periodically generated packets, the initial steep rise in the delay spike and the linear, monotonic decrease after the initial rise, is due to ‘’probe compression’’- the accumulation of a number of packets from the connection under consideration (the audio session, in my case) in a router queue behind a large number of packets from other sources. Probe compression is a plausible conjecture about the cause of delay spikes.

4.2 Classification

As noted above, it is necessary to have a playout buffer at receiver side, which stores temporarily incoming Media Units (MUs), and a playout scheduler, whose the role is to provide a presentation schedule that resembles as much as possible the temporal relationship that was created by the encoding process.

Some applications, such as desktop videoconferencing require very strict latency, for example a few hundreds of milliseconds. Other unidirectional applications, such as video on demand (VOD) are more tolerant with larger latencies that range from around 1second for responsible Web-based distribution of short video clips to several minutes in near-VOD systems. That means there is some compromise between the intra-stream synchronization quality and the increase of end-to-end delay due to the buffering of MUs. If the receiver is has no buffer, the scheduler provides minimal stream delay by presenting frames as soon as they arrive. Another example of a playout buffer, which eliminates completely the effects of jitter at the expense of a long stream delay, is the *assured synchronization method*.

Various playout schedulers differ from each other in the usage of *timing information*. *Time-oriented playout schemes* put timestamps on MUs and use clocks at the sender and receiver in order to measure the network delay or differential network delay (jitter). *Buffer-oriented schemes* do not use timing

information. Instead, they implicitly assess the current level of network jitter by observing the occupancy of the playout buffer. In the both of cases, the level of synchronization between the sender and receiver influence strongly the design and capabilities of the system.

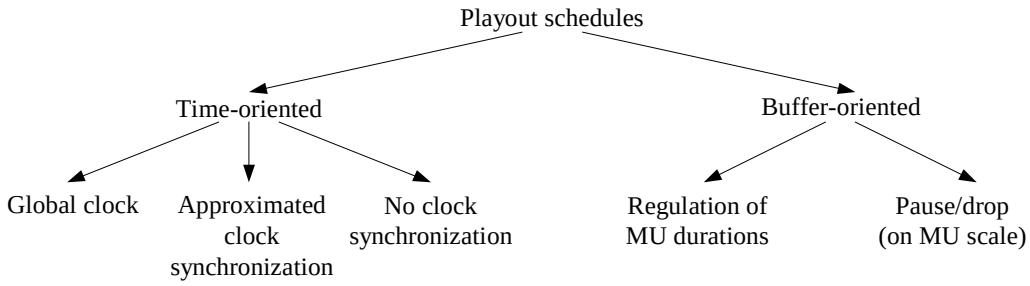


Figure 12 A general classification of playout schedulers

Figure 12 shows different types of playout schedulers. Systems that use their clock for the synchronization protocol are called having a *global clock*. The global clock measures exactly the network transfer delay of an MU and along with the buffering delay at the receiver end, the total end-to-end delay of the MU is smoothed. With such knowledge, the receiver can then guarantee that an MU is delivered before an available (requested) end-to-end delay budget is exhausted. Other methods, such as *differential delay methods* cannot measure precisely the network delay of an MU because of not using a global clock. They operate namely on delay variations (capture by the difference between subsequent delay measurements) and try to maintain a fixed trade-off between the perceived delay and the synchronization quality of the stream across time-varying jitter.

The playout algorithm, which uses *approximated clock synchronization*, tries to bind the offset between the two clocks by using the virtual clock algorithm. Under the virtual clock, the clock of the receiver adopts as local time the timestamp form a reference packet sent by the sender.

The previous schemes as global clock and approximated clock detect increases in end-to-end latency, and belong to time-oriented type. Time-oriented systems can realise if an MU is ‘’late’’ by comparing its arrival timestamp with its scheduled playout time. Unlike time-oriented system, buffer-oriented system will determine a latency increase by detecting an overbuilt queue of MUs waiting to be displayed. In addition, it is able to treat late MUs through two ways: *delay-preserving* and *non-delay-preserving* schemes. The first category, called delay-preserving schemes, in which all late frames are discarded to preserve the delay requirements of the stream; and non-delay-preserving schemes, in which some or

all late packets are accepted for presentation in order to protect the continuity of the stream from further degradation due to discarding MUs that are late.

In general, playout buffer schemes control the occupancy of the playout buffer with the variation of network delay jitter. If the delay variability increases, the playout buffer is also increased in order to smooth this variation. Contrarily, in time of reduced jitter, this playout buffer is decreased as well.

Packet-audio systems with silence detection technique modify the playout buffer by taking advantage of silence periods, which they use to make adjustments on a per-talkspurt basis. There are two types of buffer-oriented systems: systems with *pause/drop method* usually drop a frame to reduce latency, or stop the presentation of frames for one frame period (a pause), and systems that change the occupancy of playout buffer by *regulating the duration* of video frames (they present frames faster instead of dropping to reduce latency, and present frames slower to avoid underflows).

4.2.1 Influence of Media Type on Classification of Playout Adaptation

Differential media types play an important role for the classification of the playout adaptation methods. Normally, there are two media types: *continuous* or *semi-continuous.*

A *Continuous media stream* is a media type with a regular inter-MU interval, while *semi-continuous media* type is described by inactivity periods that intervene between periods of continuous MU flow. Some examples of continuous media are streaming video (live or stored) and streaming audio (Web radio). Spoken voice with silence detection is a typical example for semi-continuous media.

The important characterization of a semi-continuous media type is that the inactivity periods give the playout scheduler the opportunity to adjust the playout point for the imminent activity period without affecting its continuity. Unlike the semi-continuous media type, playout schedulers for continuous media do not have inactivity periods, but could act on inter-MU intervals to adjust the playout point.

4.2.2 Time-oriented Playout schemes

This section presents common time-oriented playout schedulers, that is, schedulers which timestamp MUs and which make use of local or global clocks to determine the presentation instant and duration of each MU.

4.2.2.1 Assured Synchronization under Bounded Delay Jitter

The most important role of playout adaptation is to restore the stream to its initial form and to eliminate any kind of distortion in the temporal relationships of MUs. This goal could not be achieved easily because MUs have variable network transfer delays. The network transfer delay contains two components: a static propagation delay and a variable queuing delay caused by variable waiting times in the queues of intermediate network nodes. If the delay variation is not limited or an infinitely long inter-arrival period may appear, the length of buffer becomes infinite for eliminating the discontinuities from the reconstruction process.

For the guaranteed service defined by IETF or constant bit rate CBR of ATM, the maximum delay difference is bounded by the network and that is why the total resynchronisation is feasible. This optimal synchronization is called *assured synchronization.*

For the implementation of assured synchronization two different approaches apply. The following notations are used to describe these two variants of the assured synchronization:

- $D_{n,i}$ is the network delay of the *ith* MU.
- $D_{b,i}$ is the buffering delay of the *ith* MU.
- $D_{tot,i}$ is the total end-to-end delay of the *ith* MU.
- $D_{n,\min}$ is the minimum network delay.
- $D_{n,\max}$ is the maximum network delay.
- $J_{\max}$ is the maximum different in any two network delays.

In [Nayl82, San93, Shiv95, Fran93], the authors say that if the *ith* MU causes a synchronization loss event when its network delay, $D_{n,i}$, is larger than all previous network delays, $D_{n,j}$: $1<=j<i$, plus the initial buffering delay for the first MU, $D_{b,i}$. Hence, in order to ensure that there is no loss of synchronization, it is necessary to guarantee that the first MU incurs a total delay $D_{tot,i} = D_{n,1} + D_{b,1}$ that is no less than $D_{n,\max}$. The total delay of the stream is then equal to the total delay of the first MU $D_{tot} = D_{tot,1}$.

The following two approaches, which differ from each other on the use of known or unknown $D_{n,1}$, are used for assured synchronization. This synchronization is

done by adding an appropriate $D_{b,1}$ to the first MU. The first method needs to have known $J_{\max}$, the second requires that $D_{n,\max}$ should be known.

Unknown $D_{n,1}$: In this case the scheduler could not measure precisely $D_{n,1}$. Thus in order to guarantee the assured synchronization, the scheduler must keep the first MU in the buffer for an interval equal to the maximum delay difference (i.e., $D_{b,1} = J_{\max}$) before the presentation of frames is initiated by extracting frames from the head of the playout buffer [Gey96].

This initial buffering delay protects the synchronization of the stream against the worst possible scenario, which corresponds to the first MU experiencing the minimum network delay, $D_{n,\min}$, while a subsequent frame experiences the maximum network delay, $D_{n,\max}$.

The total end-to-end delay of the stream is now $D_{tot} = D_{tot,1} = D_{n,1} + J_{\max}$, taking values in $[D_{n,\min} + J_{\max}, D_{n,\max} + J_{\max}]$, since $D_{n,\min} \le D_{n,1} \le D_{n,\max}$.

By doing that, the total end-to-end delay $D_{tot,1}$ of the first MU is not less than the maximum network delay $D_{n,\max}$, so it is guaranteed that no packet will arrive late.

Known $D_{n,1}$: This approach uses timestamps and a global clock that make the measure of the delay of the first MU possible. It adds the minimum buffering delay that makes the total delay of the first MU exactly equal to $D_{n,\max}$. The implementation in [Bald00] performs a potentially smaller end-to-end delay, while continuing to guarantee absolute synchronization at the receiver. Because of knowing $D_{n,1}$ before, the scheduler keeps the first MU in the playout buffer for an additional interval $D_{b,1}$ so that the total delay of the MU becomes $D_{tot,1} = D_{n,1} + D_{b,1} = D_{n,\max}$. It can guarantee that no MU will experience a larger delay; thus, no loss of synchronization will appear.

4.2.2.2 Allowing for Latency/ Synchronization Trade-off by Allowing for Loss due to MU Lateness.

The most important disadvantage of the assured synchronization method is the poor delay performance that makes this method inadequate for interactive applications, even the improved version of this method, which uses a global clock, produces a prohibitive end-to-end delay. Nowadays, almost modern packet

networks cannot guarantee an upper bound on delay [Paxs97, Kali94]. In addition, modern video and audio codecs can accept a substantial amount of packet loss with acceptable degradation of the perceived quality. This property of new applications leads to the fact that playout schedulers can accept a certain amount of lost MUs, in order to reduce the overall delay of the stream and effectively support real-time applications.

For bi-directional interactive real-time applications, the designed schedulers utilize precise timing information for providing a small constant total end-to-end delay D_{tot}. In [Bald00], the authors demonstrated that given a small network delay, a system can be configured (packetization, compression, rendering) to provide a total delay as small as *100ms*; and the scheduler will discard the packets that have a network delay larger than D_{tot}. D_{tot} controls the trade-off between intra-stream synchronization quality and delay. When the sender and receiver timestamps (used for measurement of the network delay) come from synchronized clock (GPS [Mills91]), it is guaranteed that any played MU will be delivered with an accurate D_{tot} (e.g., the Concord algorithm [Shiv95]).

4.2.2.3 Playout Mechanisms without Global Clock

For applications which demand that all MUs have a constant (small) delay or are dropped, it is necessary to use a global clock because it can provide the utmost interactivity precision. Unfortunately, it is not so easy to have a global clock in network; that is why almost the playout scheduler schemes, which do not require a constant end-to-end delay guarantee, operate on delay differences, and not on absolute delays. In this case, the two clocks need to run at approximately the same speed, and they need not to be synchronized, since their offset is cancelled when taking differences of timestamp values. The basic idea of these schemes is that the total delay of MUs are not constant or confined under an absolute value. In addition, this delay can fluctuate in response to changes in network delay variability, so a level of synchronization (e.g., percentage of late packets) or the more relaxed requirement of a constant trade-off between continuity and delay can be maintained. Network delay differences are used as indications of the current jitter level, and drive the regulation of the playout buffer.

An example of such schemes is done by Naylor and Kleinrock [Klein76]. In this method, the first packet of a talkspurt is delayed adaptively based on recent jitter measurements. In detail, the receiver logs the last m delays of MUs prior to the initiation of a new talkspurt and extracts the *k* partial range, *D(m,k),* which is the maximum difference between the m samples having first discarded the k largest delays. The first MU of the talkspurt is delayed for *D(m,k).* The partial range is used for eliminating isolated cases of extremely large delay that do not have a

significant impact on the probability of a late arrival. The relation between gap probability and delay is controlled by the level of conservatism in the partial range. Some packet lateness can occur because *D(m,k)* is smaller than the maximum network delay.

There are two methods for the handling of late packets: *method E* extends the delay of the stream by presenting late frames (*data-preserving* method) and *method I* preserves the delay of the stream by discarding late packets (*delay-preserving* method).

A well-known method in this field uses timestamps to approximate the one-way network delay $\hat{d}$ and its variability $\hat{v}$. The presentation time of the first MU of a talkspurt, p_i, is scheduled for:

$$p_i = t_i + \hat{d}_i + \beta.\hat{v}_i \tag{4.1}$$

when t_i is the generation time of the *ith* MU according to the sender's clock The coefficient *ß* controls the synchronization/latency trade-off. In [Ram94], the authors developed four algorithms that differ only in the way they derive the estimate $\hat{d}_i$. The evaluation is performed on a per-packet basis, using as input the network delay of the *ith* MU, d_i, while delay adjustments are applied on a per-talkspurt basis. d_i is the difference between the arrival timestamp a_i and generation timestamp t_i:

$$\hat{d}_i = \alpha.\hat{d}_{i-1} + (1-\alpha).d_i \tag{4.2}$$

$$\hat{v}_i = \alpha.\hat{v}_{i-1} + (1-\alpha).\left|\hat{d}_i - d_i\right| \tag{4.3}$$

Two of the proposed algorithms are based on the linear recursive estimator of the above equation. The third algorithm is adopted from the NEVoT (Network Voice Terminal) audio tool, and a fourth is a novel algorithm with delay spike detection capabilities and dual mode of operation, aimed at improving performance in delay environments with sharp delay spikes.

Moon et al in [Moon98] developed another algorithm that replaces the linear recursive estimators of $\hat{d}_i$ and $\hat{v}_i$ with the calculation of a percentile point *q* of the underlying network delay distribution. This is done by logging the last w packet delays (no clock synchronization) and using their *qth* percentile point as the playout delay for the next talkspurt.

4.2.2.4 Playout Schedulers with Approximated Synchronization- Virtual clocks

The playout adaptation with approximated clock synchronization can not maintain a delivery delay in absolute values because of not using a global clock. Such schemes produce a soft guarantee that is more specific than the freely fluctuating delay of differential delay systems, where the network delay component is completely unknown. The total delivery delay is bounded by measuring the round-trip time (*RTT*) between the communicating end points. This assures that no MU will be presented with a delay exceeding some expression that involves the *RTT*.

Roccetti *et al* in [Rocc01] use probe packets for measuring exactly the RTT between the communicating endpoints. Synchronization between the clocks of sender and receiver is realized by adopting the timestamps of the probe packets as local time. This immediately leads to a clock offset equal to the one-way network delay of the probe packet, t_0. A playout delay of t_0 would be too small, leading to increased packet lateness. Thus, the clock of the receiver is delayed by an additional RTT, as measured by the latest probe packet, that gives an overall time gap between the two clocks equal to $t_0 + RTT$, that is, the clock of the receiver falls $t_0 + RTT$ time units behind the clock of the sender. Packets that have timestamp larger than the local clock are buffered and packets that have timestamps smaller than the local clock are considered lost and are dropped. In the buffer, if the local clock equals their timestamp, packets will be extracted. The algorithm controls the playout point in accordance with the current network delay by refreshing the *RTT* every second.

Another similar example is done by Alvarex-Cuevas et al [Alv93]. The authors use also a probe packet, but rather at the beginning of every silence period. After measuring *RTT, RTT/2* is considered as one-way delay of the talkspurt and is sent to the receiver. At the receiver, this *RTT/2* is used as the estimate of the network delay and add an additional delay component so that a fixed target end-to-end delay D_{tot} that results in only *1%* packet lateness.

The additional delay up to D_{tot} should be $J_{\max}$, where $J_{\max}$ denotes the maximum delay variability ($D_{n,\max} - D_{n,\min}$). However, $J_{\max}/2$ is not known a priori, but is approximated and corrected by observing the extent of synchronization errors and increasing D_{tot} accordingly. By dynamically measuring *RTT* and adjusting D_{tot}, the algorithm adapts to network delay fluctuations and maintains the targeted synchronization quality. In the same work, a second method of estimating the target D_{tot} is described. It identifies the ‘’fastest’’ packet - the one with the

smallest delay (propagation only) - by looking for the packet that incurs the largest waiting time in the buffer; it is assumed that this is the fastest packet. D_{tot} is approximated as the network delay of the fastest packet plus the largest observed difference in buffering delays, which should approach $J_{\max}$, resulting in approximately *1%* packet lateness. Both methods can be improved by applying the method of hop-by-hop network delay accumulation, which results in very accurate network delay estimations (only the local clocks of intermediate nodes are involved; each node accumulates its own added delay [Mont83]).

4.2.2.5 Non Delay-preserving Playout Schemes

A *delay-preserving* method is a method that does not produce late MUs, and a *non-delay-preserving* method may have late MUs instead of dropping them for protecting the continuity of the stream from further degradation.

Figure 13 illustrates the different regions of a scheduler. In delay-preserving playout schedulers, the arrival time of a MU could fall into two regions: the *acceptance region* and the *discard region*. The acceptance region is bounded by the targeted end-to-end delay where an MU waits in the playout buffer for its playout time. The discard region is for arrivals with a total delay longer than the targeted D_{tot}.

C. Liu et al in [Liu96] introduce the no-wait region, which lies between the other two regions. The packets arriving in this region will be extracted immediately. An arriving MU with delay that is placed in the no wait-region is an MU that has missed its targeted D_{tot}, but not enough to be discarded, so it is played immediately to prevent further degradation of synchronization caused by MU discard.

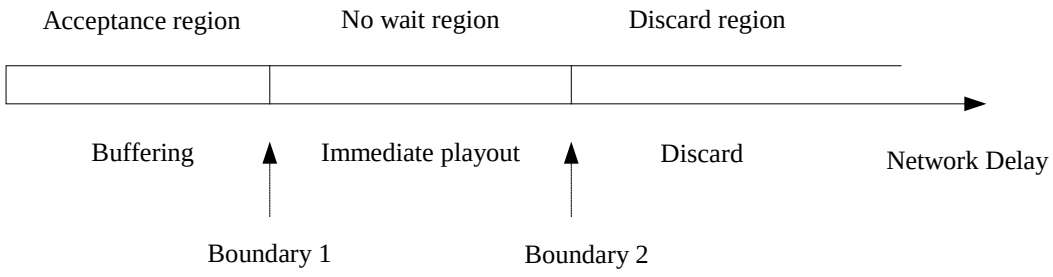


Figure 13 Network delays fall in one of three possible regions.

MUs are buffered if they arrive early (D_n in the acceptance region), presented immediately if they are slightly late (D_n in the no wait region) and discarded if they arrive too late (D_n in the discard region).

The playout scheduler proposed by H. Liu et al [Liu99] is also a non-delay-preserving. After each late packet a synchronization recovery phase is used for the scheduler and the presentation for the duration of the late frame is determined during this phase. A full presentation duration is undesirable because it increases the end-to-end delay of subsequent frames. Otherwise, a truncated presentation-up to the scheduled presentation instant of the next frame might truncate the late frame excessively causing motion jerkiness that is easily detected by the end user. The scheduler has a limited minimum-frame duration so that motion jerkiness is not detectable and he can choose to apply it to a series of frames following the late arrival, thus progressively reducing the additional delay. This approach reduces delay successfully and at the same time protects the quality of intra-stream synchronization by introducing a rather ‘’mild’’ delay-control function. Another interesting feature of the scheduler is that it uses a second-order continuity metric called RMSE. A user-requested threshold RMSE is maintained by the scheduler across different transmission conditions by regulating the buffering delay accordingly.

4.2.3 Buffer-oriented Playout Schemes

Similar to time-oriented schedulers that use differential delay methods, buffer-oriented schedulers adjust the playout point by observing the occupancy of the playout buffer. The lack of timing information precludes any kind of absolute total end-to-end delay guarantees for the MU presentation epochs. The only ‘’visible’’ delay component for the scheduler is the buffering delay of the MUs at the playout buffer.

Various buffer-oriented schemes which differ in the trade-off between media continuity and buffering delay are developed. The total end-to-end delay although unknown (and fluctuating) can be controlled as a consequence of the regulation of the buffering delay component; the suppression (expansion) of the buffering delay leads to the suppression (expansion) of the total end-to-end delay with a subsequent cost (gain) in intra-stream synchronization quality.

With this method, delay guarantee can be approached but is not guaranteed in absolute values. Due to this uncertainty concerning the interactivity of the system buffer-oriented schedulers are usually applied in video applications where the interactivity requirements are more relaxed than in audio applications.

4.2.3.1 Non initial Buffering: Self-adjusting and Bufferless schedulers

This method is a good solution (if there is no jitter) because it produces a potentially low initial delay (only the network part) by presenting the first MU as soon as it arrives. But when jitter exists, the first underflow will occur as soon as some MU experiences a network delay greater than the delay of the first MU. The self-adjusting schedulers present all MUs that arrive at the receiver; thus, the playout buffer builds up as a natural effect of the induced underflows (since the mean arrival rate equals the mean presentation rate and an underflow is analogous to a ‘’server vacation’’). This leads in a stream presentation with a small initial delay (no initial buffering) but also an initially poor synchronization quality (frequent underflows); continuity improves with time, since the jitter buffer expands with underflows, but this also increases the perceived delay. For regulating delay some of the MUs must be discarded. Delay regulation depends on the current buffer occupancy.

4.2.3.2 Buffer Occupancy Control: Queue Monitoring and Watermark-based Schedulers

Stone and Jeffay in [Ston95] show that it is able to measure the impact of delay jitter on a receiver by observing the occupancy of the playout buffer over time. This policy is called as *Queue Monitoring* (QM). With QM, a continuous sequence of video frames has the meaning that the queue was never found empty following the completion of a presentation. In addition, this continuous sequence of frames is used as indication of reduced delay variability and triggers a reduction of the end-to-end delay of the stream by discarding the newest frame from the buffer. By selecting the duration of the gap-free interval, it is possible to control how aggressively QM tends to reduce latency and is thus the synchronization/latency trade-off parameter. For deciding which frame should be discarded, a series of thresholds and associated counters is used. Increasing network jitter causes buffer underflows and naturally increases the occupancy with the acceptance and presentation of ‘’late’’ frames. That is, QM is to some extent *data-preserving*, on some occasions presenting late frames.

For the adjustment of delay, QM uses a window mechanism; however, many buffer-oriented schedulers are given the freedom to adjust the playout point in a per-MU fashion [Roth95, Bier96, Yua96, Laout1]. The authors in [Roth95] introduce the idea of occupancy *watermarks* (high-watermark, HWM, and low-watermark, LWM) in order to define a range of desired playout buffer occupancies that balance the risk between buffer underflow and overflow. A *targeted area*, lying between the HWM and LWM levels, is defined by the upper target boundary (UTB) and lower target boundary (LTB). The positioning and

width of the targeted area (inside the watermark limits) reflect the desired synchronization/ delay compromise. For example, a minimum delay policy sets the LTB equal to the LWM and the UTB to a slightly larger value. When the occupancy of the buffer falls outside the targeted area-in the so called critical buffer regions- the scheduler enters an adaptation phase with the aim of returning the occupancy inside the targeted area. This is accomplished by modifying the receiver's consumption rate until the occupancy returns in the targeted area. The width of the targeted area determines the aggressiveness of the buffer control algorithm, while the selection of watermarks mostly contributes to the data loss rate. The watermarks are fixed in [Roth95], but it is noted that the scheduler can be make adaptive by dynamically regulating the watermarks in response to jitter effects (e.g., MU loss).

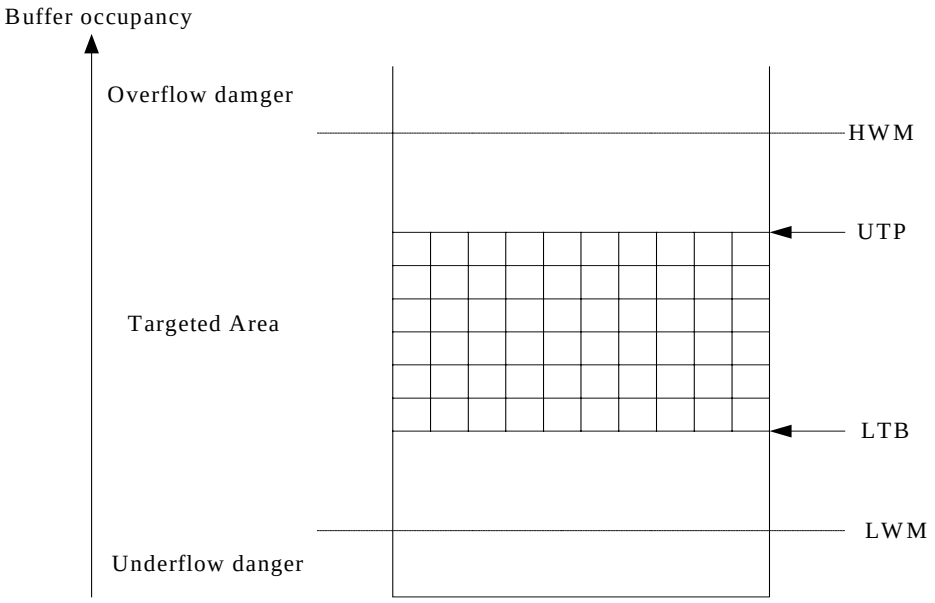


Figure 14 The watermark-based playout scheduler of Rothermel and Helbig.

The selection of watermarks mainly affects the underflow and overflow probabilities. The positioning of the target area, inside HWM and LWM, regulates the trade-off between intra-stream synchronization and stream latency.

Another method of Biersack *et al* [Bier96] uses the rate adaptation mechanism at the sender (which is a VOD server) rather than at receiver. In order to smooth out occupancy fluctuations caused by short-term jitter the receiver uses a gradient descent estimator of the buffer occupancy. If the smoothened buffer occupancy is within the *critical region* (outside LWM. HWM) then the receiver sends a signal to the sender suggesting that the latter should adjust its rate so that the smooth occupancy returns into the targeted area. The sender either skips some frames or pauses (pause/drop method) to adjust its rate. Source-rate adaptation that affects

the encoding process could also be used but the author does not consider it due to high implementation complexity.

In [Orion00], a low complexity algorithm for detecting clock skew in network audio applications that function with local clocks and in the absence of a synchronization mechanism is presented. In addition, a companion algorithm to perform skew compensation is also described. For the skew detection algorithm, high and low water mark are employed in order to limit the number of compensating actions to keep the computational cost low and to reduce sensitivity to the transient transit variations. When the skew detection algorithm indicates that the number of samples in the receiver´s playout buffer requries adjustment, the compensation mechanism is applied.

4.2.3.3 Buffer Occupancy Control with Dynamic Regulation of the Duration of MUs

The above schedulers base on the *slotted* approach in the regulation of the buffering delay. That means they increase or decrease it in constant amounts that equal the duration of an MU. Discarding ‘’late’’ frames [Bier96] and the *tail-drop* from overbuilt queues [Ston95] lead to sharp delay reduction jumps of duration *T*, equal to the duration of a video frame. And when the playout buffer empties, the presentation resumes after one or more MU periods [Bier96]. This approach has the advantage: easy implementation, but it can be also quite crude, especially in the case of low-frame-rate streams where the slot (video frame) has a significant duration.

In [Yua96, Yua97] Yang and others improved the perceptual quality of video achieved by a fine-grained regulation of playout durations based on the current occupancy of the playout buffer. Another method, called *threshold-based* is also proposed in [Yua96]. This scheme uses reduced playout rates aimed at avoiding large underflow discontinuities as the buffer occupancy i drops below a threshold value TH. The selection of TH is done prior to stream initiation and remains unchanged despite jitter fluctuations; it governs the trade-off between stream continuity and reduction of playout rate. Stream continuity is described by two disjoint metrics: the probability of an empty buffer and the frame-loss probability due to buffer overflow. The work has been enhanced in [Yua97] by introducing a dynamic playout scheduler that uses a *window* to optimise some quality metric by responding to changing network jitter conditions. The window is actually a time-varying dynamic version of the threshold approach. A neural network (NN) traffic predictor and an NN window determinator are being used for online estimation of traffic characteristics and for the regulation of window size. The value derived for the window is compared to the current buffer occupancy resulting in the selection of playout durations for the buffered frame. Stream continuity is described by a

second-order metric, the variance of discontinuity (VOD), which accounts for underflow occurrences and discontinuities due to reduced playout rates. A number of playout schedulers are derived, each providing a different trade-off between playout continuity (captured by VOD) and reduction of mean playout rate.

The method in [Laout1] is an extension of Laoutaris and Starvrakakis that uses a compact and fair continuity metric *distortion of playout* (DoP). The definition of DoP has been motivated by experimental perceptual results for video transportation over packet network s conducted by Claypool and Tanner [Clay99], reporting that jitter degrades the perceptual quality of video nearly as much as packet loss. The study has limited the range of the threshold parameters *TH* by identifying a range of values when there is no beneficial trade-off between continuity and reduction of mean playout rate - the two antagonistic metrics of the Internet. Interestingly, it has been shown that this range of values changes with the burstiness of the frame arrival process, revealing the danger of an initially meaningful *TH* appearing in the undesirable area due to a change of arrival burstiness. Finally, the work is supplemented with online algorithms for the detection and maintenance of the operational parameter *TH* within the area of beneficial trade-off across unknown non stationary delay jitter.

Finally, by solving an appropriate optimisation problem, Laoutaris and Stavrakakis have developed a scheduler that outperforms the earlier scheduler of [Laout2]. Stream continuity is described by using the first two moments of the DoP metric. This approach allows a fine-grained optimisation of stream continuity by catering to a combination of the expected frequency of synchronization loss and its appearance pattern. It is noticed that the minimization of the expected value of DoP and of the variability of DoP are two contradicting objectives. It is concluded that for a perceptually optimal result the scheduler must be allowed to increase the frequency of discontinuities, if this increase is providing a smooth spacing between discontinuity occurrences and thus helps in concealing them. The Markov decision theory is applied for the derivation of the optimal playout policy for some common levels of network jitter, a playout scheduler can use a jitter estimator and adaptively ‘’load’’ the appropriate offline-computed optimal policy and thus can approach the optimal performance in a dynamic environment with low complexity (no online optimisation required).

4.2.4 Comparisons of Playout Buffer Delay Adjustment algorithms

In general, time-oriented schedulers are prefered when there is an interactivity requirement, in most cases in systems that handle spoken voice. Buffer-oriented systems are employed in video communication systems, where some compromise in delay is acceptable, even in interactive systems, if this is to provide for a

smooth presentation of frames. Bidirectional, interactive audio applications usually implement time-oriented playout schedulers.

Streaming of stored content can be carried out by using the assured synchronization under bounded delay jitter algorithms which provide for an absolute re-synchronization at the lowest possible end-to-end delay. In the real world, implemented systems seldom use these algorithms, mainly because they induce an initial delay that is up to the maximum network jitter which in most cases is unknown. The assured synchronization under bounded delay jitter algorithms provide no packet late (loss rate is 0%), and this is the most ideal case, but they are considered inadequate for interactive applications due to the total stream delay is seconds, but the acceptable delay is only milliseconds. Furthermore, the algorithms that do not know the network delay of the first packet are unflexible due to fixed end-to-end delay, while the assured algorithms that know this delay before are more efficient to be implemented than the previous case and their end-to-end delay is reduced.

Unlike the assured synchronization approach, the playout schemes that allow for the latency/synchronization tradeoff by allowing for loss due to MU lateness can have loss rate. Hence they can support real-time applications better because the overal end-to-end delay is decreased. It is also necessary to note that both the assured synchronization and this class of time-oriented scheme need to have a global clock in order to determine the network delay of each packet.

The other type of time-oriented schedulers is the scheme that does not require a global clock. All of these playout schedulers are based on per-talkspurt basis and they try to determine the playout buffer time based on network jitter estimated. These schemes differ from each other only in the way of estimating this network jitter.

In addition, schedulers with approximated clock synchronization fill the gap between the two extreme approaches. Such systems do not require a global clock, so they cannot guarantee a delivery delay in absolute values, but they provide a soft delivery guarantee that is more specific than the freely fluctuating delay of differential-delay systems, where the network delay component is completely unknown.

The following table summarizes the characteristics of different time-oriented playout schemes.

Author	Media	Global clock	Delay Performance	Synchronization Performance	Delay Adaptation
Geyer [Geyer96]	Audio/video	Not assumed	$D_{n,1} + J_{max}$	No loss of synchronization	Delay static during connection
Baldi [Bald00]	Audio/video	Assumed	$D_{n,max}$	No loss of synchronization	Delay static during connection
Concord [Shiv95]	Audio/video	Assumed	variable→min	Static-guaranteed	Based on PDD estimation
Naylor and Kleinrock [Klein76]	Audio+sil. Det.	Not assumed	Stat. tradepff	Stat. tradeoff	Per talkspurt (partial range filter)
Ramjee [Ram94]	Audio+sil. Det.	Not assumed	Stat. tradeoff	Stat. tradeoff	Per talkspurt (recursive filter)
Moon [Moon98]	Audio+sil. Det.	Not assumed	Stat. tradeoff	Stat. tradeoff	Per talkspurt (percentile point)
Roccetti [Rocc01]	Audio+sil. Det.	VC, t_0 + RTT	Stat. tradeoff	Stat. tradeoff1	Periodic (1sec) (RTT based)
Alvarez-Cuevas [Alv93]	Audio+sil. Det.	VC, RTT/2	variable→min	1%MU lateness	Per talkspurt (RTT based)
C. Liu [Liu96]	Audio/video	VC	variable→min	Satisfy user input	Per MU (delay region based)
H. Liu [Liu99]	Audio/video	VC	variable→min	Satisfy user input	Per MU↑, per *W* MUs↓

Table 1 Overview of time-oriented schedulers. The abreviation VC indicates Virtual Clock synchronization method, with some approximated offset. The label stat. Tradeoff identifies systems where a constant tradeoff between synchronization and delay is maintained across different levels of jitter.

Bidirectional video applications are usually less demanding, as far as interactivity is concerned, compared to their audio counterparts. The buffer-less approach is very simple and provides for the best interactivity (frames are displayed as soon as they arrive), but the synchronization quality quickly degrades with jitter, as there is no dejitter buffer. The self-adjusting buffer is also quite simple to implement, and assuming a small amoount of jitter, provides a very good synchronization/delay tradeoff as the buffer quickly adjusts to an occupancy that eliminates all jitter, at a small delay. The downside is that it is sensitive to rate occurrences of unusually large jitter; in such cases the delay of the stream will rise and willl remain large since there is no delay control function to restore it. Neither scheme employs smoothing of long-lasting discontinuities. Dynamic regulation of MU durations can be used to improve the intrastream synchronization quality. The threshold-based schemes are also quite simple to

implement while systems that require some offline optimization are more complex.

The following table overviews the buffer-oriented playout schedulers:

Author	Buffer control	Delay performance	Intrastream sync. mectric	Evaluation
Stone and Jeffay (QM) [Ston95]	Tail dropping	Try to reduce	Gap freq., 1st order	Experimental
Rothermel and Helbig [Roth95]	Adjust playout rate	Predefined (static tradeoff)	Gap prob., 1st order	Simulation
Biersack [Bier96]	Adjust source rate	Predefined (static tradeoff)	Gap prob., 1st order	Experimental
Yuang [Yua96]	Static threshold	Threshold dependent	Gap prob., 1st order	Analytical
Yuang [Yua97]	NN dynamic threshold	Variable delay	2nd order	Simulation
Laoutaris and Stavrakakis (Laout1]	Adaptive threshold	Small initial increase	2nd order	Analytical
Laoutaris [Laout2]	Offline optimal policy	Policy dependent	Combined 1st, 2nd order	Analytical

Table 2 Overview of surveyed buffer-oriented schedulers. In the buffer control column, a *slow (fast)* in parentheses, denotes that the scheduler is able to apply reduced (increased) playout rate.

Among all these playout schemes, I see that the Concord algorithm is an very good example in order to be implemented in our system, because it plays a direct tradeoff between end-to-end delay of system and the loss rate predefined at the playout buffer. By adjusting the loss rate at playout buffer, I can control the end-to-end delay directly. Furthermore, the Concord algorithm can remove the short-term jitter produced in the network and hence it can improve the quality of interactive applications considerably.

4.3 Related Works

There are two important research areas that play an important role for the playout adaptation: Forward Error Correction (FEC) and Video Caching (or proxying). FEC and its coupling with playout adaptation is a research topic that has recently attracted much attention perhaps due to the fact that FEC plays a significant role in enabling packet-video communications in wireless environments. Video caching appears to be a very attractive way to provide high-quality non-interactive streaming content in a cost-effective manner.

4.3.1 Influence of FEC on Playout Schedulers

A lot of adaptation algorithms do not pay attention on packet losses that appear in the network due to congestion. Network losses are assumed to be out of the scope of playout adaptation, and compensation for losses is left to various FEC mechanisms [Carl97, Perk98, Liu97] that operate in isolation from the playout-adaptation algorithm. Recently research demonstrated that a considerable performance gain can be expected from combining delay-oriented playout adaptation and loss-oriented FEC [Ros00, Hart00].

Rosenberg et al. [Ros00] study the effect of *(n-k)* Reed-Solomon correction codes on existing and new playout algorithms and they show that performance improvement is achieved when considering the coupling between jitter and loss compensation. Existing playout algorithms [Ram94, Moon98] are made FEC-aware by substituting the network delay *(D_n)* of a packet with the virtual network delay *(VD_n)*, which is either D_n or the extended recovery delay (recovery time-generation time). If no error does occur in the network the recovery time of an MU coincides with its arrival time; otherwise the recovery time is the time when the reception of some redundant FEC packet allows the correct decoding of the corrupted (or lost) MU. The targeted end-to-end delay D_{tot} is shaped by VD_n which depends on the FEC algorithm and thus the coupling. Several new adaptation algorithms have been proposed. The adaptively virtual algorithm targets a desirable packet loss probability. It is based on [Ram94], and uses virtual delays to dynamically adjust the variation multiplier *b* of Equation (4.1) in order to achieve a targeted loss rate. Another algorithm is the so called Previous Optimal algorithm; it determines the optimal (minimal) delay for the previous talkspurt such that a specific application loss rate be maintained, and applies it to the next talkspurt; hence the Previous in the algorithm name. Finally, the authors describe an analytical framework for the expression of the total end-to-end delay as a function of the application-level reception probability (in the presence of FEC), the network delay distribution, and the network loss probability.

Intra-stream synchronization has been studied for wireless receivers by Liu and Zarki [Liu99]. In the wireless environment, the media synchronisation module must be coupled with the Automatic Repeat Request (ARQ) of the wireless link.

4.3.2 Influence of Video Caching on Playout Schemes

As with traditional data objects of established protocols like http and ftp, real-time objects (mostly stored video) are being cached, or proxied, with the aim of reducing network traffic and improving the interactivity of content delivery [Gar99, Wang01, Chua00, Kang01]. Unlike popular web proxies video proxies typically store a portion of a video clip only - the initial part usually (called the prefix [Gar99, Wang01]) - since the entire video object is too lengthy to be replicated on a typical proxy and replication is the most cost-effective alternative [Gar99]. Video proxies improve the quality of video delivery in many ways. First they reduce the network transfer delay since proxies are located closer to end clients, so the stream travels just a few network hops before delivery. Second, proxies assist in improvement of intra-stream synchronization quality. If the prefix of a video takes a long time within the range of minutes then the amount of proxied data completely smoothes out the jitter in the data path from the server to the proxy. This is important because it relieves a receiver of a large fraction of jitter - only the jitter in the access part remains from the proxy to the receiver which is usually small and easily can be smoothed out with a small playout buffer at the receiver. The reduction of the playout buffer also reduces total delay thus improving the responsiveness of the service. Even if the amount of prefix corresponds to the duration on the same timescale as the network jitter at the core network then it will again absorb some portion of the delay variability and thus decrease the size of the playout buffer at the receiver. For a given synchronization quality the existence of the prefix helps by hiding some portion of the total delay.

4.4 Performance of a Playout Schemes

In order to compare one adaptive playout-buffer delay-adjustment algorithm with another the trade-off between average playout delays and loss is used as a performance measure. These parameters should be considered on per-packet rather than per-talkspurt because the lengths of talkspurts depend on silence detection algorithms. Per-talkspurt playout-buffer delay is thus closely tied to silence detection algorithms used. More importantly, different talkspurts have different lengths.

End-to-end application delay is defined as difference time between playout time at receivers and generation time at senders. Figure 15 shows timing information of audio packets and formally defines average playout delay [Moon98].

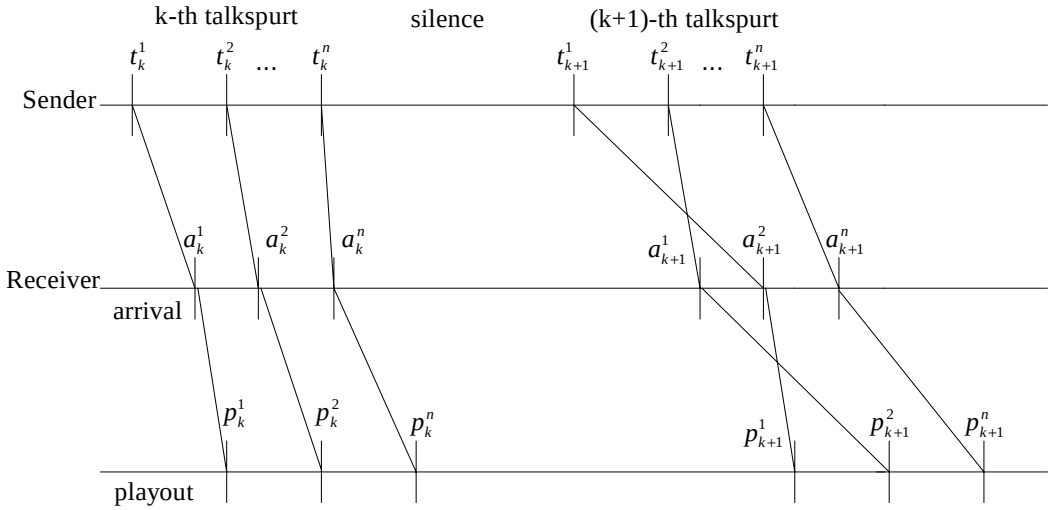


Figure 15 Timing associated with the *i-th* packet in the *k-th* talkspurt

Suppose that there is a trace consisting of *M* talkspurts. The following quantities are defined:

- t_k^i : sender timestamp of the *i-th* packet in the *k-th* talkspurt.
- a_k^i : receiver timestamp of the *i-the* packet in the *k-th* talkspurt,
- n_k : number of packets in the *k-th* talkspurt. Hence I only consider those packets actually received at receivers.
- *N*: total number of packets in a trace.

$$N = \sum_{k=1}^{M} n_k$$

The algorithm used in receivers to estimate playout delay of the packet decides its amount of playout time. Suppose that the playout algorithm is *A*. Then $p_k^i(A)$ is the playout timestamp of the *i-th* packet in the *k-th* talkspurt under A. If the *i-th* packet of the *k-th* talkspurt arrives later than $p_k^i(A)$ (i.e., $p_k^i(A) < a_k^i$), it is considered lost. Otherwise, it is played out with the playout delay of ($p_k^i(A) - t_k^i$). Let $r_k^i(A)$ be an indicator variable for whether the *i-th* packet of the *k-th* talkspurt arrives before its playout time, as computed by playout algorithm *A*:

$$r_k^i(A) = \begin{cases} 0, p_k^i(A) < a_k^i \\ 1, otherwise \end{cases}$$

N(A) is denoted as the total number of packets played under algorithm A and computed using $r_k(A)$:

$$N(A) = \sum_{k=1}^{M} \sum_{i=1}^{n_k} r_k^i(A)$$

Then the average playout delay of those played-out packets is defined as:

$$\frac{1}{N(A)} \sum_{k=1}^{M} \sum_{i=1}^{n_k} r_k^i(A)(p_k^i(A) - t_k^i)$$

If there are *N* packets in a trace and, among them, *N(A)* packets are played out under algorithm *A* the loss percentage *l* is:

$$l = \frac{N - N(A)}{N} * 100$$

4.5 Concord algorithm

4.5.1 Why Concord?

The Concord playout buffer delay adjustment algorithm described in [Shiv95] constructs a probability delay distribution (PDD), an estimate of the probable delays suffered by packets in the network over a time window. This PDD may draw on existing traffic conditions, history information or any negotiated service characteristics to derive estimate for minimum, maximum and/or mean delay distributions. From the PDD distribution, this algorithm will determine the total end-to-end delay based on a given loss rate. According to this end-to-end delay, it then adjust the playout buffer delay of each packet is adjusted**.**

In other words, the Concord mechanism try to reduce short-term jitter by performing a direct trade-off between loss rate at receiver and the total end-to-end delay. The direct trade-off between jitter produced by network, total end-to-end delay and loss rate of the Concord algorithm leads me to an interesting idea: design a new relative DiffServ model that controls the proportional jitter in the networks and verify the influence of this jitter at playout buffer on the total end-to-end delay.

Furthermore, the Concord algorithm anticipates short-term network delay with the aim of not responding too quickly to short-lived variations. This algorithm is considered as a very suitable scheme for applications, which do not tolerate high

delays but conceal a small amount of late packets. Nowaday, a lot of voice and video coding algorithms can reach satisfactory output, and hence the expense of increasing the loss rate of the Concord algorithm causes no negative impact for voice and video-coding applications. That means using of Concord algorithm at receiver end is very appropriate for the DiffServ network that provide different types of voice service. Finally, it is very remarkable, because it defines a single framework to deal with both forms of synchronization, and operates under influences of parameters, which can be supplied by the applications involved.

These reasons leads me to choose Concord algorithm as playout buffer delay adjustment algorithm in the receiver of my network.

4.5.2 Basic Characteristics

This section summarizes some features of the Concord algorithm, which uses a predictive approach to playout-buffer management.

Name	Description
PDD	Packet Delay Distribution
ted_s	Total end-to-end delay for stream *s*
mad_s	Maximum acceptable delay for stream *s*
bs_s	Buffer size for stream *s*
nd_s^i	Network delay for packet i of stream *s*
bd_i^s	Buffer delay for packet i of stream *s*
mlp_s	Maximum late packet (%) for stream *s*
alp	Actual late packets (%)
Cdf	Cumulative distribution function
$H_i(x)$	Function of histogram after its aging
$P_i(x)$	PDD function after the aging ($0 \le P_i(x) \le 1$)
F	Aging Factor
S	Sum of bin value in histogram
c	Aging coefficient $0 \le c \le 1$
f	Aging frequency (in packet spacing)
r_1	Ratio of old aged data to the newly arrived packet
r_2	Ratio of old aged data to subsequent packets until next aging

Table 3 Basic notation

4.5.2.1 Function

The Concord algorithm describes a stream as a sequence of packets, which are produced within periods and which are marked with sequence numbers. For each stream there are two important parameters: the maximum acceptable delay (mad_s) it can suffer, and the maximum late-packets (mlp_s) percentage it can accept or tolerate. In addition a packet-delay distribution (PDD), which is a statistical representation of network delays for packets in that stream is also set up. This approximate distribution is reconstructed or updated periodically so that it adapts to actual situations. The speed of getting this process established can be accelerated by having an initial approximation for the Packet Delay Distribution, perhaps based on recent observations.

The most basic and important job of the Concord algorithm is to determine the minimum buffer size at receivers so that the network jitter is smoothed out and the requirements of the maximum adaptive delay mlp_s and the maximum late packet mad_s are satisfied. In other words, it is necessary to find the minimum buffer size bs_s which satisfies the following conditions:

- For every packet *i*: $nd_i^s + bd_i^s$ is a value of ted_s, where ted_s is the *total end-to-end delay*, nd_i^s is the *network delay* suffered by packet *i*, and bd_s^i is the induced *buffer delay* for *i*.
- The chosen ted_s is less than the mad_s of the stream
- The chosen ted_s does not lead to more than mlp_s percent of packets being thrown away

Figure 16 shows an example of calculating total end-to-end delays form the PDD. With this example, it is easy to recognize that if a packet has a delay higher than *ted*, it will be declared as late. From this property, the amount of late packets will be *(1-Cdf(ted))*, where *Cdf* is the *cumulative distribution function* on the Packet-Delay Distribution. The behavior of the Concord algorithm is analysed by choosing a value of *ted* so that either *mlp* or *ted* is minimized (best visualized by moving the ted line in Figure 16 to the right or left, respectively).

Concord chooses the value for *ted* by using available information on delay probabilities such that the required conditions for *mlp* and *mad* parameters, which are supplied by application are satisfied. In this section the dynamic mode of the operation of Concord is analysed under the condition that the value of *ted* is re-updated from time to time. Basically, this algorithm demands receivers to

construct and maintain a historical record of observed *nd* (network delay) values in the form of a measured histogram. Over time, the quality of the algorithm is improved as this structure is set up. So obvious optimisation possibilities exist by initialising it appropriately - if reasonable information is available. The recorded historical information is used from time to time to revise the actual *ted* if necessary to reach the application *QoS* parameters.

For controlling the dynamic behaviors of the Concord algorithm, two parameters are used. The first one is a *threshold factor* which decides when *ted* is recalculated. The second controls relevance of the histogram data by aging its contents over time with the aim of more accurate behavior. The execution overhead can be reduced by choosing a threshold factor, which results in a smaller number of *ted* recalculations - but of course this may result in decreased effectiveness.

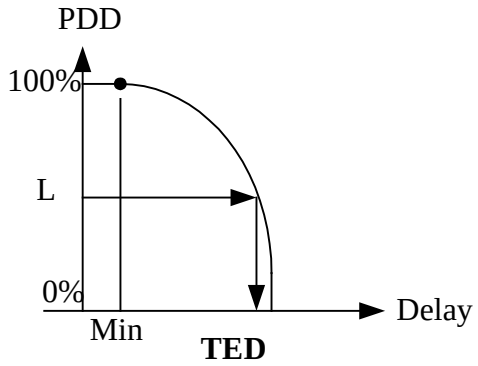


Figure 16 PDD constructed by Concord algorithm

4.5.2.2 Control of Statistical Historical Information

In order to process statistical trends, several approaches based on observed measurements are investigated. Among these researches *full aggregation* is an example, whose data is recorded and accumulated into a probability distribution over a window time in which the algorithm is running. This type of processing statistical data gives same weights for recent information and old information in terms of their influence on the probability distribution. Hence this reason leads it to less adaptive to changes of systems. For overcoming this disadvantage of the full aggregation approach, the *flush* and *refreshes* method is created. This scheme stores statistical samples for a period of time, then flush and refresh them periodically. However, in this scheme the periodic flush results in a complete loss of historic information and can introduce boundary effects at the flush instances.

It is necessary to create an appropriate statistical scheme, which can anticipate future network delays by analysing historical and current information. This scheme must examine, maintain, register and rewrite the statistical bias of network delays. In Concord - by using a measured histogram in order to approximate the PDD function - network tendency is recorded and tracked. This histogram bins store frequency of delay patterns where *bin width* means the range of network delays grouped together to represent one pattern in the histogram. It is clear to recognize that the increase of *bins* leads to a decrease of the width of the *bins*. Because conditions of network change by time this historical information will also vary widely and the current information is very important for the Concord algorithm to be successful. Thus, it demands an update and operation for each *bins* in order to reduce effects of older information. This activity of the Concord algorithm is called *aging*, which is described in the next section.

To allow accurate precise delay estimation, the aging operation ages the older samples gradually. In other words, the older information is not discarded but gradually *retired*. The balance between *bin width* and accuracy is then discussed in a subsequent section.

4.5.2.3 Aging Function

This function contributes a very important part to the success of the Concord algorithm. For this function, it is necessary to have a frequency called *aging frequency*, which is set up by users or applications in order to determine how often the aging function should be called. If users and applications set it at high frequency that means that data is updated in short time scale and the system is able to react quickly to changes - otherwise the reaction to changes is very slow.

Several approaches exist for the implementation of aging functions. A simple method could discard all data prior to a certain threshold, which is a moving window of fixed size to the current time. In addition this threshold could be based on time or on a packet count. This solution maintains better history information than the flush and refresh method, and hence it contains a high overhead. Such methods are very difficult to implement because it requires complete data to be kept rather than statistical approximation. Furthermore as addressed before this approach does not discard the older data entirely but reduces its effect on the statistical distribution gradually by periodically scaling down the existing distribution by an *aging coefficient*. This aging coefficient is determined by a user or an application while continuing to add new sample data with a constant weight. This decreases progressively the influence of the older samples and gives the newer ones larger effect.

The Concord algorithm uses aging coefficients provided by users or applications as a corresponding *aging factor*. This factor can be interpreted as a mathematical quantity used in order to scale the PDD histograms. Depending on how to translate this factor, there are three different algorithms, which are described later. The value of *aging coefficient* is chosen as a number between 0 and 1, which is determined by application and is used to diminish the effect of older statistical date. The *aging factor* is defined as a scaling factor used by Concord to scale down the value of each bin in histogram.

The aging function could be calculated based on a number of packets received. One example is applying aging to every packet arrival, while another one could age the statistical data every 1000 packets and increment the bin corresponding to each packet's delay by one whenever a packet arrives.

There are three aging algorithms, which are described next. Algorithm 1 is the basic method while algorithm 2 is the improved version of algorithm 1 and algorithm 3 is a variant of algorithm 2.

Algorithm 1: This algorithm defines aging factor as:

$$F=c$$

Where c is the aging coefficient, and F is the aging factor.

As described in the previous section, *aging factor* is a real value, and this value is used to scale down each bin while *aging coefficient* is provided by users or applications, which are identical in this algorithm. However, this relation will not be the same in algorithm 2 and 3.

In this approach, the total count is readjusted by the aging coefficient. Then the algorithm extends the bin corresponding to new packet's delay by one. After aging, all subsequent packets to arrive cause the corresponding bin to be incremented by one until aging happens again.

Suppose that $H_i(x)$ is the function of the histogram after the *ith* aging and *nd* is the network delay of new packet. The following relation is received:

$$H_i(x) = \begin{cases} F * H_{i-1}(x) 0 \le x < \infty, x \neq nd \\ F * H_{i-1}(x) + 1, x = nd \end{cases} \quad (4.5)$$

The ratio of old aged data to the newly arrived packet is defined as r_1:

$$r_1 = c * \int_0^{\infty} H_{i-1}(x)dx \tag{4.6}$$

The ratio of old aged data to the newly arrived packet is defined as r_2 :

$$r_2 = \frac{c}{f} * \int_0^{\infty} H_{i-1}(x)dx$$

where *f* is the aging frequency in packet spacing. Let $P_i(x)$ be the Packet Delay Distribution function after the *ith* aging, hence:

$$P_i(x) = \begin{cases} \dfrac{F * H_{i-1}(x)}{S}, 0 \le x < \infty, x \ne nd \\ \dfrac{F * H_{i-1}(x) + 1}{S}, x = nd \end{cases} \tag{4.7}$$

where $S = \int_0^{\infty} H_i(\hat{x})d\hat{x}$

For implementing, the continual function $(\int)$ could be calculated by a discrete function $(\sum)$. In addition, y can be limited to the upper and lower bounds of network delay, which can be rewritten whenever every packets come in. $H_i(x)$ is a function of the actual number of packets in the histogram. $P_i(x)$ normalizes $H_i(x)$ so that $\int_0^{\infty} P_i(x)dx = 1$. The following conditions must be satisfied by *mlp*:

$$mlp \ge \int_{-\infty}^{\infty} P_i(x)dx = \begin{cases} \dfrac{F}{S} \int_{-\infty}^{\infty} H_{i-1}(x)dx, nd < t \\ \dfrac{F}{S} \int_{-\infty}^{\infty} H_{i-1}(x)dx \,|_{x \ne nd + \frac{F * H_{i-1}(nd)+1}{S}}, nd \ge t \end{cases} \tag{4.8}$$

In this condition, *t* is the chosen *ted* (Figure 16). c_1 and c_2 now are called two different aging coefficients. If *mlp* value is set the same for both of them, the following relation could be received when *nd*<*t* in equation (4.8):

$$c_1 \int_1^{\infty} H_{i-1}(x)dx = c_2 \int_2^{\infty} H_{i-1}(x)dx$$

Suppose that $c_1 > c_2 > 0$, this relation is:

$$\int_{t_1}^{\infty} H_{i-1}(x)dx < \int_{t_2}^{\infty} H_{i-1}(x)dx \tag{4.9}$$

In contrast, for the case where $nd \geq t$, this equation is obtained:

$$c_1 H_{i-1}(nd) + c_1 \int_{t_1}^{\infty} H_{i-1}(x)dx = c_2 H_{i-1}(nd) + c_2 \int_{t_2}^{\infty} H_{i-1}(x)dx$$

Because $c_1 > c_2 > 0$, the result of equation (4.9) can be derived for this case also. Thus, it is easy to conclude that $t_1 > t_2$ when $c_1 > c_2$ from equation (4.9). This means decreasing aging coefficient decrease the value of *ted*.

In this algorithm, the *bin* values are delayed or scaled down by the aging coefficient when aging happens. When the number of samples increases then the corresponding bin value also increases, the packet arriving after aging plays a less important roll to the histogram in comparison the old gagged data in histogram, which have just been scaled down (see equation 4.6).

I.e. if the total of histogram bin values is 10 this is scaled down by 0.9 after aging. The new packet then contributes one the histogram. The ratio of the old aged data to the newly arrived packet is 9. After some time, the sum of the bins in histogram may be 10000. After aging by 0.9, the ratio of the old aged data to the newly arrived packet is 9000, which is different with the previous ones. This can be seen in equation (4.6) that r_1 is changing based on the value of old aged data in histogram.

Algorithm 2: The idea of this algorithm is to maintain a constant ratio of old aged data to the newly arrived packet through the time of synchronization stream. When aging happens, the next packet will contribute one to the histogram, but the old statistical data prior to aging are scaled down by the following factor:

$$F = \frac{c}{(1-c) * \int_{0}^{\infty} H_{i-1}(x)dx} \tag{4.10}$$

By scaling down the old bins by equation (4.10), the ratio of old aged data to the newly added bin is always constant, which is represented by:

$$r_1 = \frac{c}{1-c}$$

That is why at each aging the ratio of old aged data to the new arrival's bin is not dependent on the lifetime of the stream. Each arriving packet has the same

weight, which is *(1-c)* to the new statistical data. The old aged data has also the same weight *c*.

This algorithm has an advantage, because data may still be up-to-date and can react quickly to the changes of the system. The ratio of old aged data to subsequent packets until the next aging is:

$$r_2 = \frac{c}{f(1-c)}$$

Following the same deviation from equations (4.5), (4.7) and (4.8), I obtain the following relation:

$$\frac{c_1}{1-c_1}\int_1^\infty H_{i-1}(x)dx = \frac{c_2}{1-c_2}\int_0^\infty H_{i-1}(x)dx$$

Thus, for the case *nd<t,* the same result in equation (4.9) could be received. A similar procedure can be used to derive an identical result for the case of $nd \geq t$. Hence I come to the same conclusion in the aging Algorithm 1 that $t_1 > t_2$ when $c_1 > c_2$ can be obtained.

Algorithm 3: Algorithm 1 calculates the aging coefficient without considering the ratio of old aged data to a new packet. Algorithm 2 scales down each bin by the factor described in the previous section (Equation (4.10)), which keeps the ratio r_1 constant. This ratio, in its turns, does not consider the aging frequency.

The third algorithm modifies Equation (4.10) in previous section as follows:

$$F = \frac{c * f}{(1-c) * \int_0^\infty H_{i-1}(x)dx} \tag{4.11}$$

In this equation, *f* is the aging frequency in *packet spacing*. Even if the aging frequency changes dynamically, the ratio of old aged data to the subsequent packets until next aging is constant, by scaling down the bin value by this equation (4.11). The aging could be done every *f* packets. The goal of Algorithm 3 is to maintain the same weight to the new statistical data for the sum of all subsequent packets. The ratio of old aged data to subsequent arrived packets is defined as:

$$r_2 = \frac{c}{f(1-c)}$$

However, the ratio of old aged data to a new packet depends on the aging frequency:

$$r_1 = \frac{c * f}{1 - c}$$

The following equation can be obtained:

$$\frac{f_1 * c1}{1 - c_1} \int_{t_1}^{\infty} H_{i-1}(x)dx = \frac{f_2 * c_2}{1 - c_2} \int_{t_2}^{\infty} H_{i-1}(x)dx$$

When f_i is fixed, $t_1 > t_2$, if $c_1 > c_2$. Similarly, $t_1 > t_2$ if $f_1 > f_2$ when c is fixed. This indicates that more frequent aging allows a smaller value of *ted*.

The three aging methods are different from each other only in the way of aging which happens with different frequencies. For example in case all of them do aging in 100 packets: in the arrival of 100th packet, they age the histogram by scaling down each bin with different factors. For the intervening packets between the two successive agings the corresponding bin in the histogram is increased by one for each packet. Algorithm 3 takes frequency (100 in this example) into account for the factor but Algorithms 1 and 2 do not. The next section discusses histogram bin width, which is strongly related to the accuracy of histogram data.

Bin Width of Histogram: As described in previous section, bin width is defined as the range of network delay grouped together to represent one pattern in histogram, and is connected directly to the accuracy of data stored. The value of bin width could be set to 10 milliseconds. Another value of bin width 205 milliseconds can be chosen to represent the network delay in the range of 200-209 milliseconds.

If the bin is narrow then the accuracy will be better for the case of wider bins. Anyway, when the width of the bins is narrower then the required number of bins will also be increased. In the previous section, the number of bins may range from 0 to ∞ depending on the network delay. Under congestion condition the network delays fluctuate widely thus it is necessary to reduce the number of bins. For reducing the number of bins and increasing the bin width each packet delay being stored to the histogram is calculated as follows:

$$b = \left\lfloor \frac{nd}{w} \right\rfloor + 1$$

In this equation *nd* is the network delay *w* is the bin width, and *b* is the corresponding bin in histogram. The function in the above question is floor function. This is the reason why network delay stay in *w* range, it is represented by only one bin. It is still necessary to recalculate the network delay, after storing the network delay of the packet to the corresponding bin:

$$nd = \left\lfloor b * w - \frac{w}{2} \right\rfloor \tag{4.12}$$

Equation (4.12) takes the mean value of this range to represent the network delay for this bin. One can also choose the maximum or minimum value that is *(b*w-1)* and *[(b-1)*w]*, respectively. Another alternative is to calculate the real mean value of delay in this range. This approach however requires additional overhead and memory to keep track and store the mean value of delay for each bin. This may not be suitable since an aim of wide bins is to reduce the memory needs.

Concord has a disadvantage, namely multiple successive packets can be discarded. These burst loss quantities were not significant as shown in the current results. In this section, Concord algorithm for synchronizing networked-multimedia streams is described. Concord is notable because it defines a solution for synchronization, which operates under the direct influence of application-supplied parameters for QoS control. In particular these parameters are used to facilitate a trade-off between the packet lateness rates, total end-to-end delay and skew. Thus an application can directly indicate an acceptable lost packet rate, rather than by having the synchronization mechanism operate by always trying to minimize losses due to lateness.

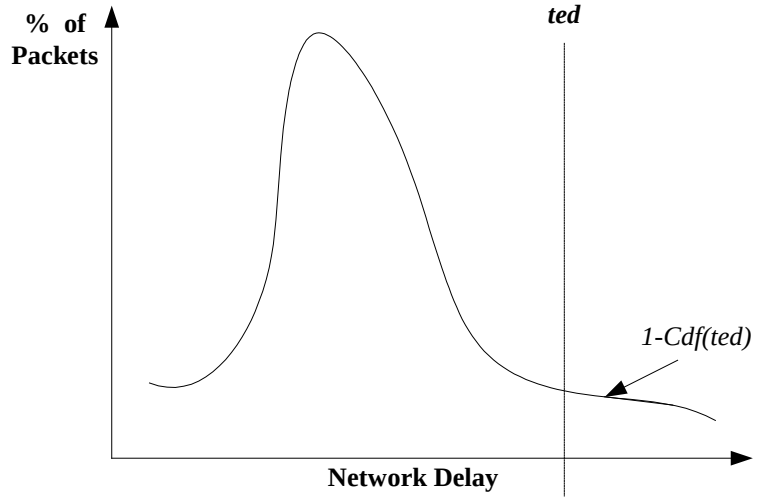


Figure 17 Concord algorithm

Chapter 5 Proportional Jitter Differentiation Model (PJDM)

The subject of this chapter is the queuing-jitter differentiations. After analysing the PDDM model in the previous chapter, I propose a new architecture, which is called *Proportional Jitter Differentiation Model* (PJDM), as a means for controllable and predictable jitter differentiation. I then discuss some properties of this model.

Subsequently, I focus on the methods of the performance evaluation. Two methods to analyse and compare different scheduling algorithms (as RJPS, PAJ, Adaptive-RJPS and Adaptive-PAJ that are described in the Chapter 6) in different models (PDDM and PJDM), two methods are proposed. The first one is for comparing the quality of my scheduling algorithms within only one hop and the other is for comparing the performance of PDDM and PJDM in a multi-hop network.

The content of this chapter is based on my work published in [Ngo3, Ngo4]

5.1 Proportional Jitter Differentiation Model (PJDM)

The PJDM aims to control the ratio of average jitter between classes based on the *Jitter Differentiation Parameters* (JDPs). Specifically, let $\bar{j}_i$ be the average queuing delay jitter, or simply the average jitter over time window *(t, t+τ)*. The PJDM model requires that the ratio of average jitter between two classes *i* and *j* is fixed to the inverse ratio of the corresponding JDPs:

$$\frac{\bar{j}_i(t,t+\tau)}{\bar{j}_j(t,t+\tau)} = \frac{\Delta_j}{\Delta_i} \tag{5.1}$$

where parameters $\{\Delta_i\}$ are the Jitter Differentiation Parameters (JDPs) and $\bar{j}_i(t,t+\tau)$ is the average queuing delay jitter of class *i'* s packets over time window *(t, t+τ)*. In this model, I say that class *i* is better than class *j* if $\Delta_i > \Delta_j$, or the Jitter Differentiation Parameter of one class can be called the *weight* of this class.

In detail, I can write: the Relative Proportional Differentiated Services Model for jitter is characterized by the following (*N-1*) equations:

$$\begin{aligned} \bar{j}_1(t,t+\tau)\Delta_1 &= \bar{j}_2(t,t+\tau)\Delta_2 \\ \bar{j}_2(t,t+\tau)\Delta_2 &= \bar{j}_3(t,t+\tau)\Delta_3 \\ &\cdots \\ \bar{j}_{N-1}(t,t+\tau)\Delta_{N-1} &= \bar{j}_N(t,t+\tau)\Delta_N \end{aligned} \tag{5.2}$$

Leaving the problem of delay and jitter measure for further studies, I assume that jitter of one packet in a queue is the difference of queuing delay of this packet and the preceding packet in this class (this definition is based on the standards of IP performance metrics working group of IETF)

$$j_i^{\,k} = \left| d_i^{\,k} - d_i^{\,k-1} \right| \tag{5.3}$$

Where $d_i^{\,k}$ is the queuing delay of packet number *k* of class *i* and $j_i^{\,k}$ is the jitter of this packet.

5.2 Some Properties

I consider a packet scheduler that services *N* queues, one for each class.

Property 1: For a *non-work-conserving* scheduler, it is possible to set the delay spacing between classes to arbitrary levels, so that the delays of each class stays proportional with each other. On the other hand - when the delay of each packet of each class is proportional to the other classes then its delay difference or jitter also becomes proportional. That means intuitively it is always possible to realise a non-work-conserving proportional jitter scheduler.

Definition 1: I say that a set of JDPs is *feasible* if there exists a work-conserving scheduler that can set the average jitter of each class as in equation (5.1). So the set of JDPs is feasible when the set of class jitters is feasible. Some examples of feasible sets of JDPS can be found in the Chapter 6 by simulations.

Definition 2: An *ideal proportional delay scheduler* is a scheduler, which produces delay proportionally for each packet of all the classes. See equation (3.2).

$$\frac{d_i^{\,k}}{d_j^{\,k}} = \frac{\delta_i}{\delta_j} \text{ for all pairs of } i \text{ and } j, \text{ and for all packets of classes } i \text{ and } j$$

Where $d_i^{\,k}$ is the queuing delay of packet number *k* of class *i* .

Property 2: An ideal proportional delay scheduler, which produces proportional delay for each packet is a proportional jitter scheduler with the Differentiation Parameters defined as $\Delta_i = \frac{1}{\delta_i}$.

For the packet number *k*, I have: $\frac{d_i^k}{d_j^k} = \frac{\delta_i}{\delta_j}$

And for the packet number *k+1*, I have

$$\frac{d_i^{k+1}}{d_j^{k+1}} = \frac{\delta_i}{\delta_j}$$

That means: $\frac{d_i^{k+1}}{d_j^{k+1}} = \frac{d_i^k}{d_j^k} = \frac{\delta_i}{\delta_j} = \frac{d_i^{k+1} - d_i^k}{d_j^{k+1} - d_j^k}$

From these equations, I have: $\frac{j_i^{k+1}}{j_j^{k+1}} = \frac{\left|d_i^k - d_i^{k+1}\right|}{\left|d_j^k - d_j^{k+1}\right|} = \frac{\delta_i}{\delta_j} = \frac{1/\delta_j}{1/\delta_i}$

Note that for the PJDM model Jitter Differentiation Parameters Δ_i are the importance or weight of class *i*, while for the PDDM model the importance or weight of each class is defined as $1/\delta_i$.

Property 3: Delay accumulates, jitter, however, does not.

It is easy to see that delay has accumulated property. That means that the delay of a class or flow through a network is the sum of the queuing delays at each router and the propagation time, which is considered small compared to the queuing delays. In addition, we assume that we cannot have any type of signaling in the proportional Diffserv model in order to carry control information for realising proportional delay at only egress router. Hence there is a need to implement proportional delay scheduling algorithms at *every router* in a network based on PDDM model for receiving proportional delay between different classes.

However, in the case of the existence of proportional delay scheduling schemes at every router, each packet transfers through network along different paths. Furthermore, each path can contain different hops numbers. In addition, it is important to recall that the local delays of each class at each hop is proportional with the delays of the other classes, and the network delay of each class is the sum

of local delays of all the hops that the path contains. When each path of each packet can contain different numbers of hops, the sums of local delays (or network delay) will not stay proportional with each other any more. That means the network delay of each class become unproportional with each other.

That is why the proportional property of queuing delay is just maintained for only one local hop, but the sum of queuing delay of one class does not stay proportional with other classes any more. In other words, such networks can not guarantee the proportional property of delay between different classes.

Jitter however is *not accumulated* and if there is proportional jitter scheduling mechanism at every router in the network, the jitter of each class after the egress router is not the sum of the jitter of each router produced within the network. Intuitively, the more the routers with proportional jitter scheduling algorithm are near the side of receiver end, the more strongly it will influence the playout delay adjustment algorithm implemented at receiver.

Finally, I believe that only work-conserving forwarding mechanism will be used in practice, because of the competition for the best possible service between service providers; this is mainly a non-technical issue however. Furthermore, for a non-work conserving scheduler, it is possible to set the jitter spacing between classes to arbitrary levels. For this reason, from now on I will only focus on the algorithms that belong to work-conserving type.

5.3 Methodology

Before going to the new scheduling algorithms in detail, it is necessary to establish the method for performance evaluation and comparison. At the beginning I focus on the method of evaluating the schedulers within only a single hop and then on the method of performance comparison of PJDM and PDDM models in multi-hop networks.

5.3.1 Method for Performance Evaluation of Schedulers within Single Hop

In this section, the methodology used to analyse and compare the behaviors of my new schedulers within only one hop is presented. It comprises performance criterion, network topology, simulation tool and traffic model.

5.3.1.1 Performance Criteria

All of my algorithms aim to produce proportional jitter between different classes. Hence in order to evaluate and compare these schemes, I need to use jitter ratio as the main performance criterion. A critical issue is not only to exam whether these schedulers can approximate *long-term jitter ratio* (calculated from the beginning of the simulation) but also *short-term jitter* (calculated over a moving window). Another important performance criterion is the *average delay* of each class produced by each algorithm.

5.3.1.2 Network Topology, Simulation tool and Traffic Model

In this section, I use only a simple network topology that is shown in the following Figure 18. The links are all 6 Mbps with a latency of 10 ms. The classes are numbered from 0 to 1 and 2. Each class contains some flows that are described more concretely in each simulation. Each flow is characterized by a sender *Si* and a receiver *Di*. My new algorithms are implemented at Router R1, while the FIFO scheme is used at the second router R2. I run and collect my simulation in 100 seconds.

In addition, I employ the Network Simulator ns-2 [NS2]. The traffic that is appropriate for voice sources can be well modeled by on/off sources. In the simulator, packets arrive in an on/off pattern. This type of traffic has different parameters, such as: rate, on-time and off-time. The talkspurt length (on period) and the gap length (off period) of speech signal is exponentially distributed. During the on-time, packets are generated with this predefined rate. The transport protocol used is UDP protocol. The FIFO scheduling is placed at the end The details of traffic model is described concretely in each simulation.

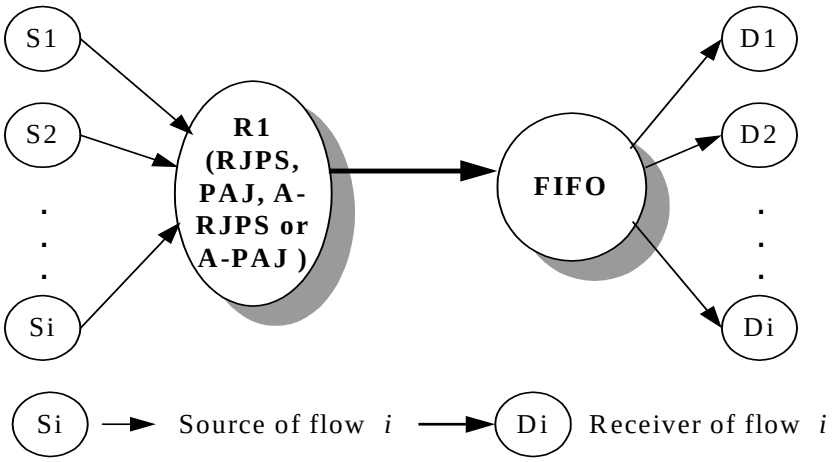


Figure 18 Network topology

In my simulation, I will set all the parameters so that the load in the links achieve high condition (approximately from 80 to 100%). This is based on the reason that the proportional jitter schedulers work only stable when there are enough packets in the queue. With light load, the packets should be scheduled immediately, the queuing delay stays small, jitter stays small, too, and no jitter differentiation is probably needed.

The average size of packet for UDP traffic is set based on the measurement in [Mc00].

5.3.2 Method for Performance Evaluation and Comparison of PJDM and PDDM

After analysing and comparing the quality of my new mechanisms within only one hop, it is also important to examine their performance in multi-hop networks based on PJDM and to compare with performance of networks based on PDDM. For this issue, I presents the necessary network topology, performance criteria in the following.

5.3.2.1 Network Topologies

It is necessary to explain the notion of *networks based on PJDM and PDDM models*. The networks, which uses *only* proportional delay scheduling scheme (as WTP) in their routers, are called based on PDDM. The networks using *only* proportional jitter scheduling algorithms (as RJPS, PAJ, Adaptive-RJPS and Adaptive-PAJ) are called based on PJDM.

Before going to simulative results, it is necessary to create some concrete conditions for the context of my experiments, especially the required elements for my system.

To simulate, there is a need to choose a precise system. It is noteworthy to say that my system contains the following elements (Figure 19):

- Sender
- Network based on the PJDM or PDDM model
- Playout buffer controlled by Concord, as playout buffer delay adjustment algorithm
- Receiver

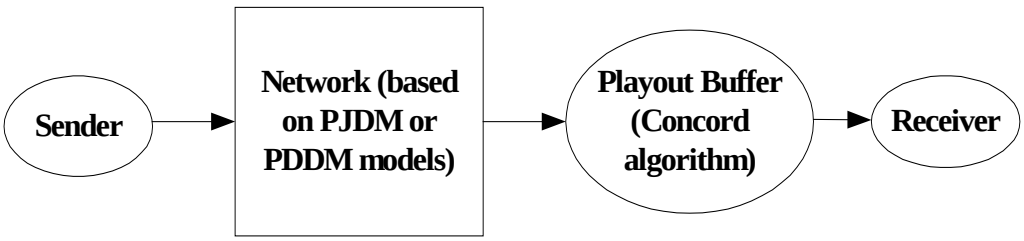


Figure 19 Network elements

In order to create appropriate topologies, it is now necessary to repeat some arguments described already in Chapter 4:

- Delay has accumulated property. That means the delay of a class or flow through a network is the sum of the queuing delays at each router and the propagation time, which is considered small compared to the queuing delays in case of congestion. Hence there is a need to implement proportional delay scheduling algorithms at *every router* in a PDDM model for receiving proportional delay between different classes. However, in the case of the existence of proportional delay scheduling schemes at every router, each packet can transfer through network along different paths that can contain different hops numbers. That is why the proportional property of queuing delay is just maintained for only one local hop, but the sum of queuing delay of one class does not stay proportional any more. In other words, such networks can not guarantee the proportional property of delay between different classes.

- Jitter, however, is *not accumulated* and if there is proportional jitter scheduling mechanism at every router in the network, the jitter of each class after the egress router is not the sum of the jitter of each router produced within the network. Furthermore, the more the routers with proportional jitter scheduling algorithm are near the side of receiver end, the more strongly it will influence the playout delay adjustment algorithm.

These two reasons lead me to an interesting idea: to implement proportional jitter scheduling schemes at *different positions* of networks *(core or egress routers)* in order to examine the influence of the proportional jitter schedulers of the PJDM model at the receiver end. In addition, it is also interesting to compare the performance of such topologies based on PJDM model with similar topologies based on PDDM model (WTP).

Independent of implementing the model of Relative Proportional Differentiated Services considering delay(PDDM) or jitter (PJDM), the objective of these

models is to improve end-to-end quality of service, or end-to-end delay. In other words, the model that produces smaller end-to-end delay is better.

In the rest of my thesis, I am just interested in network approaches, which contain either proportional jitter or delay schedulers at *all the router* or *only at the egress router*. These reasons lead us to create two approaches for each PJDM or PDDM models, as shown in Figure 20, 21, 22, 23.

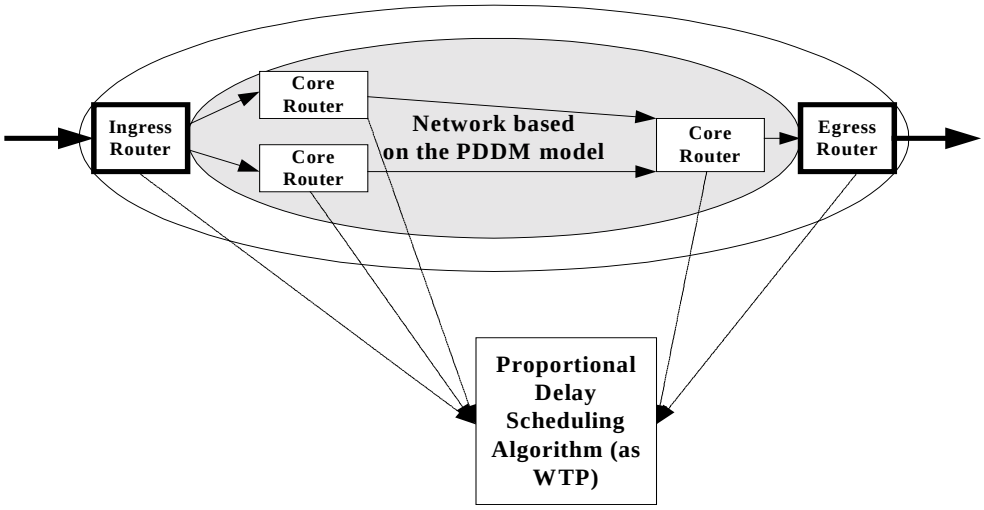


Figure 20 Network based on the PDDM model, type 1

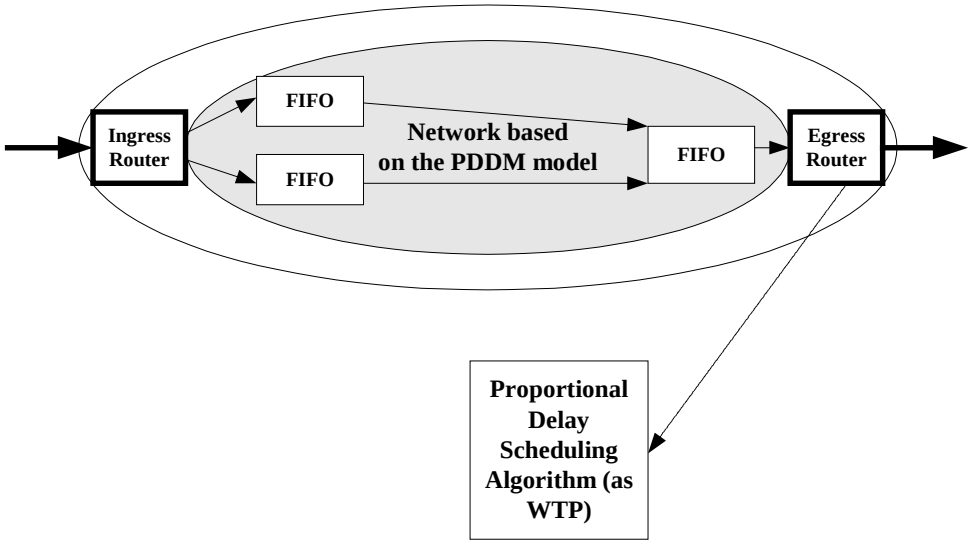


Figure 21 Network based on the PDDM model, type 2

With the PDDM model, the first type contains proportional delay schedulers at every router of its network (Figure 20). The second type (Figure 21) contains only proportional router scheduling scheme at the egress router.

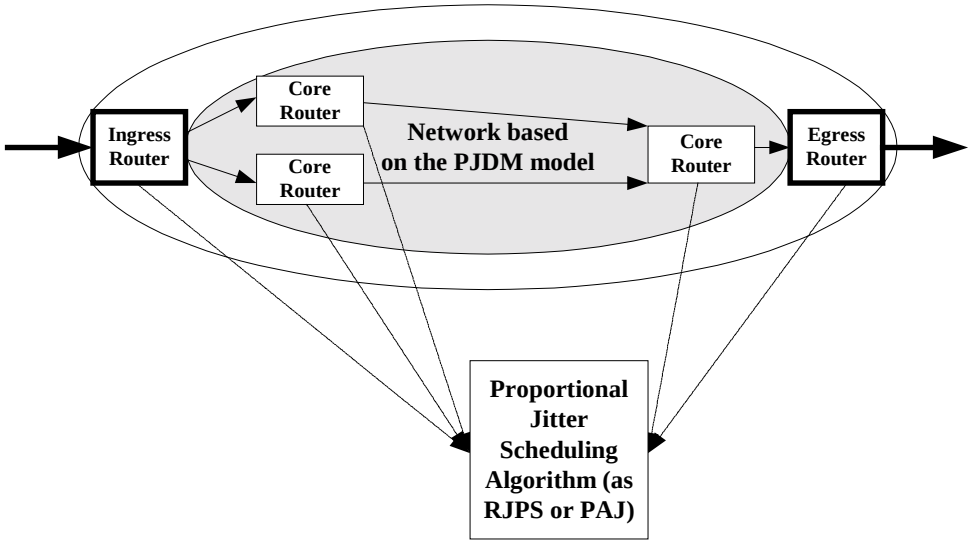


Figure 22 Network based on the PJDM model, type 1

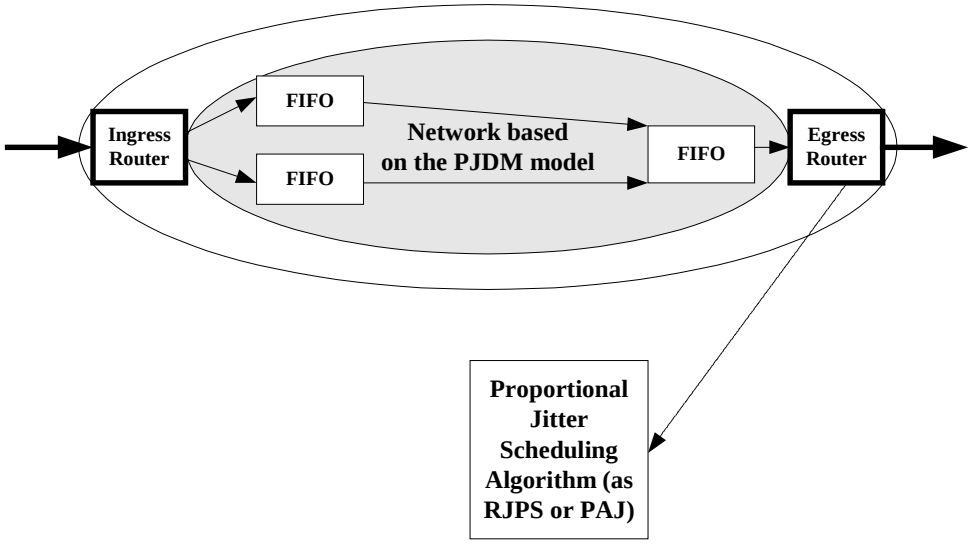


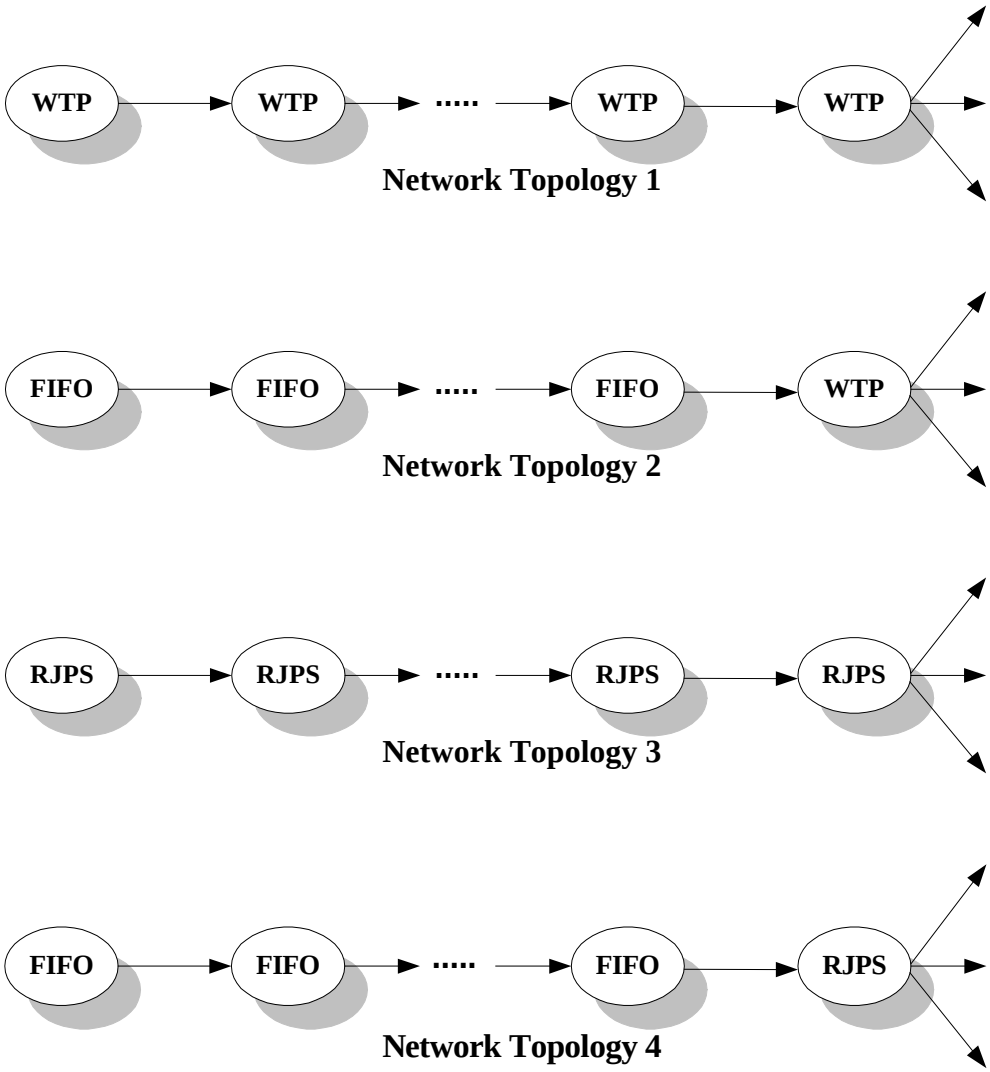
Figure 23 Network based on the PJDM model, type 2

Similarly to the PDDM model, there are also two types for the PJDM. The first type contains proportional scheduling algorithms at every router of its network

(Figure 22). The second type contains only proportional scheduling algorithm at the egress router (Figure 23).

Starting from this network types, I receive a total of 10 Network Topologies (Figure 24) by replacing proportional delay mechanisms by WTP and proportional jitter schemes by RJPS, PAJ, Adaptive-RJPS or Adaptive-PAJ. The two first ones are based on the PDDM model, and contain WTP at every router or only at the egress router. The others are based on the PJDM model, and contain RJPS, PAJ, Adaptive-RJPS or Adaptive-PAJ at every router of the network or only at the egress router.

These configurations are illustrated in the Figure 24:



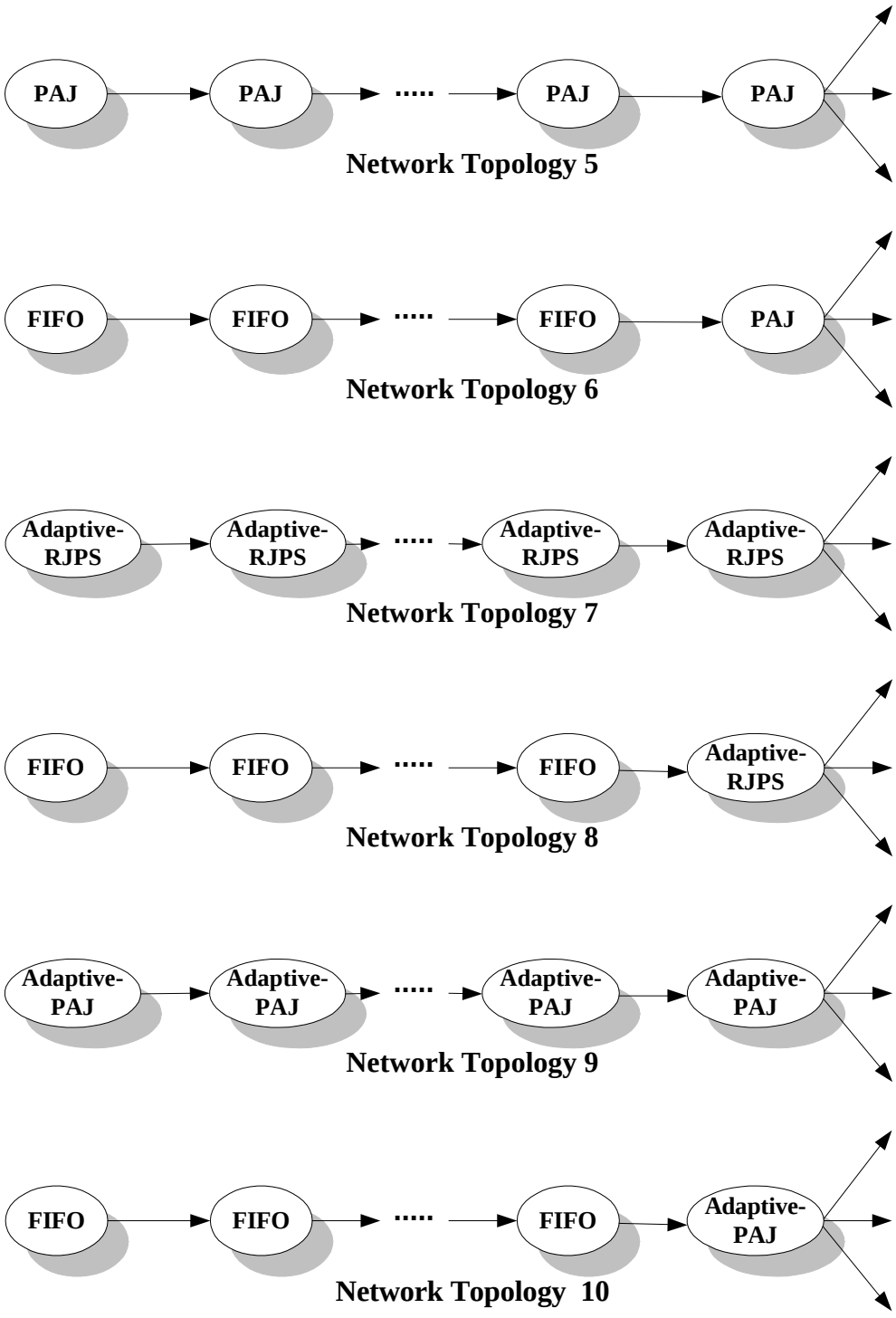


Figure 24 Different network topologies en details

There are 10 network configurations, which are illustrated in Figure 24. The two first ones are network topologies based on the PDDM model, because they use proportional delay scheduling algorithm WTP. In contrast, the other topologies are based on the PJDM model, because of the use proportional jitter scheduling algorithms as RJPS, PAJ, Adaptive-RJPS and Adaptive-PAJ in its routers.

On the other side, these network topologies differ from the use of proportional delay or proportional jitter scheduling algorithms at all routers or only at the egress network. Network Topologies numbered 1, 3, 5, 7 or 9 use this scheduling algorithm at all the positions of the networks, but Network Topologies numbered 2, 4, 6, 8 or 10 use this scheduling algorithms only at the egress routers.

5.3.2.2 Performance Criteria

In a multi-hop network based on PJDM model, it is important to examine the *jitter ratio* between different classes. In addition, independent of implementing the model of PJDM or PDDM in the networks, the application users want to receive better end-to-end quality of service. The most important end-to-end quality of service is *end-to-end delay*. Hence I intend to examine also end-to-end delay as performance metric in order to compare the quality of different networks based on PDDM or PJDM. End-to-end delay can be considered as the sum of network delay and playout buffer delay

Suppose that the routers in my experiments use proportional delay scheduling scheme (as WTP) or proportional jitter scheduling mechanisms (as RJPS, PAJ, Adaptive-RJPS or Adaptive-PAJ) in order to schedule packets proportionally between different classes. There are a total of *N* classes. Each class *i* (*i* is from *1* to *N*) has a weight Δ_i. For the PDDM model, this weight is the inverse ratio of the Delay Differentiation Parameter, while for the PJDM model this weight relates directly to the Jitter Differentiation Parameter. Through my network, each packet numbered *n* of class *i* suffers a network delay $D^{network,i,n}$.

At receiver, I use Concord mechanism as playout adaptation in order to smooth the jitter produced by the network. The packet numbered *n* in class *i* is buffered, and its playout buffer delay is called $D^{Playoutbuffer,i,n}$. $D^{Playoutbuffer,i,n}$ depends on delay variation $J^{network,i}(t)$, loss rate L_i and window *w* (The PDD distribution defined by Concord algorithm is calculated over this window).

$$D^{Playoutbuffer,i,n} = f_{PDD}^{Playoutbuffer}(w, J^{network,i}(t), L_i)$$

The end-to-end delay of this packet is calculated as the sum of network delay and playout buffer delay:

$$TED = D_i^{endtoend,i,n} =$$
$$D^{network,i,n} + D^{Playoutbuffer,i,n} = D^{network,i,n} + f_{PDD}^{Playoutbuffer}(w, J^{network,i}(t), L_i)$$

Finally, in order to estimate the average end-to-end delay of one class, I average the end-to-end delay of *n* packets belong to this class:

$$\overline{D}^{endtoend,i} = \frac{\sum_n D^{endtoend,i,n}}{n}$$

Let me suppose that I have *N* classes. Each class has a weight of Δ_i, and network topology numbered *j* produces $(\overline{D}_{NTj}^{endtoend,0}, \overline{D}_{NTj}^{endtoend,1}, ..., \overline{D}_{NTj}^{endtoend,N-1})$ end-to-end delays for *N* classes when the loss rate is *L%*. Under the same conditions (the same load, the same loss rate, the same load distribution between different classes), Network configuration numbered *k* produces $(\overline{D}_{NTk}^{endtoend,0}, \overline{D}_{NTk}^{endtoend,1}, ..., \overline{D}_{NTk}^{endtoend,N-1})$ end-to-end delay for these *N* classes.

The problem of comparing two network topologies numbered *k* or *j*, in terms of end-to-end delay, is now explained as the comparison of two sets of end-to-end delays of *N* classes, produced by these two topologies $(\overline{D}_{NTj}^{endtoend,0}, \overline{D}_{NTj}^{endtoend,1}, ..., \overline{D}_{NTj}^{endtoend,N-1})$ and $(\overline{D}_{NTk}^{endtoend,0}, \overline{D}_{NTk}^{endtoend,1}, ..., \overline{D}_{NTk}^{endtoend,N-1})$.

I believe that the question of comparing these two sets of end-to-end delays of any two network configurations is complicated because it depends on cost structures and on weight of each class. One possibility is to introduce a cost for each byte in one class, and to compare the gain I receive. An alternative is to introduce profit for decreasing the delay in the higher class and a loss for decreasing the delay in the lower class.

In order to resolve this problem, I propose a simple criterion, which is based on the set of end-to-end delays of *N* classes and on the weights of each class. It is described as follows:

Let me suppose that each class has its own weight that describes its importance level. For example, if Class 1 has a weight of *1* and Class 2 has a weight of *2*, that means Class 2 is as twice important as Class 1. This observation leads me to the normalization of end-to-end delay of each class with its weight. In other words,

the normalized end-to-end delay of one class is the product of its end-to-end delay and its weight. Furthermore, the topology, which produces a smaller sum of all these normalized end-to-end delays than the other topologies, has better end-to-end quality of service, or better performance.

Briefly, in order to compare the performance of different network topologies I use the sum of all normalized end-to-end delays produced by this topology as comparison criterion. This performance criterion (called normalized end-to-end delay) of one topology is formally described by the following equation:

$$P_k = \sum_{i=1}^{N} \frac{\overline{D}_{NTk}^{endtoend,i} * \Delta_i}{\sum_{i=1}^{N} \Delta_i}$$

where P_k is the normalized end-to-end delay of Network Topology numbered *k*. And I say that a *Network Topology is better, whose normalized end-to-end delay is smaller.*

I decide to choose normalized end-to-end delay as the first performance criterion for comparison because end-to-end delay is considered more important than jitter, specially for interactive applications as voice. The reason of this assumption is that small delay with high long-term jitter is better than high but stable delay, because intractive applications such as voice is only sensitive with short-term jitter and can tolerate long-term jitter.

Chapter 6 New Scheduling Algorithms and Performance Evaluation of PJDM and PDDM models

Starting from the PJDM model, I design some proportional jitter differentiation scheduling mechanisms.

Originally, there are two schemes, which are proposed in this work: Relative Jitter Packet Scheduling (RJPS) and Proportional Average Jitter algorithms (PAJ).

Subsequently, I describe new adaptive Jitter Differentiation Parameters that are changeable in these schemes. The performance of RJPS and PAJ using these variable parameters are also compared with its original mechanisms, based on the methodology described in Chapter 5. The two new mechanisms are called Adaptive-RJPS and Adaptive-PAJ.

I then focus on the evaluation and comparison of performance of PJDM and PDDM model. Finally, I propose a combination of PJDM and PDDM models in order to overcome their disadvantages.

The content of this chapter is based on my work published in [Ngo1, Ngo2, Ngo3].

6.1 Relative Jitter Packet Scheduling Algorithm (RJPS)

In this section, RJPS algorithm with its behaviors under different contexts is described.

6.1.1 Algorithm Description

Suppose that each router has a prespecified number of jitter classes *N*. Each jitter class is served by a single first-in-first-out (FIFO) packet queue (Figure 25). Packets of a flow belonging to a jitter class *i* are queued in the corresponding queue in each router that the flow passes through. All flows with the same jitter class specification share the same FIFO queue at the router. The goal of my scheduling algorithm is to serve the packets such that the short-term average jitter and long-term average jitter experienced by packets in jitter class satisfy equation (5.1) for all pairs of *i* and *j*. In other words, queues of different classes are served such that the average jitter experienced by packets in a class is inversely proportional to the jitter weight of the class.

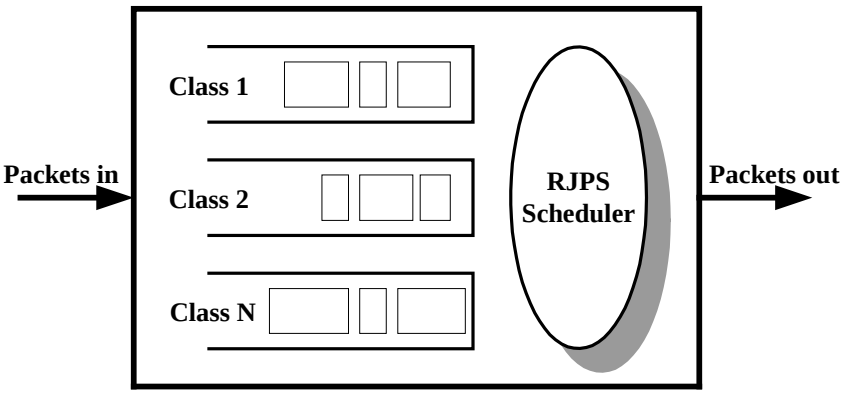


Figure 25 RJPS scheduler

This above observation is illustrated by the equation: $\left|\bar{j}_i(t)\Delta_i - \bar{j}_j(t)\Delta_j\right| \to 0$. A simple heuristic to achieve this is to serve the class with the maximum value of $\bar{j}_i(t)\Delta_i$ at any time *t*. Because the Jitter Differentiation Parameters are all known before, now the question is: how to evaluate the average jitter of each class? In the following, I will present the way that average jitter of each class is evaluated.

Note that for the class *i*, at any time *t*, there are packets that have been served and there are packets that are still in the queue. For the packets that have been served, their queueing delays, and thus, their jitters are determined. In addition, the queueing delays, and the jitters of packets that are still in the queue are unknown. In order to evaluate average jitter of each class, I will try to evaluate the jitters of the packets that are still in the queue.

Assuming that: in class *i* at time *t*, for each packet number *k* I know the arrival time t_i^k, the starting time of transmission T_i^k and the transmission time TS_i^k.

- For all packets that have already been served, I call $j_i^*(t)$ the aggregate jitter experienced by all packets that have been served in the queue *i* at time *t*. This value is already determined, because all packets were served, and thus the queuing delays of these packets are already determined, too.

$$j_i^*(t) = \sum_k jitter - of - each - packet = \sum_{k=1}^{s_i(t)} \left| d_i^k - d_i^{k-1} \right| = \sum_{k=1}^{s_i(t)} \left| (T_i^k - t_i^k) - (T_i^{k-1} - t_i^{k-1}) \right|$$

Where d_i^k is the queuing delay of packet number *k* in class *i*, $s_i(t)$ is the number of packets served from jitter class *i* till time *t*.

- For all packets that are now queued in, I call $j_i^{\min}(t)$ minimum jitter for all packets that have already arrived. Assuming that no other packet arrives for this class i in the future, this value can be calculated as:

$$j_i^{\min}(t) = \sum_{k=s_i(t)+1}^{s_i(t)+q_i(t)} \left| TS_i^k - (t_i^k - t_i^{k-1}) \right| \qquad (6.1)$$

Where TS_i^k is the transmission time of packet number *k* in the queue *i*, $q_i(t)$ is the number of packets that are now queued in this class. This formula is illustrated in Figure 26.

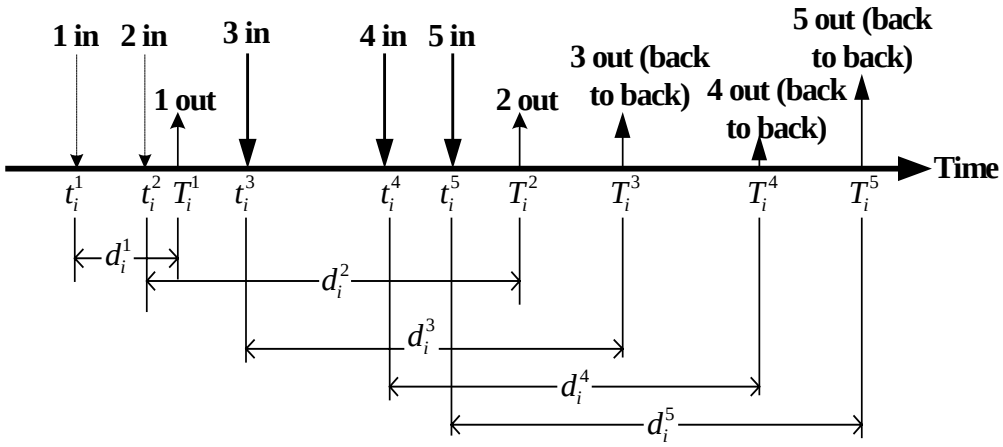


Figure 26 Packets in the class

Following this example, the packets number 1 and 2 of this class have already been served and their queuing delays are determined. In this class there are still packets number 3, 4, 5 that should be scheduled (I have here $s_i(t) = 2$ and $q_i(t) = 3$). Recall that jitter of one packet in a queue is the difference of queuing delay of this packet and the previous packet in this class:

$$j_i^k = \left| d_i^k - d_i^{k-1} \right|$$

Because of a work-conserving scheduler, packets numbered 3, 4, 5 achieve minimum jitter $j_i^{\min}(t)$ when all theses packets 3, 4, 5 are transmitted back-to-back in order to assure minimum assumed queering delays. That means:

$$j_i^{\min}(t) = \sum_{k=3}^{5} \left| d_i^k - d_i^{k-1} \right|$$

If all packets 3, 4, 5 are transmitted back-to-back.

In Figure 26, I can rewrite:

$$\left|d_i^k - d_i^{k-1}\right| = \left|(T_i^k - t_i^k) - (T_i^{k-1} - t_i^{k-1})\right| = \left|TS_i^k - (t_i^k - t_i^{k-1})\right|$$

That means I can have:

$$j_i^{\min}(t) = \sum_{k=s_i(t)+1}^{s_i(t)+q_i(t)} \left|TS_i^k - (t_i^k - t_i^{k-1})\right|$$

I can evaluate the value of average jitter $\bar{j}_i(t)$ for all the packets in class *i* at time *t* as:

$$\bar{j}_i(t) \geq \frac{j_i^*(t) + j_i^{\min}(t)}{s_i(t) + q_i(t)}$$

And $\bar{j}_i^{\min}(t) = \dfrac{j_i^*(t) + j_i^{\min}(t)}{s_i(t) + q_i(t)}$

In my scheduler, I set the priority of the Head of Line packet in class *i* at time *t* to:

$$p_i(t) = \bar{j}_i^{\min}(t)\Delta_i \qquad (6.2)$$

Recall that the goal of my scheduler is to serve the packets such that the short-term average jitter and long-term average jitter experienced by packets in a jitter class satisfy Equation (5.1) for all pairs of *i* and *j*. A simple heuristic to achieve this equation is to serve the jitter class with the maximum value of $p_i(t)$ at any time *t*. In other words, the router selects the HOL packet of class *i* for which its priority is a maximum among all backlogged classes.

Using this priority structure, after a time *t*, every class's jitter converges to the value:

$$\bar{j}_i\Delta_1 = \bar{j}_2\Delta_2 = ... = \bar{j}_N\Delta_N$$

That means the average jitter for each class is proportional to its weights, satisfying equation (5.1).

From equation (6.2) I have:

$$p_i(t) = \bar{j}_i^{\min}(t)\Delta_i = \frac{j_i^*(t) + j_i^{\min}(t)}{s_i(t) + q_i(t)}\Delta_i = \frac{j_i^*(t) + \sum_{k=s_i(t)+1}^{s_i(t)+q_i(t)} \left| TS_i^k - (t_i^k - t_i^{k-1}) \right|}{s_i(t) + q_i(t)} \qquad (6.3)$$

It is necessary to note that I should maintain a window of packets in order to address the inaccuracies caused by non-backlogged queues and accumulated history because when I do not maintain this window, as the number of packets that are served increases, the current queue sizes start to have minimal impact on the service order. Note that I will also use this window for the purpose of evaluation of average short-term jitter ratio. In conclusion, the following remarks are implied from the above equation:

- The change in size of packets makes its transmission time TS_i^k, and hence, the priority of HOL packet calculated in (6.3.), vary quickly, too. That means the packet sizes is an influence factor on the algorithm.

- Link utilization plays an important role for the behavior of RJPS scheduler. If there are not enough packets in backlogged classes, the average jitter calculated in (6.3.) is based only on the history situation of the system and cannot response quickly to the changes of current load conditions. In other words, if the link utilization is high, it is easier to achieve proportional jitter between different classes than lower link utilization.

- The quality of RJPS scheduler decreases when the window's size decreases, because the width of window determines how closely the average jitter value follows the short-term variation of jitter. The increase of window size improves the quality of RJPS in expense of implementation cost.

- The spaces between the weights of different classes have an influence on the calculation of this priority. That means when these spaces are high then it is difficult to realise the above equation, or to realise the PJDM model.

Figure 27 describes the necessary operations of RJPS algorithm :

Receive (packet)

1 $q_i \leftarrow q_i + 1$;

2 **queue** in corresponding jitter class;

Select_packet_to_transmit()

3 $k = \arg\max_i \{\bar{j}_i^{\min}(t)\Delta_i\}$;

4 transmit from jitter class *k*;

5 $q_i(t) \leftarrow q_i(t) - 1$;

6 $s_i(t) \leftarrow s_i(t) + 1$;

7 **foreach** jitter class *i*;

8 $j_i^*(t) \leftarrow j_i^*(t) + \left|(T_i^{k+1} - t_i^{k+1}) - (T_i^k - t_i^k)\right|$;

9 $j_i^{\min}(t) \leftarrow j_i^{\min}(t) + \left|TS_i^{k+1} - (t_i^{k+1} - t_i^k)\right|$;

Figure 27 RJPS algorithm

6.1.2 Simulations

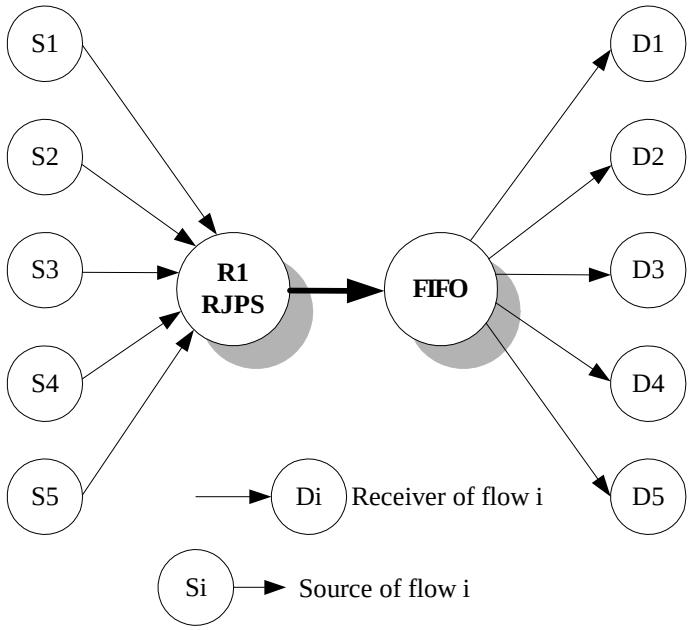


Figure 28 Network topology

The simulation model is as follows. The topology used is shown in Figure 28. The links are 6 Mbps with a latency of 10ms. There are a total of 3 classes 0, 1 and 2. Flow 1 from S1 to D1 (1.5 Mbps) and Flow 2 from S2 to D2 (2 Mbps) belong to

class 0, while Flow 3 from S3 to D3 (0.5 Mbps) and Flow 4 from S4 to D4 (0.5 Mbps) belong to class 1 and Flow 5 from S5 to D5 (2 Mbps) belong to class 2.

The objective of this simulation study is to evaluate the behavior of RJPS scheduler in terms of long-term and short-term jitter ratio.

6.1.2.1 Behavior of RJPS with Constant Size of Packets and Heavy Load

In this simulation, the jitter differentiation parameters of classes 0, 1, 2 are respectively 1; 1,5 and 3. The predefined jitter ratio between class 0 and class 2 is 3 and between class 1 and 2 is 2. The window's size is set to 200 packets and all packets have a size of 160 bytes (this moving window is used in order to evaluate the short term jitter). I intended to test the performance of RJPS in terms of average long-term jitter and average short-term jitter. The link utilization between the RJPS router and the FIFO router in this simulation is set to 100%.

Average long-term jitter: shows that average long-term jitter ratio for 3 classes achieve the pre-defined ratio 3 and 2. This ratio is achieved after a time of fluctuation of 10 seconds. See Figure 29.

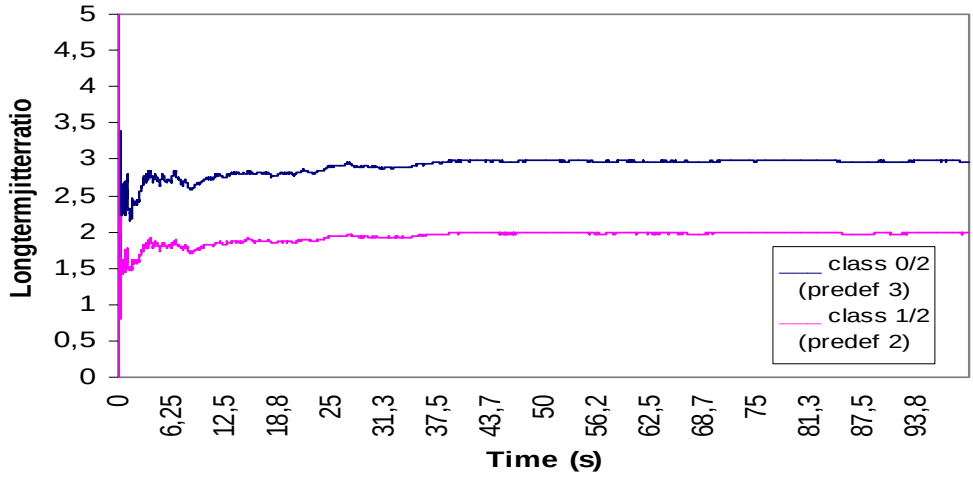


Figure 29 Variation of long-term jitter ratio with constant packet's size and heavy load

Average short-term jitter: Figure 30 shows that the short-term jitter ratio fluctuates strongly and can reach up to 45, although the predefined ratio is only 3 and 2. I can say that my scheduler achieves poor quality with short-term jitter. One reason is that I evaluate short-term jitter over my window size of 200 packets only.

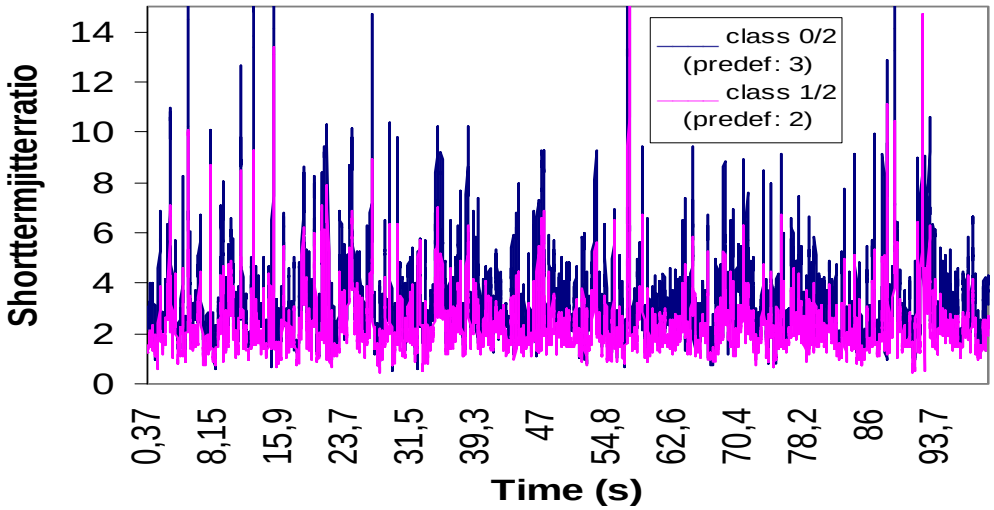


Figure 30 Variation of short-term jitter ratio with constant packet's size and heavy load

Average delay: Clients are satisfied only if they received both better jitter and better delay. Hence, it is very important to examine the behavior of RJPS in term of delay because if my algorithm works well for proportional jitter, but a class with higher weight would receive higher delay, it is difficult to conclude that the class with higher weight is better than the class with lower weight.

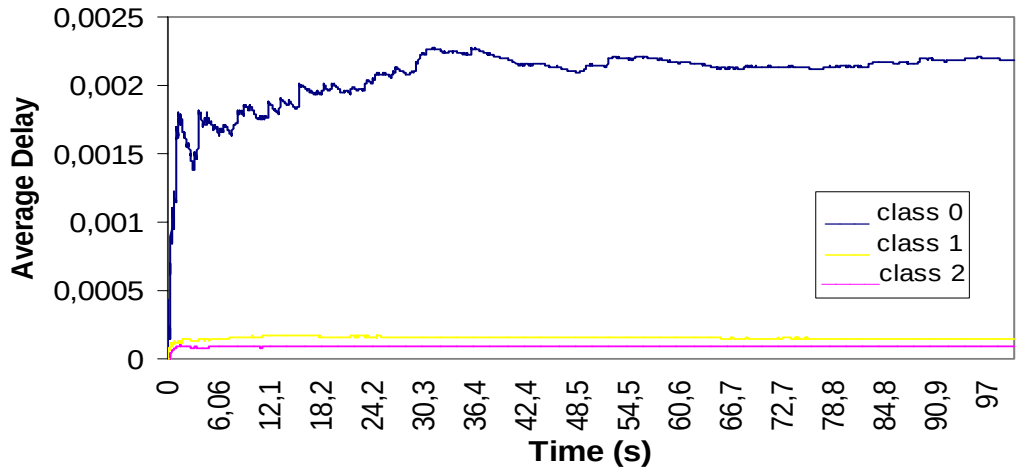


Figure 31 Average delay of different classes

In this simulation, I evaluate average long-term delay for each class. Figure 31 shows that delay of a class with higher weight is smaller than delay of a class with lower weights.

6.1.2.2 Behavior of RJPS Scheduler with Variation of Packet's Size

Change of Packet Size	Long-term jitter ratio						Short-term jitter ratio					
	Class 2/ Class 0 (predefined 0.5)			Class 1/ Class 0 (predefined 0.666)			Class 2/ Class 0 (predefined 0.5)			Class 1/Class 0 (predefined 0.666)		
	Ave.	Max	Min	Ave.	Max	Min	Ave.	Max	Min	Ave.	Max	Min
72 to 256 bytes	0.4996	0.5021	0.498	0.6661	0.676	0.6579	0.5331	2.5249	0.245	0.65	5.4588	0.1879
72 to 512 bytes	0.4839	0.7731	0.4629	0.6669	1.3032	0.6543	0.7384	6.979	0.228	0.82	1026.4	0.3557
72 to 1024 bytes	0.4779	0.5013	0.4525	0.6652	0.706	0.6302	0.6975	23.769	0.1689	0.77	49.543	0.1432

Table 4 Performance of the RJPS algorithm

In this simulation the jitter differentiation parameters of classes 0, 1, 2 are respectively 1; 1,5; 2. The predefined ratio between class 2 and 0 is 0.5 and between class 1 and 0 is 0.667. Window size is 200 packets.

The packet size plays an important role for the performance of my scheduler, as changing packet size makes the time of transmission of packets varying widely, and hence it makes the deviation of $j_i^{\min}(t) = \sum_k \left| TS_i^k - (t_i^k - t_i^{k-1}) \right|$ between different classes larger. My proportional jitter ratio is difficult to achieve.

My traffic is based on the study of packet size. With UDP traffic the size of packets varies around 157 bytes (See [Mc00]). When the size of packets varies, the ratio of short-term jitter varies widely. The results in Table 4 show that ratio of jitter when packets size varies from 72 to 256 bytes, 72 to 516 bytes and 72 to 1024 bytes.

Figure 32 and Figure 33 compare the average long-term jitter ratio and short-term jitter ratio between different classes. Results derived from theses experiments showed that in most cases, the performance of long-term jitter ratio of RJPS stays nearly constant. But when the packets with variable sizes come to my router, the ratio of short-term jitter fluctuates very strongly. The worst case is when packet size varies from 72 to 1024 bytes and the best case appears when packet size varies from 72 to 256 bytes. It is noteworthy that the short-term jitter ratio of my scheduler depends strongly on the variation of packet size, for example, this ratio can fluctuate between 0.1432 and 49.5437 where the predefined ratio is only 0.66 when the packet size is between 72 and 1024 bytes.

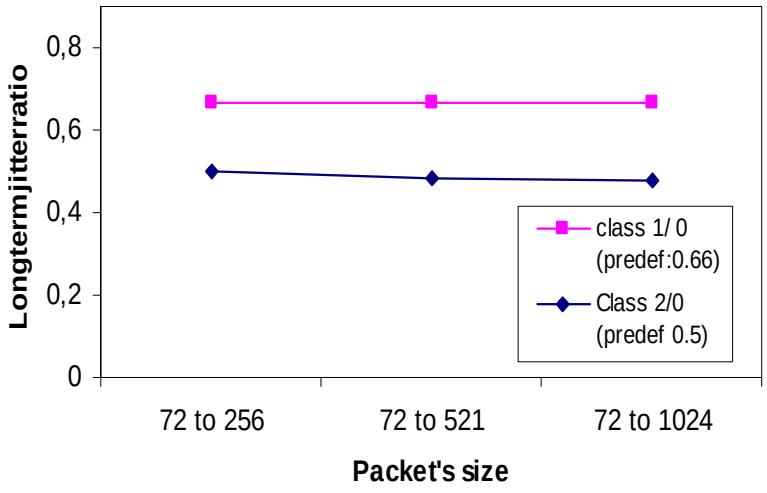


Figure 32 Long-term jitter ratio with variable packet's size

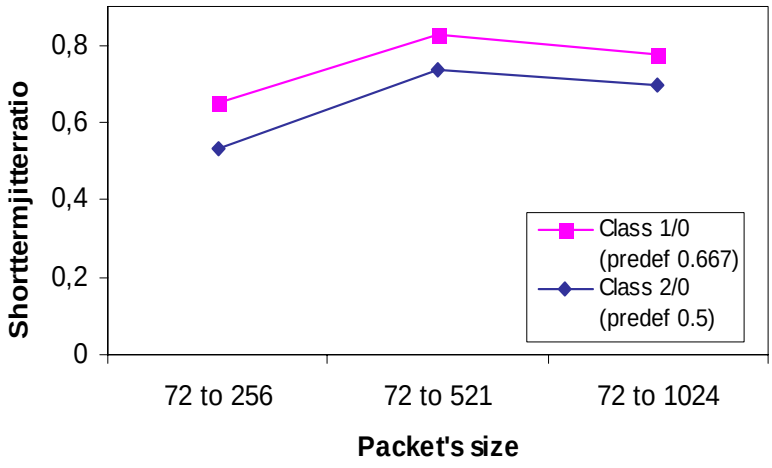


Figure 33 Short-term jitter ratio with variable packet's size

6.1.2.3 Behavior of RJPS with Variation of Link Utilization

Link utilization	Long-term jitter ratio						Short-term jitter ratio					
	Class 2/ Class 0 (predefined 0.5)			Class 1/ Class 0 (predefined 0.666)			Class 2/ Class 0 (predefined 0.5)			Class 1/Class 0 (predefined 0.666)		
	Ave.	Max	Min	Ave.	Max	Min	Ave.	Max	Min	Ave.	Max	Min
60 %	0.6047	0.6058	0.5834	0.9094	1.034	0.6446	0.6249	1.1022	0.407	0.837	1.7735	0.449
70 %	0.5537	0.5608	0.5521	0.7386	0.7476	0.7277	0.5609	1.0633	0.286	0.744	1.3656	0.422
80 %	0.5039	0.5131	0.498	0.6725	0.6818	0.6709	0.5157	1.2803	0.3261	0.6898	1.679	0.4276
90 %	0.5009	0.51	0.499	0.668	0.67	0.667	0.5104	1.4082	0.3188	0.68	2.33	0.22
100 %	0.5047	0.5067	0.5043	0.667	0.669	0.666	0.5008	1.4839	0.31	0.667	2.17	0.3689

Table 5 Performance of the RJPS algorithm

I will now investigate the jitter ratio between different classes where the total traffic varies from moderate load to heavy load. In this simulation the jitter differentiation parameters of classes 0, 1, 2 are respectively 1; 1,5; 2. The predefined ratio between class 2 and 0 is 0.5, between class 1 and 0 is 0.667. The results in Table 5 show the performance of jitter ratio in this context. It is necessary to note that my scheduler deviates remarkably from the desired values at moderate loads, while the proportional jitter differentiation can be maintained more accurately in heavy load situations. For example, with load of 60%, the average ratio of class 2/0 is 0.6249 (predefined 0.5), while with load of 100%, this ratio is 0.5008. Figure 34 and Figure 35 plot the variation of jitter ratio with variation of link utilization.

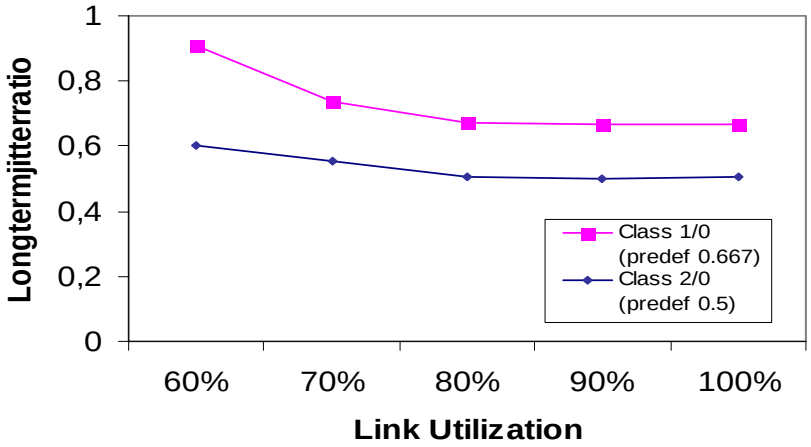


Figure 34 Long-term jitter ratio with variation of link utilization

My scheduler works stable when there are enough packets in the queue. With light load, the packets should be scheduled immediately, the queuing delay stays small, jitter stays small, too, and no jitter differentiation is probably needed. That is why the proportional jitter model works well only under heavy load condition.

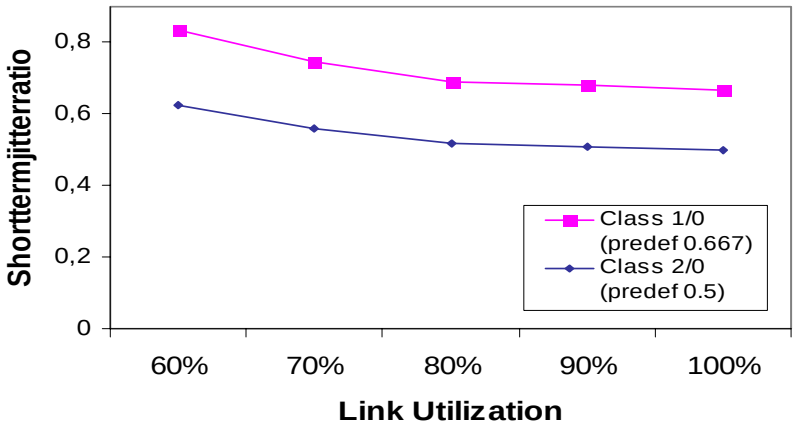


Figure 35 Short-term jitter ratio between different classes with variation of link utilization

6.1.2.4 Behavior of RJPS with Variation of Window's Size

Window Size	Long-term jitter ratio						Short-term jitter ratio					
	Class 2/ Class 0 (predefined 0.3334)			Class 1/ Class 0 (predefined 0.5)			Class 2/ Class 0 (predefined 0.3334)			Class 1/ Class 0 (predefined 0.5)		
	Ave.	Max	Min	Ave.	Max	Min	Ave.	Max	Min	Ave.	Max	Min
100 packs	0.33402	0.336	0.332	0.50075	0.501	0.4945	0.3385	0.92	0.038	0.503	5.25	0.053
200 packs	0.33302	0.335	0.332	0.49873	0.499	0.496	0.3246	0.6	0.058	0.491	0.86	0.14
300 packs	0.33289	0.334	0.332	0.50054	0.501	0.49	0.328	0.53	0.109	0.4935	0.79	0.23

Table 6 Performance of the RJPS algorithm

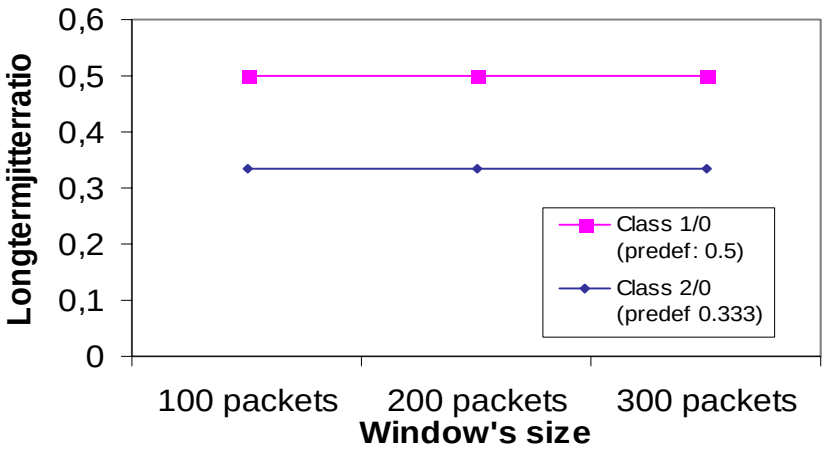


Figure 36 Long-term jitter ratio with variable window's size

I will now examine the performance of my scheduler when the window size varies. In this simulation the jitter differentiation parameters of classes 0, 1, 2 are respectively 1, 2, 3. The predefined ratio between class 2 and 0 is 0.333, between class 1 and 0 is 0.5, and the size of packet is 160 bytes.

The choice of window size has an important effect on the stability of my algorithm because it makes the average jitter vary $j_i^*(t)$. It is straightforward to see that the accuracy of my algorithm increases with the size of window size chosen. But when the window size is high then the computation cost grows rapidly, too. In the previous simulation I present the result of RJPS scheduler with the window of 200 packets. In this section I have evaluated the behavior of RJPS in the context of variable window size from 100 packets to 300 packets. The performance is shown in the Table 6.

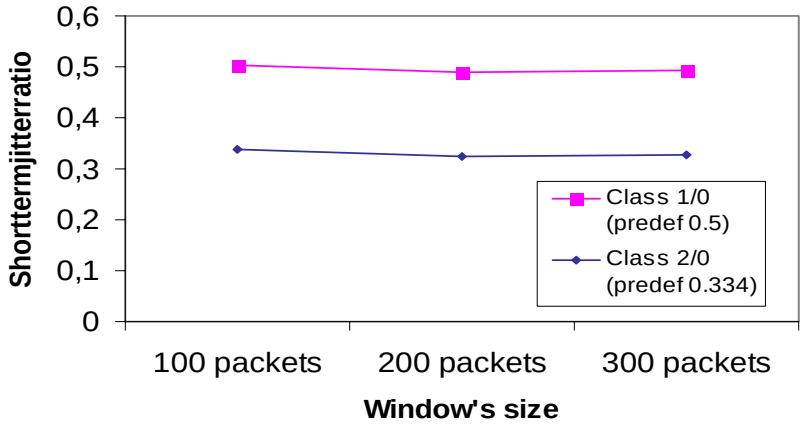


Figure 37 Short-term jitter ratio with variable windows size

As I can see, the result of long-term jitter ratio is similar but the maximum of short time jitter ratio of class 1/class 0 when the window is 100 packets can increase to 5.247. With a window size of 200 packets this maximum value is only 0.8565 and with a window size of 300 packets this value is 0.7840. My result shows that performance of RJPS, especially short-term jitter ratio, increases with the packet size while long-term jitter ratio is not influenced by this window size. Figure 36 and Figure 37 plot the variation of long-term and short-term jitter ratio between different classes.

6.2 Proportional Average Jitter Scheduling Algorithm (PAJ)

6.2.1 Algorithm Description

A way to interpret the Proportional Jitter Differentiation model is that the priority $p_i(t)$ of class i (defined as $p_i(t) = \bar{j}_i(t)\Delta_i$) must be equal in all classes, i.e.

$$p_i(t) = \bar{j}_i(t)\Delta_i = \bar{j}_k(t)\Delta_k = p_k(t) \tag{6.4}$$

Similar to RJPS, this new scheduler aims to equalize the priority among all classes. I refer to this algorithm as Proportional Average Jitter (PAJ) scheduling algorithm.

Assume that there was at least one departure from class i before the time t, the priority of class i at time t is

$$p_i(t) = \frac{\sum jitter - of\ - all - packets - served}{Number - of\ - packets - served}\Delta_i = \frac{\sum_{k=1}^{s_i(t)} j_i^k}{s_i(t)} \tag{6.5}$$

Where $s_i(t)$ is the number of packets served till time t of class i and j_i^k is the jitter of packet numbered k of class i.

Suppose that a packet has to be selected for transmission at time t. PAJ chooses the backlogged class with the maximum priority at t:

$$k = \arg_{i=1\ldots N} \max p_i(t) \tag{6.6}$$

The selection of the maximum priority, requires at most *N-1* comparisons with *N* is the number of classes, which is a minor overhead for the small number of classes I consider here. The main computation overhead of PAJ is a division, after each packet departure.

The basic idea in PAJ is that if some packets are serviced from class *j* with the maximum priority, the delays of these packets remain similar and hence its jitter does not increase any more and thus the increase of $\sum_{k=1}^{s_i(t)} j_i^k$ due to these packets are minimized. So serving some packets from class *j* tends to reduce the difference from the priorities of the other classes. In the long run, if the scheduler always minimizes the difference between the priorities in this manner, I expect that the priorities are about the same.

The similarities of PAJ and RJPS are now obvious. In the same way that PAJ chooses for service the class with the maximum priority, RJPS also chooses for service the class with the maximum priority. PAJ attempts to minimize in this manner the differences of the class priority. RJPS maintains priority of a moving window and for all packets in the queue, thus making the forwarding behavior more responsive to current queue conditions, but is more complicated than PAJ. Here are the necessary operations for the implementation of PAJ in a router:

	Receive (packet)
1	$s_i(t) \leftarrow s_i(t)+1$
2.	**queue** in corresponding jitter class;
	Select_packet_to_transmit()
3	$k = \arg\max_{i=1\ldots N} p_i(t)$
4	transmit from jitter class *k*;
5	**foreach** jitter class *i*;
6	$\sum_{k=1}^{s_i(t)+1} j_i^k \leftarrow \sum_{k=1}^{s_i(t)} j_i^k$

Figure 38 PAJ algorithm

6.2.2 Simulations

My simulation study shows that PAJ scheduler approximates the proportional jitter differentiation model.

The simulation model is as follows. The topology used is shown in Figure 39. The links are 6Mps with a latency of 10ms. There are a total of 2 classes 0 and 1. Flow

1 (from S1 to D1) and Flow 2 (from S2 to D2) belong to class 0, while Flow 3 (from S3 to D3) and Flow 4 (from S4 to D4) belong to class 1.
The objective of this simulation study is to evaluate the behavior of PAJ scheduler in terms of long-term jitter ratio and short-term jitter ratio (this short-term jitter ratio is calculated over a moving window of 200 packets)

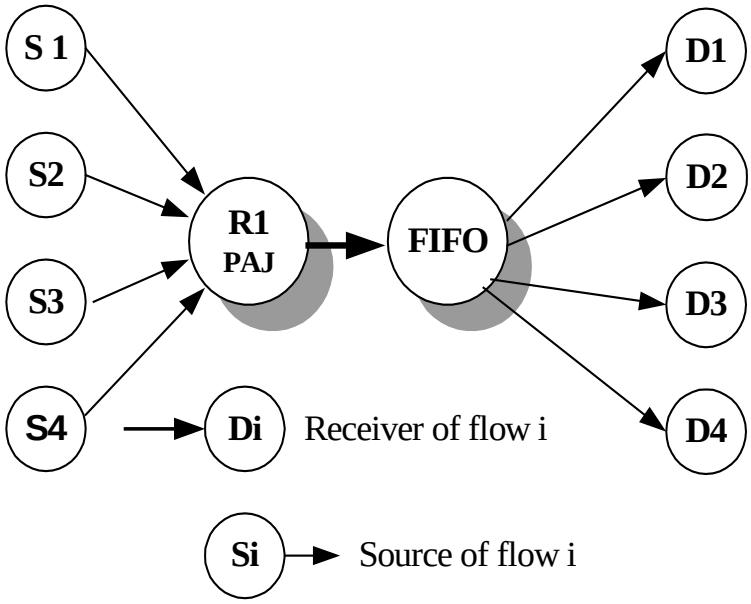


Figure 39 Network topology

6.2.2.1 Behavior of PAJ with Heavy Load

In this simulation, the jitter differentiation parameters of class 0 and 1 are respectively 1 and 2. The predefined ratio between class 1 and class 0 is 0.5. The link utilization between the PAJ router and FIFO router in this simulation is set to 100%. Flow 0 and 1 belong to Class 1. Flow 1 has the burst time of 40ms and idle time of 10ms. For Flow 2 it is 50ms and 20ms respectively. Flow 3 and 4 belong to Class 0. Flow 3 has bursttime of 60ms and idle time of 15ms. Flow 4 has 45ms burst time and 20ms idle time. The total speed of class 0 and class 1 is 3.5 Mps. The first experiment intended to test the performance of PAJ scheduler in terms of long-term jitter ratio and short-term jitter ratio.

Average long-term jitter: Figure 40 shows that average long-term jitter ratio for 2 classes achieve the predefined ratio 0,5. This ratio is achieved after a time of fluctuation of about 6s seconds.

Average short-term jitter: Figure 41 shows that the short-term jitter ratio fluctuates strongly and can reach up to 3.5 and down to 0.17, although the predefined ratio is only 0.5.

Average delay: Clients can only be satisfied if they receive both better jitter and better delay. Hence, it is very important to examine the behavior of PAJ in term of delay because if my algorithm works well for proportional jitter, but a class with higher weight would receive higher delay, it is difficult to conclude that the class with higher weight is better than the class with lower weight. In this simulation, I evaluate average long-term delay for each class, too. Figure 42 shows that delay.

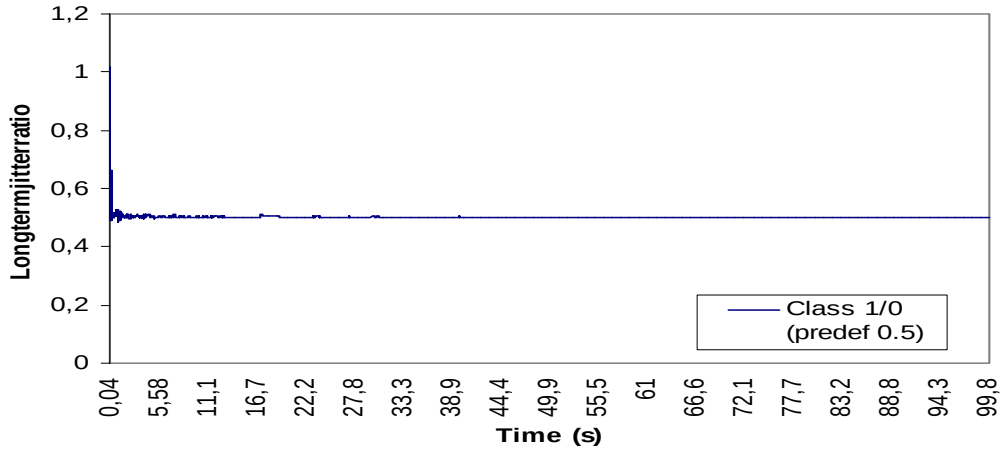


Figure 40 Long term jitter ratio of the PAJ scheduler

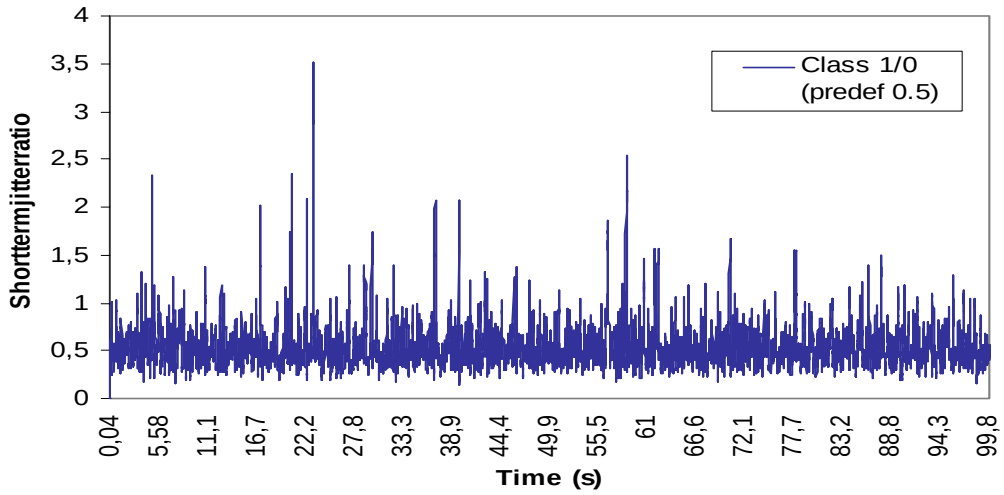


Figure 41 Short-term jitter ratio of the PAJ scheduler

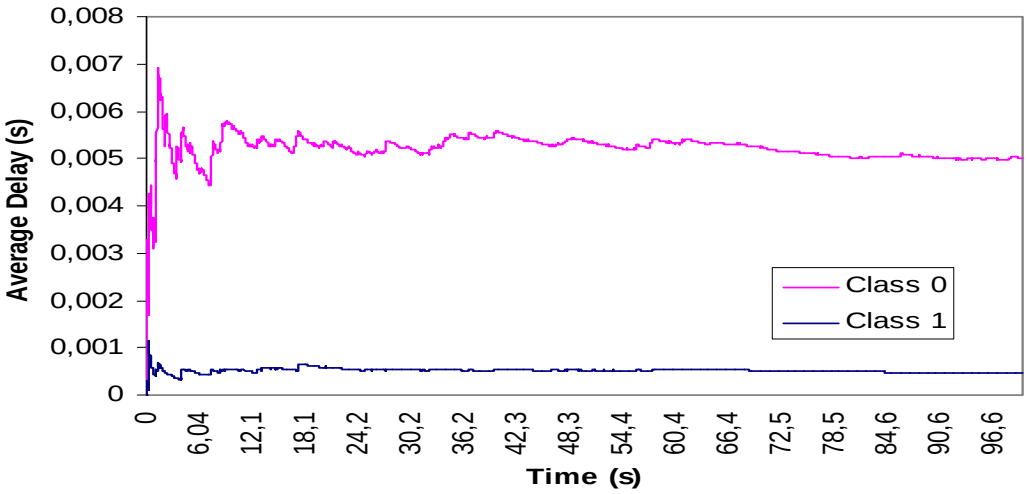


Figure 42 Average delay

6.2.2.2 Behavior of PAJ Scheduler under Different Load Distribution

The second experiment aimed to investigate the long-term jitter ratio and short-term jitter ratio between these two classes under different load distributions. Similar to the first experiment with the same topology, this scenario is set with the predefined jitter ratio 0.5. Flow 1 and 2 belong to Class 1. Flow 1 has the burst time of 100ms and idle time of 30ms and Flow 2 it is 90ms and 40ms respectively. Flow 3 and 4 belong to Class 0. Flow 3 has burst time of 60ms and idle time of 35ms and Flow 4 has 75ms burst time and 30ms idle time. There are 8 simulations in this scenario, in which the load pattern between two classes varied from symmetric to asymmetric distributions. Figure 43 denotes the load distribution of these two classes in percentage.

Results derived from these experiments showed that in most cases, the performance of long-term jitter ratio of PAJ stays nearly constant. As we see in the graph, when the load distribution between classes 0 and 1 is very asymmetric (10%-90% or 90%-10%), the PAJ produces a long-term jitter ratio of 0.6753 and 0,4172, while the predefined ratio is 0.5. In addition, the maximum and minimum long-term jitter ratio is very different from the average and predefined ratio, too. In the other cases, when the load distribution between classes is symmetric (50%-50%), the long-term jitter ratio reaches a very good accuracy.

The short-term jitter ratio produced by PAJ fluctuates much more than long-term jitter ratio. As shown in the following graphs, the maximum of short-term jitter

ratio can reach the value of 87, while the predefined ratio is only 0.5 when the load distribution between two classes is 80%-20%. The average short-term jitter ratios in these 8 cases are around the predefined ratio 0.47318, 0,48823, 0,4648, 0,6231, 1.0127, 0.9786, 1.17786, 2.24526, 2.6751 respectively. That means the quality of short-term jitter ratio depends strongly on the load distribution between classes.

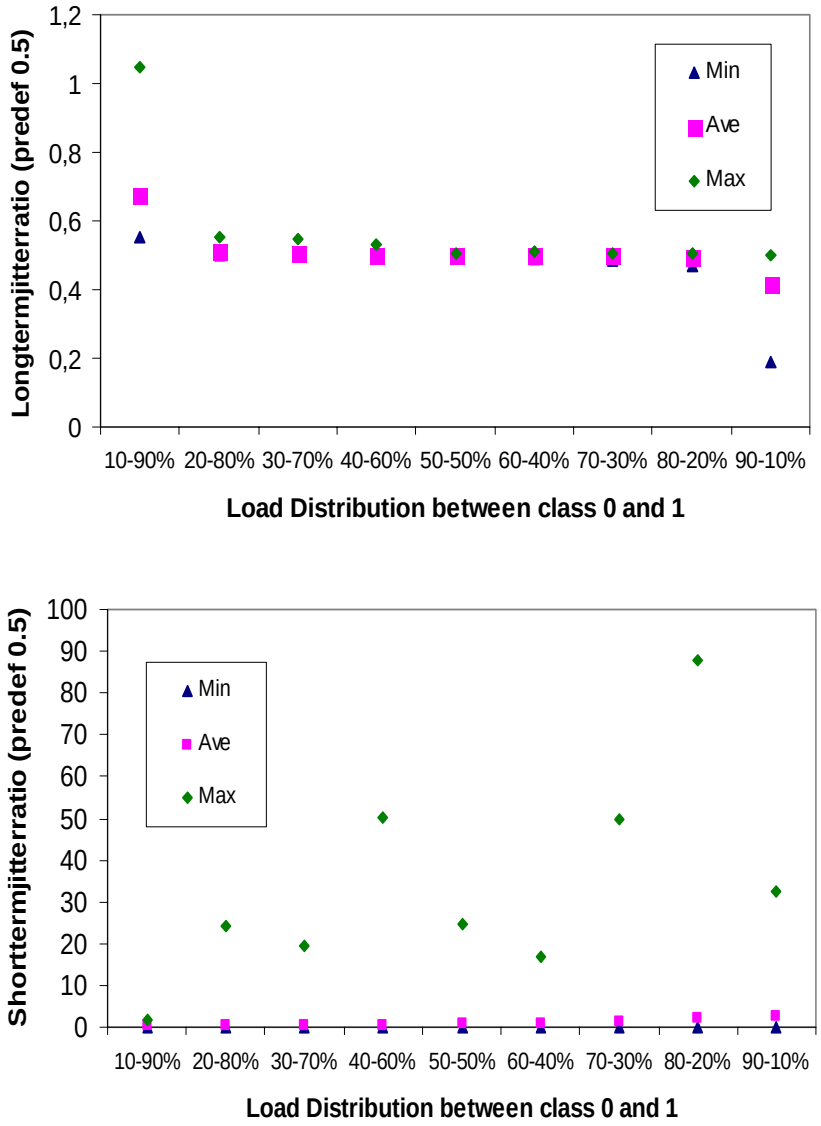


Figure 43 Different load distribution between classes

6.2.2.3 Behavior of PAJ Scheduler under Different Traffic Conditions

In this section, I investigate the performance of the long-term jitter ratio and short-term jitter ratio of PAJ scheduler under different conditions, as the traffic profiles varies .The scenario is similar to the network topology shown in Figure 42, but the traffic profiles are listed in the Table 7. As shown in this table, the long-term jitter ratios stay stable, but the short-term jitter ratio varies, too.

TRAFFIC PROFILES	Flow, on-time, off-time, rate
Case 1	Flow 1: 40ms, 10ms, 1.5 Mbps Flow 2: 50ms, 20ms, 2 Mbps Flow 3: 60ms, 15ms, 1.5 Mbps Flow 4: 45ms, 20ms, 2 Mbps
Case 2	Flow 1: 100ms, 30ms, 2.5 Mbps Flow 2: 90ms, 40ms, 2.5 Mbps Flow 3: 60ms, 35ms, 2.5 Mbps Flow 4: 75ms, 30ms, 2.5 Mbps
Case 3	Flow 1: 45ms, 15ms, 1.5 Mbps Flow 2: 50ms, 20ms, 2 Mbps Flow 3: 70ms, 15ms, 1.5 Mbps Flow 4: 45ms, 15ms, 2 Mbps
Case 4	Flow 1: 90ms, 25ms, 2.5 Mbps Flow 2: 90ms, 20ms, 2.5 Mbps Flow 3: 70ms, 40ms, 2.5 Mbps Flow 4: 75ms, 25ms, 2.5 Mbps
Case 5	Flow 1: 70ms, 20ms, 2 Mbps Flow 2: 50ms, 10ms, 3 Mbps Flow 3: 60ms, 15ms, 0.5 Mbps Flow 4: 80ms, 20ms, 1 Mbps
Case 6	Flow 1: 50ms, 10ms, 2 Mbps Flow 2: 40ms, 20ms, 2 Mbps Flow 3: 30ms, 5ms, 1 Mbps Flow 4: 60ms, 10ms, 1Mbps

Table 7 Different traffic profiles

For long-term jitter ratio, as depicted in Figure 44, my PAJ reaches a very good quality because this ratio is approximately 0.5, which is predefined ratio, too. But the short-term jitter ratio is very unstable. In these 6 cases, the average short-term jitter ratio is 0.5165, 0.5369, 0.5037, 1.3695, 0.5524, 0.5912 respectively and the maximum value of this ratio can reach particularly to 24.01. The smallest minimum value in these 6 cases is 0.01255 (case 4)

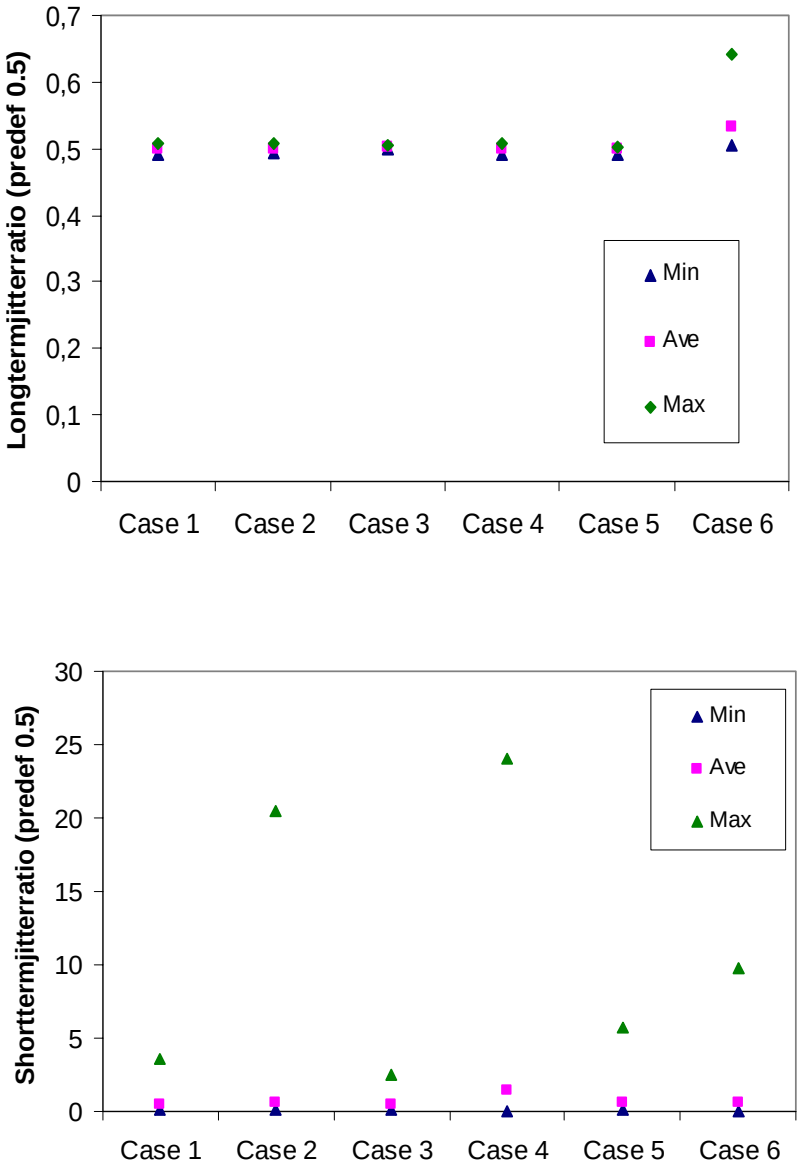


Figure 44 Different traffic profiles

6.3 Adaptive Differentiation Parameter

6.3.1 Algorithm Description

Generally, the goal of Proportional Differentiation Jitter Model is producing constant and accurate jitter ratios:

$$\frac{\bar{j}_i(t,t+\tau)}{\bar{j}_j(t,t+\tau)} = \frac{\Delta_j}{\Delta_i} \qquad \text{or} \quad p_i(t) = p_j(t) \tag{5.1}$$

Until now, there are some approaches for the Proportional Differentiation Jitter Model: RJPS and PAJ. These scheduling algorithms use Jitter Differentiation Parameters as the *fixed* parameters and try to control the estimated normalized jitter so that jitter ratios between different class remain constant.

Contrary to RJPS and PAJ, I can use Jitter Differentiation Parameters as *variable* parameters in order to receive constant jitter ratios. This parameters can vary with the traffic load so that the constant jitter ratio is achieved under bursty traffic context. This idea leads me to create other proportional jitter scheduling algorithm that use Adaptive Jitter Differentiation Parameters for reaching the proportional jitter condition under different load conditions.

Assume that packet *k* from queue *i* leaves the system at time *t*, the *normalized average jitter of all classes* and the *normalized average jitter of class i* are calculated as follows. I normalize jitter j_i^k of packet *k* of class *i* with Jitter Differentiation Parameter Δ_i, and then I calculate the average value of normalized jitters over all jitter classes. Let the *normalized average jitter of class i* be $\bar{j}_i^{normalized-of-class-i}(t)$, the *normalized average jitter of all classes* be $\bar{j}^{normalized-of-all-classes}(t)$, *A(x(t))* be the *average function of variable x* at time *t*, I have:

$$\bar{j}^{normalized-of-all-classes}(t) = A(\Delta_i j_i^k) \tag{6.7}$$

$$\bar{j}_i^{normalized-of-class-i}(t) = A(j_i^k) \tag{6.8}$$

In this section, I present a new variant for my previous algorithms, which attempts to keep the average short-term and long-term jitter of each class as close to

$\bar{j}^{normalized-of-all-classes}$ as possible, that is $\Delta_i \bar{j}_i^{normalized-of-class-i} \rightarrow \bar{j}^{normalized-of-all-classes}$.
The adaptive jitter differentiation parameter $\Delta_i^{adaptive}(t)$ is defined as below:

$$\Delta_i^{adaptive}(t) = \frac{\Delta_i \bar{j}_i^{normalized-of-class-i}(t)}{\bar{j}^{normalized-of-all-classes}(t)}\Delta_i = \frac{\Delta_i A(j_i^k(t))}{A(\Delta_i j_i^k(t))}\Delta_i$$
$$= \frac{A(j_i^k(t))}{A(\Delta_i j_i^k(t))}\Delta_i^2 \qquad (6.9)$$

In this section I use $\Delta_i^{adaptive}(t)$ (*Adaptive Jitter Differentiation Parameters-AJDP*) instead of Δ_i (Jitter Differentiation Parameters-JDP).

The deployment of the AJDPs $\Delta_i^{adaptive}(t)$ instead of JDPs is a mechanism to provide consistent proportional jitter between different classes, independent of varied load patterns. I verify it in next section.

It is noted that different average function *A(.)* can be used. In my approach I use the simple average value from the previous algorithm of RJPS and PAJ.

$$\Delta_i^{adaptive}(t) = \frac{\bar{j}_i^{long-term}(t) * \Delta_i^2 * N}{\sum_{l=1}^{N} \bar{j}_l^{long-term}(t)\Delta_l} \qquad (6.10)$$

Another possibility could be the use of exponential averaging. This algorithm is based on the exponential averaging technique proposed by Jacobson and Karels for estimating TCP round trip times.

From these AJDPs, I create two new algorithms, Adaptive-RJPS and Adaptive-PAJ. These two new algorithms are based on RJPS and PAJ, but make use of AJDPs instead of JDPs.

In the following figures (Figure 45 and 46), the operations of Adaptive-RJPS and Adaptive-PAJ are described.

Receive (packet)

1 $q_i \leftarrow q_i + 1;$

2 **queue** in corresponding jitter class;

Select_packet_to_transmit()

3 $k = \arg\max_i \{p_i(t)\} = \arg\max_i (\bar{j}_i^{\min}(t)\Delta_i^{adaptive}(t))$

4 transmit from jitter class *k*;

5 $q_i(t) \leftarrow q_i(t) - 1;$

6 $s_i(t) \leftarrow s_i(t) + 1;$

7 **foreach** jitter class *i;*

8 $j_i^*(t) \leftarrow j_i^*(t) + \left|(T_i^{k+1} - t_i^{k+1}) - (T_i^k - t_i^k)\right|;$

9 $j_i^{\min}(t) \leftarrow j_i^{\min}(t) + \left|TS_i^{k+1} - (t_i^{k+1} - t_i^k)\right|;$

10 $\Delta_i^{adaptive}(t) = \dfrac{\bar{j}_i^{long-term}(t) * \Delta_i^2 * N}{\sum_{l=1}^{N} \bar{j}_l^{long-term}(t)\Delta_l};$

Figure 45 Adaptive-RJPS algorithm

Receive (packet)

1 $P_i \leftarrow P_i + 1;$

2. **queue** in corresponding jitter class;

Select_packet_to_transmit()

3 $k = \arg\max_{i=1\ldots N} p_i(t);$

4 transmit from jitter class *k*;

5 **foreach** jitter class *i*;

6 $\sum_{k=1}^{s_i(t)+1} j_i^k \leftarrow \sum_{k=1}^{s_i(t)} j_i^k;$

7 $\Delta_i^{adaptive}(t) = \dfrac{\bar{j}_i^{long-term}(t) * \Delta_i^2 * N}{\sum_{l=1}^{N} \bar{j}_l^{long-term}(t)\Delta_l};$

Figure 46 Adaptive-PAJ algorithm

6.3.2 Simulations

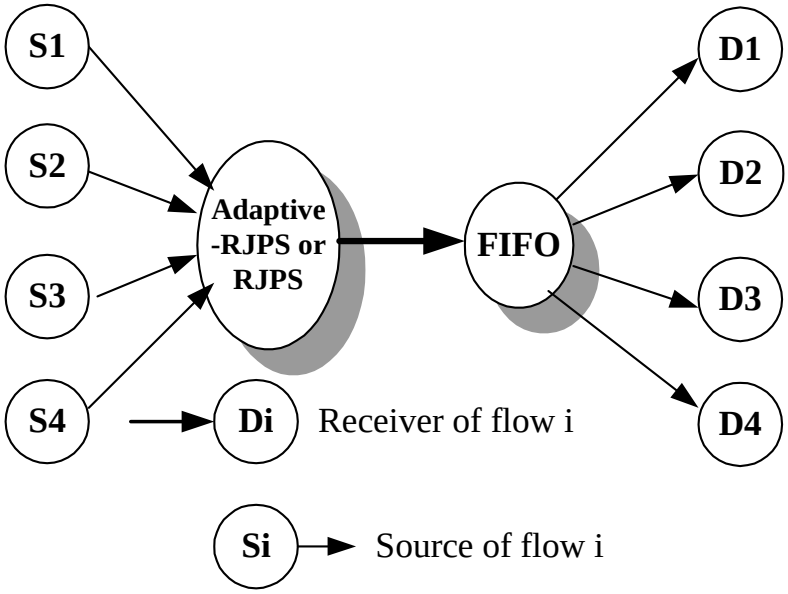


Figure 47 Network topology

I evaluate the performance of my adaptive algorithms and compare it with the previous original mechanisms. There are total of 2 classes 0 and 1. Flow 1 (from S1 to D1) and Flow 2 (from S2 to D2) belong to class 0, while Flow 3 (from S3 to D3) and Flow 4 (from S4 to D4) belong to class1 (see Figure 47).

Figure 48 compares the performances of RJPS and Adaptive RJPS in the same context. The predefined long-term jitter ratio is 0.2. The traffic of type on-off is quite bursty. Flow 1 has a rate of 0.5 Mbps, on time is 30ms and off time 100ms. Flow 2 has a rate of 0.5 Mbs, on time is 65ms and off time 100ms. Flow 3 has a rate of 6 Mbps, on time is 45ms and off time is 90ms. Flow 4 has a rate of 6 Mbps, on time of 35ms and off time 80ms. As we see in the results, the long-term jitter ratio of Adaptive RJPS achieves better quality than the RJPS algorithm.

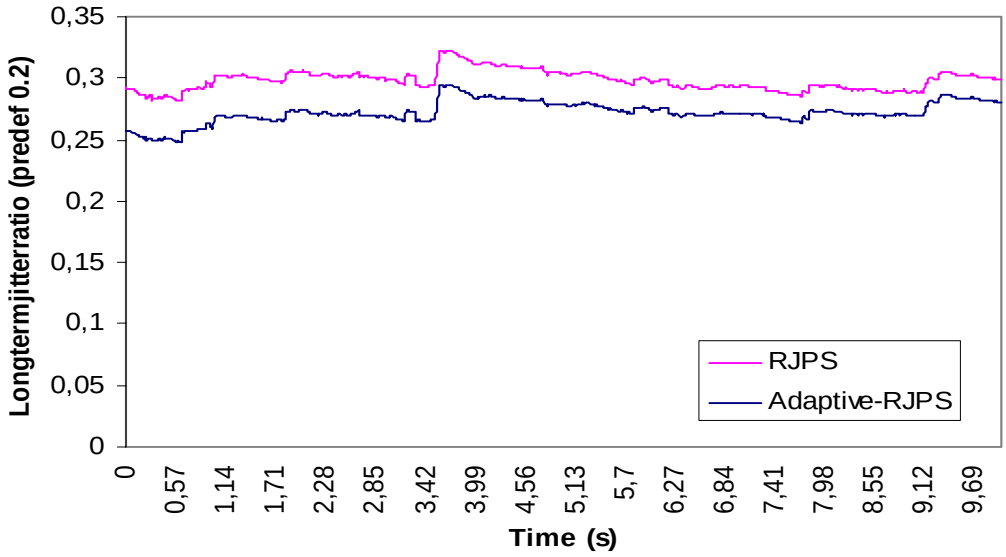


Figure 48 Performance comparison of Adaptive RJPS and RJPS algorithm.

The following experiments intend to test and compare the long-term jitter ratio of Adaptive-RJPS and RJPS under different context. The traffic pattern is described in the following table (Table 8):

Traffic Profile	Flow, on-time, off-time, rate
Case 1	Flow 1: 40ms, 150ms, 0.5 Mbps; Flow 2: 35ms, 110ms, 0.5 Mbps Flow 3: 25ms, 150ms, 6 Mbps; Flow 4: 17ms, 100ms, 6 Mbps
Case 2	Flow 1: 30ms, 100ms, 0.5 Mbps; Flow 2: 65ms, 100ms, 0.5 Mbps Flow 3: 45ms, 90ms, 6 Mbps; Flow 4: 35ms, 80ms, 2.5 Mbps
Case 3	Flow 1: 30ms, 170ms, 0.5 Mbps; Flow 2: 25ms, 150ms, 0.5 Mbps Flow 3: 38ms, 130ms, 6 Mbps; Flow 4: 27ms, 105ms, 6 Mbps
Case 4	Flow 1: 20ms, 200ms, 0.5 Mbps; Flow 2: 29ms, 169ms, 0.5 Mbps Flow 3: 33ms, 145ms, 6 Mbps; Flow 4: 47ms, 156ms, 6 Mbps

Table 8 Traffic profiles

As we see in Figure 49, the Adaptive-RJPS algorithm produces better performance than the RJPS algorithm, when the traffic is very bursty. The

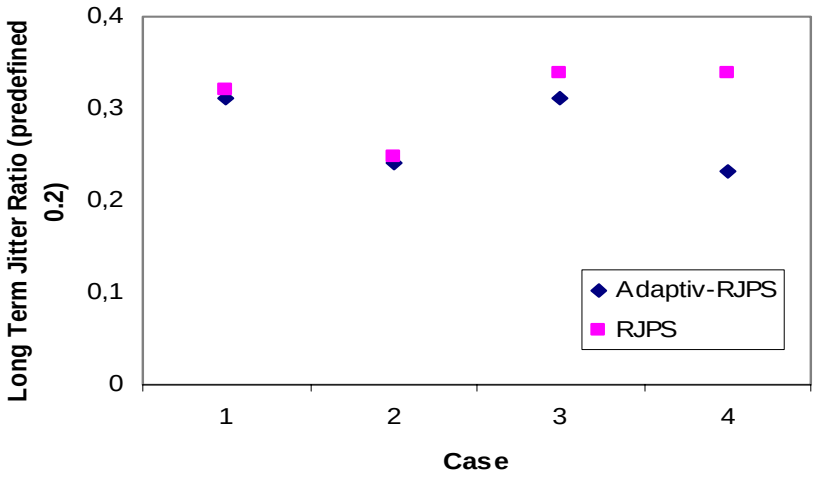


Figure 49 Performance comparison between RJPS and Adaptive-RJPS

predefined long-term jitter ratio is 0.2, and the Adaptive-RJPS always performs a better long-term jitter ratio than RJPS. It means that using Adaptive Jitter Differentiation Parameters increases the quality of the original algorithm RJPS.

Now I investigate the result of long-term jitter ratio when the packet size changes quickly. The traffic profile is showed in the following table (Table 9). At the beginning I set the packet size 160bytes, and this size is changed as indicated in the table. The predefined jitter ratio is 0.5 and the traffic is set to be consistent for verifying only the influence of packet size on the behaviors of my algorithms. Figure 50 points the performance of long-term jitter ratio along the time. Results from this experiment showed that in most cases, the performance of Adaptive-RJPS is better than the RJPS algorithm.

TRAFFIC PROFILE	Flow, on-time, off-time, rate. The change of packet size at time
Case 1	Flow 1: 200ms, 5ms, 3 Mbps. 10s 1024bytes; 30s 512bytes; 50s 256bytes. Flow 2: 150ms, 10ms, 3 Mbps. 35s 512bytes; 50s 216 bytes; 60s 1024bytes; 80s 96 bytes. Flow 3: 180ms, 5ms, 6 Mbps. 20s 128 bytes; 35s: 96bytes; 50s 512 bytes; 69s 1024 bytes; 85s 2045 bytes. Flow 4: 170ms, 10ms, 6 Mbps. 20s: 96 bytes; 40s 256bytes; 55s 1024 bytes; 75s 2056 bytes.

Table 9 Traffic profile

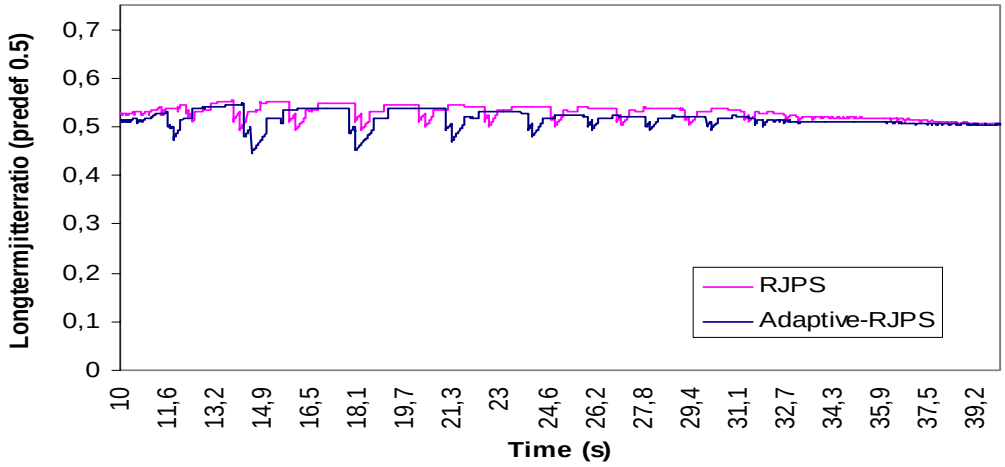


Figure 50 Performance comparison between RJPS and Adaptive RJPS

Similarly, I examine now the performance of Adaptive-PAJ and PAJ algorithm. I use the same network topology as in Figure 47, which contains only two classes. The two first flows belong to class 0, the others belong to class 1. The predefined jitter ratio is defined as 0.5. The traffic profile is chosen to be not very consistent, and is described in the following table.

TRAFFIC PROFILES	Flow, on-time, off-time, rate
Case 1	Flow 1: 80ms, 100ms, 0.75 Mbps. Flow 2: 60ms, 100ms, 0.75 Mbps. Flow 3: 40ms, 70ms, 4 Mbps. Flow 4: 35ms, 65ms, 4 Mbps.
Case 2	Flow 1: 80ms, 60ms, 1 Mbps. Flow 2: 60ms, 50ms, 1 Mbps. Flow 3: 50ms, 50ms, 3 Mbps. Flow 4: 45ms, 45ms, 3 Mbps.

Table 10 Traffic profile

In Figure 51 I see the variation of long-term jitter ratio of these two algorithms along the time when the traffic profile is indicated as in Case 1. This result points out that the use of Adaptive Jitter Differentiation Parameters improves the quality of jitter ratio, especially when the traffic is very bursty.

When the traffic pattern changes from Case 1 to Case 2, I receive the same result (Figure 52). The meaning of this figure states that in this case the PAJ algorithm receives worse quality than the other.

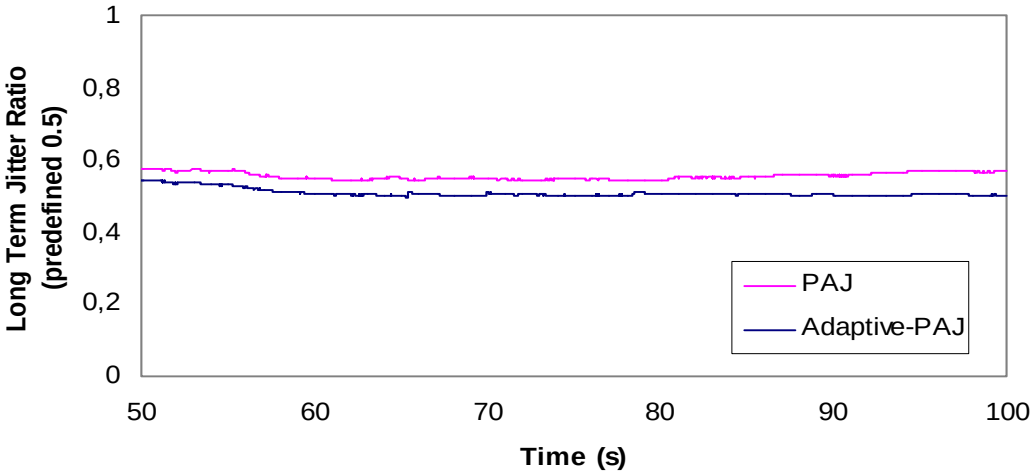


Figure 51 Performance comparison between Adaptive- PAJ and PAJ

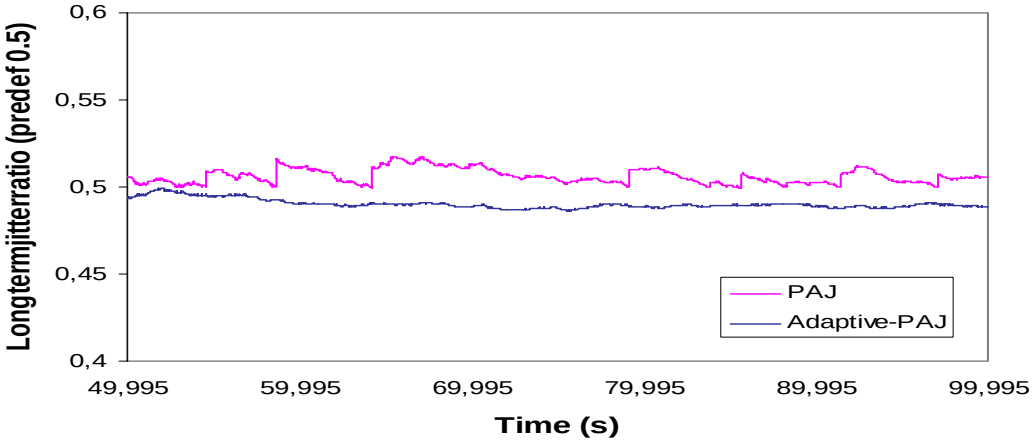


Figure 52 Performance comparison between Adaptive-PAJ and PAJ

In conclusion, the use of AJDPs in RJPS and PAJ helps to improve the performance of jitter ratio, specially when the traffic is bursty, but it makes the new schemes (Adaptive-RJPS and Adaptive-PAJ) also more complicated than its

original variants. The implementation costs of these two new algorithms therefore become higher.

6.4 Performance Evaluation of PJDM and PDDM models

This section will present results on performance evaluation and comparison of PJDM model using my new scheduler algorithms with the PDDM model using WTP. The network topologies and performance criteria used in my simulations are described previously. The content of this section is based on my work published in [Ngo1, Ngo5].

6.4.1 Comparison between Proportional Delay and Proportional Jitter Network

The Proportional Delay Differentiated Services Model PDDM is complex because it is necessary to have proportional-delay scheduling schemes at every router of the network. And even though I have such complicated scheduling algorithm at every router in a PDDM model, each packet is transmitted to the sender through various ways where the number of hops differs from each other. Hence, at the receiver, the sum of packet delay in one class does not stay proportional to the sum of packet delay of other classes any more. It leads me to conclusion that I will not receive proportional delay in a PDDM model except for the case that all packets of one class belong to the same route, so that the number of hops remains unchanged. In addition, when the transmission delay becomes larger compared to the queuing delay in a network, the sum of delay of one class in a PDDM model will not remain proportional any more to each other.

Caused by the need of implementing a complicated proportional-delay scheduling algorithm at all the routers of the network, the PDDM model is not as flexible at high-speed network as the PJDM model.

Finally, although the PDDM model is more complex than the PJDM model, at the receiver the end-to-end delay produced by PJDM model is better than delay produced by PDDM model under some cases.

	Proportional Delay DiffServ Model	Proportional Jitter DiffServ Model
Complexity	Need to have complex scheduling algorithms at all the routers	
Properties	Providing proportional delay between different classes locally	Providing proportional jitter between different classes locally
Scalability	When the transmission delay are dominated and packets have different routes, difficult to achieve proportional delay between different classes	
Efficiency for high speed network	Appropriate at high-speed network	Very appropriate for high-speed network
Quality of Service	Provide poor end-to-end delay	Provide better end-to-end delay in some cases

Table 11 Comparison between the PDDM and the PJDM model

6.4.2 Simulations

6.4.2.1 Jitter Ratio Evaluation through Multi-hop Networks based on PJDM model

As noted above, the jitter ratio produced by schedulers RJPS, PAJ, Adaptive-RJPS and Adaptive-PAJ remain stable under different context when the network contains only one hop. In this section, I present the result of long-term jitter ratio at different routers via a larger topology. This topology is shown in Figure 53.

My network is enlarged and RJPS is used in three routers: R17, R18, R19 while the others are FIFO only. There are six more flows. The two first flows belong to class 0, the next flows belong to class 2 and the two last flows belong to class 1. The weight of classes 0, 1 and 2 are respectively 1; 1,5; 2. The predefined jitter ratio between class 2 and class 0 is 0,5, while the predefined jitter ratio between class 1 and 0 is 0,667. All the links are 6 Mbps with a latency of 10ms, but I am only interested in the jitter differentiation in the routers R17, R18, R19. The packets have a size of 160 bytes, and window size is 200. The total link utilization is 99%.

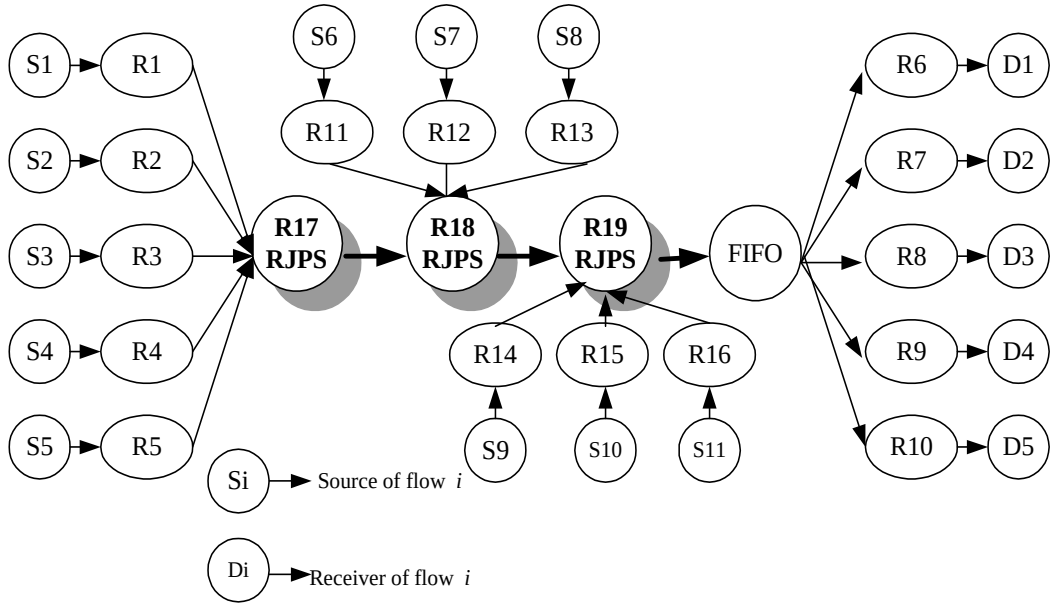


Figure 53 Network topology

Figure 54 and Figure 55 show that my scheduler achieves approximately the long-term jitter differentiation ratio via different routers.

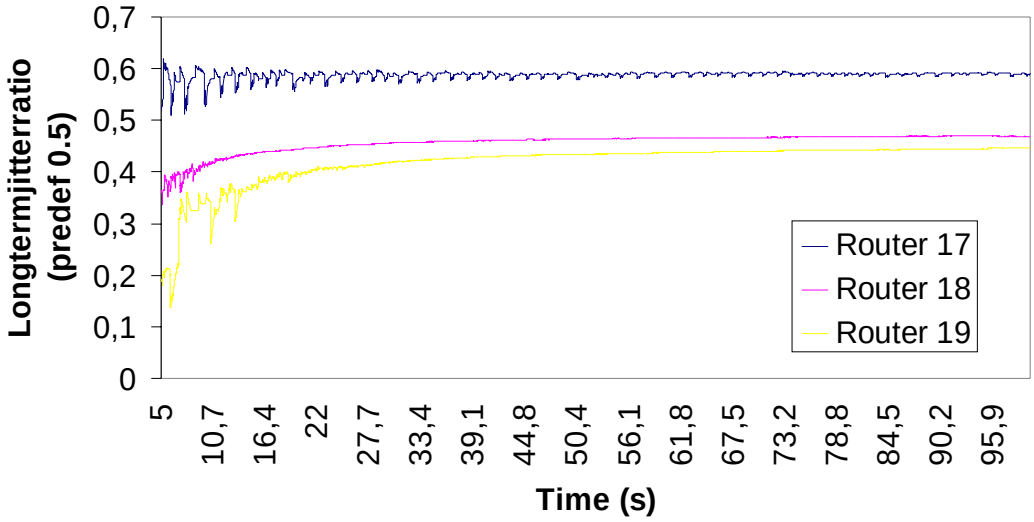


Figure 54 Long-term jitter ratio between class 2 and 0, large topology (predefined: 0.5)

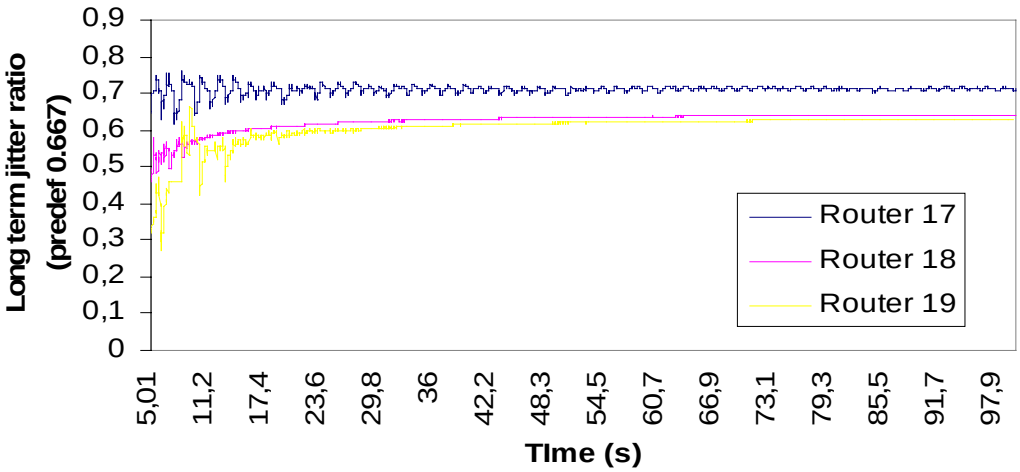


Figure 55 Long-term jitter ratio between class 1 and 0, large topology (predefined: 0.667)

As shown in these two figures, the long-term jitter ratios achieve their predefined value: 0.5 and 0.667.

6.4.2.2 Comparison of PDDM and PJDM when Scheduling is implemented at Every Router (type 1).

In this simulation, I compare the performance of PDDM and PJDM model in case when scheduling is implemented at every router (type 1). PDDM uses WTP as scheduling algorithm, while PJDM uses RJPS. The topologies are illustrated in the following figure:

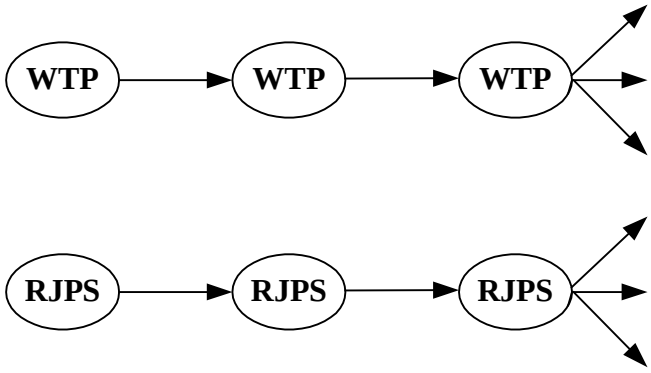


Figure 56 Network Topology, type 1

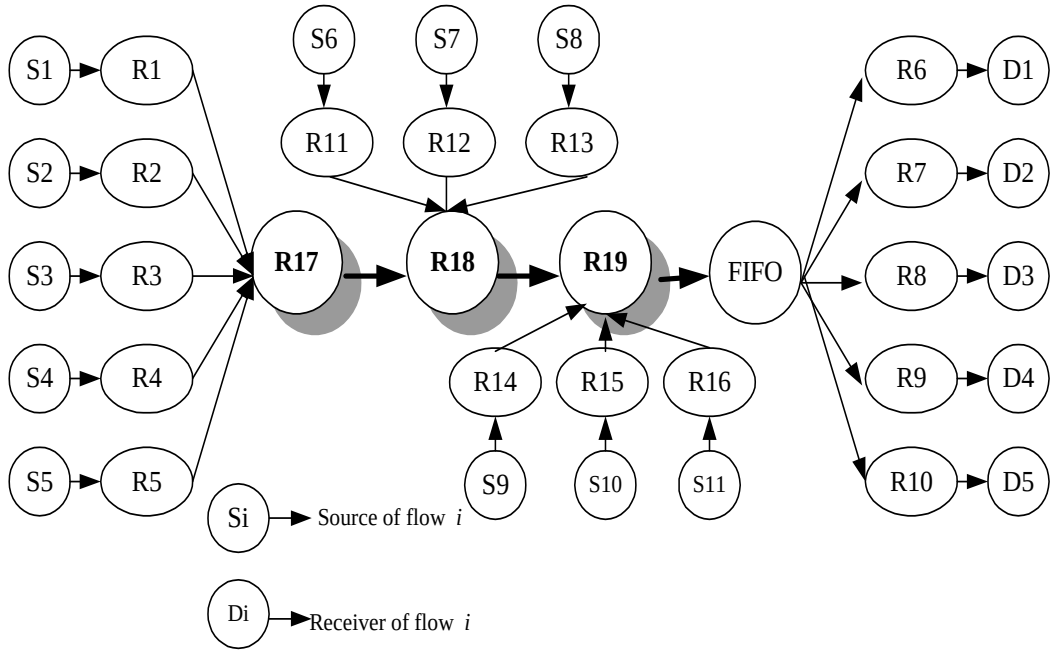


Figure 57 Network topologies

The topology used is shown in Figure 57. The links are 6 Mps with a latency of 10ms. There is a total of 3 classes 0, 1, 2. Flow 1 (S1- D1), 2 (S2- D2), 6 (S6-D1) and 9 (S9- D1) belong to class 1, while flow 3 (S3- D3) , 4 (S4- D4), 7 (S7- D2) and 10 (S10- D2) belong to class 2 and flow 5 (S5- D5), 8 (S8- D3) and 11 (S11- D3) belongs to class 0. The weights of class 0, 1, 2 are 1, 1.5 and 2 respectively.

Figure 58 shows the network delay of class 0. This network delay is produced after three routers R17, R18 and R19 at the topology above. The result shows that network that contains only RJPS produces smaller network delay than network that contains WTP. But as we can see, RJPS fluctuates much more than WTP, which is a very stable mechanism.

The normalized end-to-end delay of this topology (Figure 59) shows that network that contains my algorithm RJPS produces smaller normalized end-to-end delay, that means better end-to-end quality of service. In my simulation I use a window of size from 3000 packets for updating the network delay of Concord algorithm and this window is calculated once pro 200 packets for saving the cost of computation. The predefine loss rate is chosen 10%.

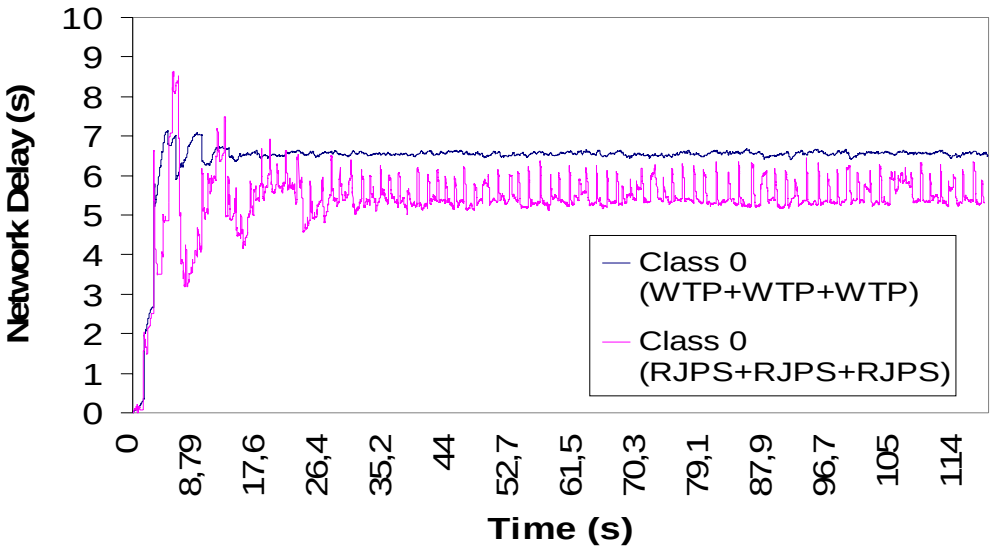


Figure 58 Network delay of class 0

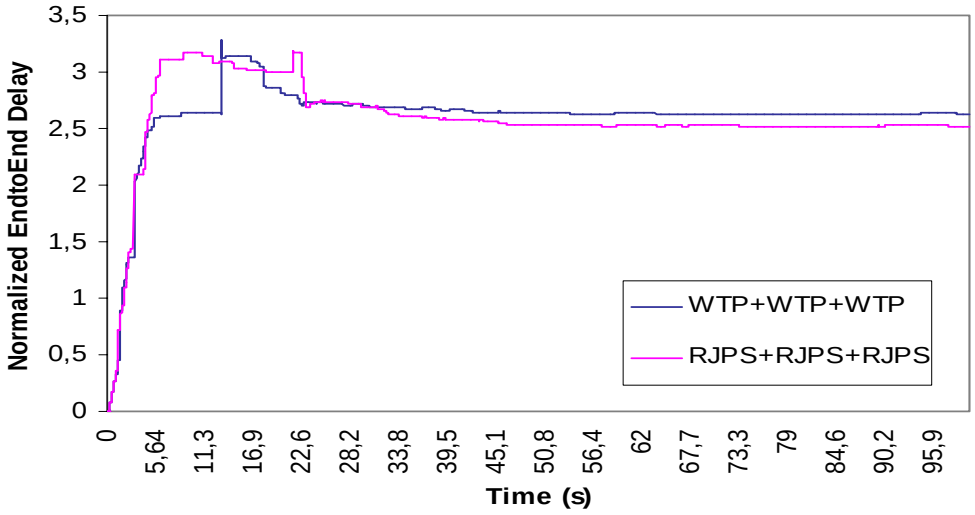


Figure 59 Normalized end-to-end delay of two topologies

6.4.2.3 Comparison of PDDM and PJDM when Scheduling is implemented at Every Router (type 1) or only at Egress Router (type 2).

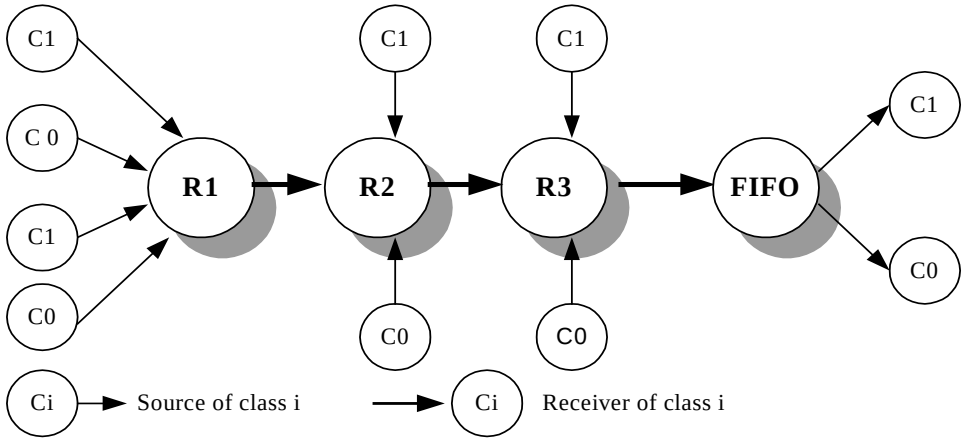
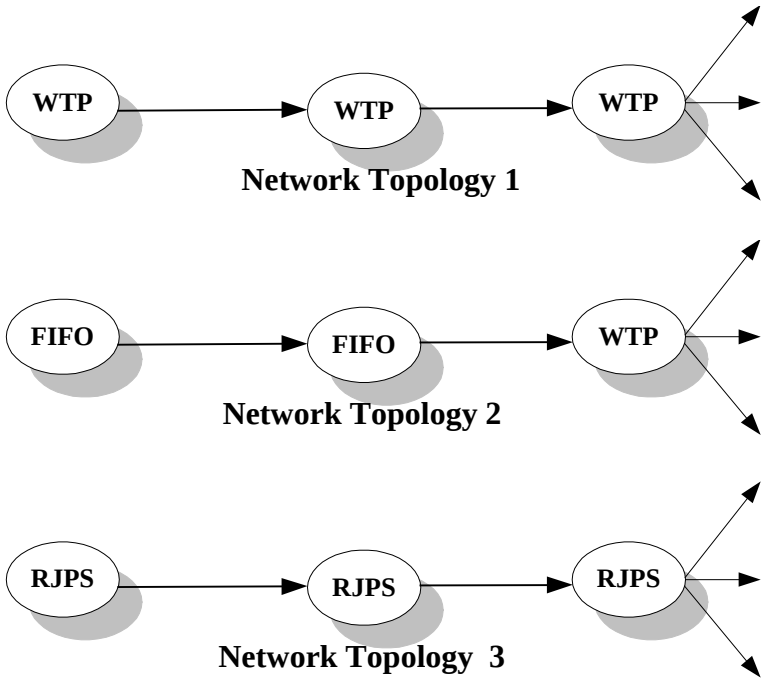


Figure 60 Network topology

I simulated a network of 3 routers. The algorithm PAJ, RJPS, Adaptive-RJPS, Adaptive-PAJ and WTP are implemented at different positions of this network, core or egress, according to the network topologies in Figure 61.



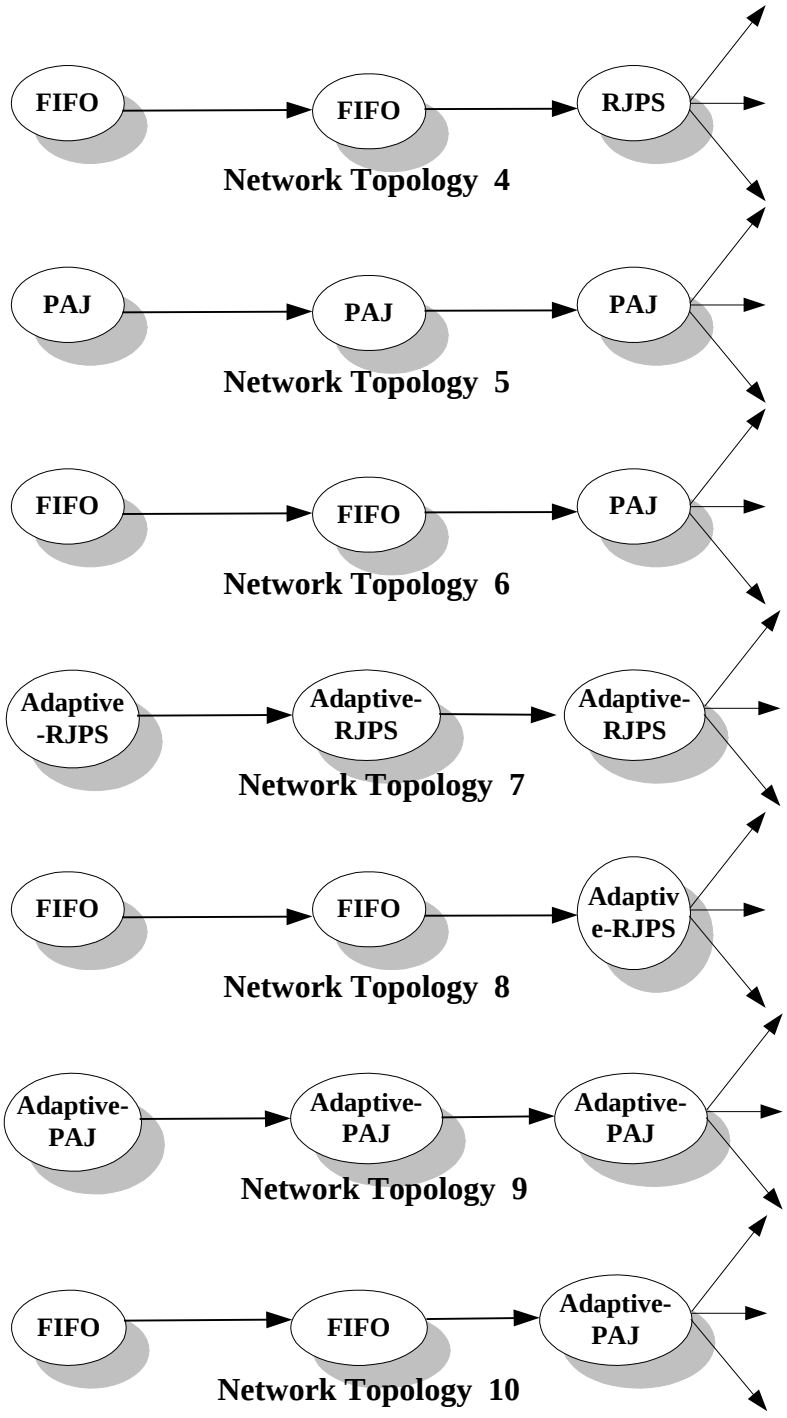


Figure 61 Network topology

TRAFIC PROFILES	Flow, on-time, off-time, rate
At R1	Flow 1: 80ms, 5ms, 2 Mbps Flow 2: 70ms, 5ms, 2 Mbps Flow 3: 70ms, 5ms, 2 Mbps Flow 4: 60ms, 5ms, 2 Mbps
At R2	Flow 1: 80ms, 5ms, 6 Mbps Flow 2: 200ms, 5ms, 6 Mbps
At R3	Flow 1: 200ms, 5ms, 6 Mbps Flow 2: 200ms, 5ms, 6 Mbps

Table 12 Traffic profiles

The simulation scenario is described as follows. The rate of the links from R1 to R2 and from R2 to R3 is 6 Mpbs, 3 Mpbs and 1.5 Mpbs, respectively. There are a total of 2 classes: Class 0 and 1 with the weight of 1.0 and 3.0. At the R1 there are 4 flows, at the R2 and R3 there are only 2 flows. I run and collect my simulations in 100 seconds. The traffic profiles are described in the Table 12.

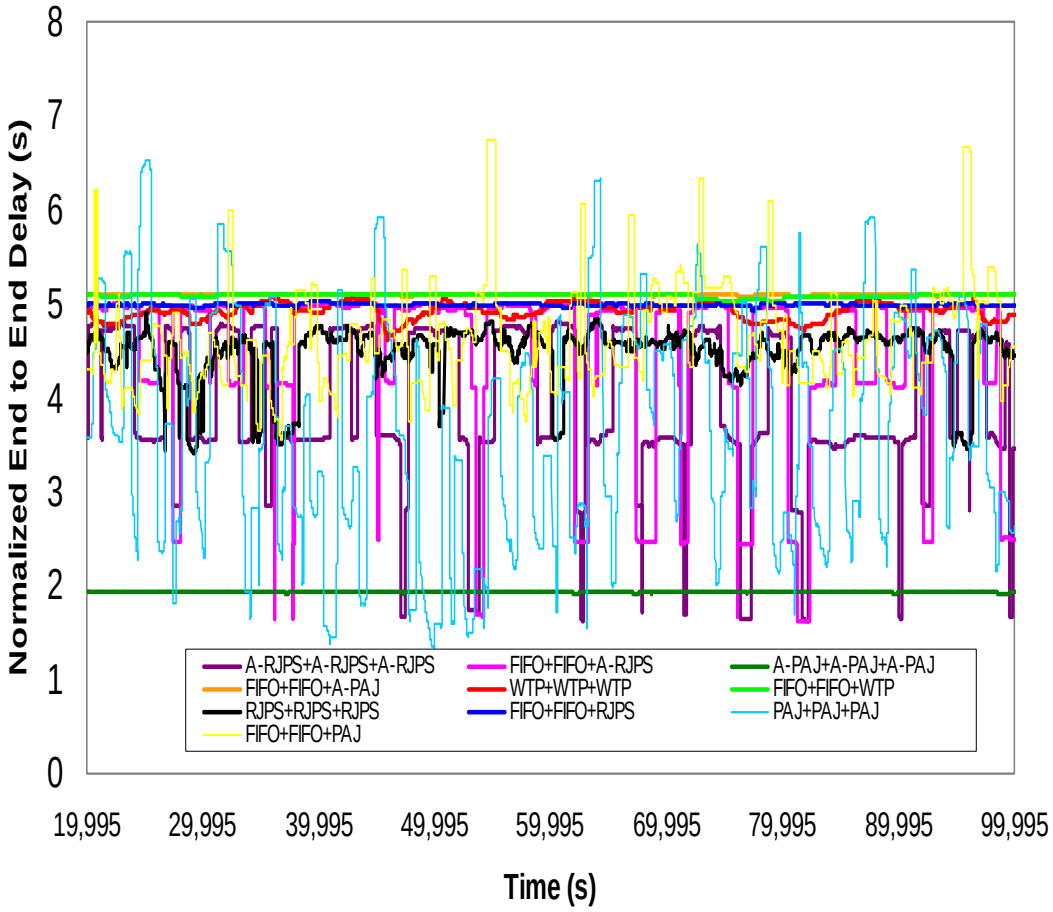


Figure 62 Normalized end-to-end delay

The size of the moving window used at the playout buffer is 3000 packets, and the loss ratio is set to 15%. The normalized end-to-end delay is shown in Figure 62.

It is necessary to say that in order to compare the normalized end-to-end delay of two networks, I say that the network that produces small but fluctuating delay is better than the network that has stable but high delay. This observation is based on the following reason. For the interactive application, the variation of delay over long-term scale (seconds, in my simulations) do not have bad impact on the quality of service. In contrast to the long-term scale, the variation of delay over short-term scale will degrade the quality of service of these interactive applications considerably. Fortunately, the variation over short-term scale produced by networks are removed by the Concord algorithm used at receiver end.

From the above case, we can see that the topology that provides the best performance is the case with only Adaptive-PAJ in its routers. The normalized end-to-end delay produced by this case is small and very stable, while the performance produced by FIFO+FIFO+Adaptive-PAJ is also stable but much higher. Compared to other topologies, the network that implements only Adaptive-PAJ has the best performance.

Unlike the stable case of Adaptive-PAJ, the second case is the networks that use Adaptive-RJPS algorithm. As shown in the previous figure, the normalized end-to-end delay produced by the only Adaptive-RJPS case is better than FIFO+FIFO+Adaptive-RJPS but worse than the only Adaptive-PAJ case, and fluctuate also very quickly.

The PAJ+PAJ+PAJ case has a very unstable performance, and it is still worse than the only Adaptive-PAJ case. The yellow curve that is the normalized end-to-end delay produced by FIFO+FIFO+PAJ topology is high, and oscillates quickly. As you see in the figure, the yellow curve is some times higher than all the rest curves.

Now, I examine the peformance of the network that has RJPS algorithm in its routers. The FIFO+FIFO+RJPS has an approximate performance as the FIFO+FIFO+Adaptive-PAJ. The only RJPS case produces higher quality than the only Adaptive-RJPS case but it is more stable.

Finally, the performance of the only WTP and the FIFO+FIFO+WTP cases are both very high but stable. I can say that these topolgies are worse than all the other cases.

I conclude the following:

For the PJDM model:

- FIFO+FIFO+RJPS topology produces high and stable normalized end-to-end delay. Compared to the previous configuration, RJPS+RJPS+RJPS topology improves the normalized end-to-end delay, but this delay fluctuates strongly than the others.

- FIFO+FIFO+PAJ and PAJ+PAJ+PAJ topologies create smaller (in some times) but unbalanced normalized end-to-end delay.

- The best case is Adaptive-PAJ+Adaptive-PAJ+Adaptive-PAJ topology, whose delay and jitter are smallest. The case of FIFO+FIFO+Adaptive-PAJ produces considerable jitter, and its normalized end-to-end delay remains high. I can say that the use of Adaptive-PAJ improves the quality of PJDM that uses PAJ in its network.

- Adaptive-RJPS+Adaptive-RJPS+Adaptive-RJPS and FIFO+FIFO+Adaptive-RJPS minimize the normalized end-to-end delay. However, their jitters still stay considerable. That means compared to RJPS, the use of Adaptive-RJPS in PJDM model decreases the normalized end-to-end delay, but also make this normalized end-to-end delay fluctuate strongly

For the PDDM model:

- With network topologies of the PDDM model (FIFO+FIFO+WTP and WTP+WTP+WTP) delay is stable, but considerable compared to other topologies of the PJDM model.

In [Ngo2] the quality of network topologies based of type 1 and 2 using WTP. RJPS and PAJ as scheduling algorithm are evaluated. The performance of these networks is shown in Figure 63. The loss rate used by Concord is 15%.

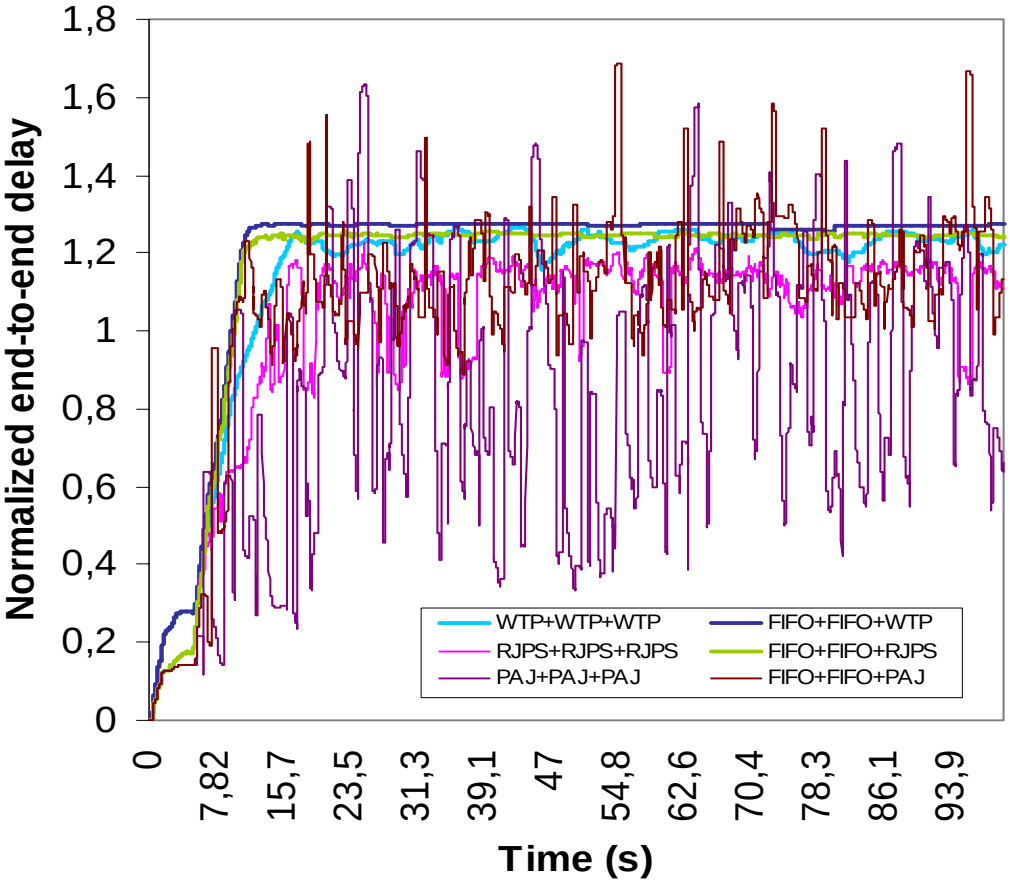


Figure 63 Normalized end-to-end delay

From the figure 63, I have the following markings: The topology PAJ+PAJ+PAJ has the smallest performance compared to all other cases, but this normalized end-to-end delay fluctuate strongly, while the FIFO+FIFO+PAJ network produces higher and also unstable performance. Compared to PAJ case, the only RJPS case has higher normalized end-to-end delay, but does not oscilliate so quickly as the only PAJ cases. Compared to all previous cases, the topology FIFO+FIFO+RJPS has the worst but very stable performance. Finally, the WTP+WTP+WTP and FIFO+FIFO+WTP cases generate the same quality as the FIFO+FIFO+RJPS case. All these conclusions are based on my assumption that is already noted above: small delay with high long-term jitter is better than high but stable delay.

The simulative results from my simulations show that network topologies of PJDM model can produce smaller normalized end-to-end delay in some case, but this delay fluctuates strongly. The network topologies based on the PDDM model produce stable normalized end-to-end delay, but considerable compared to the network topologies based on PJDM model. This observation leads me to an

interesting idea in order to benefit these two advantages of PDDM and PJDM models: use network topologies of the PDDM model, but implement scheduling algorithms of PJDM model at the egress router. Other possibility is using network topologies based on PJDM model, but implementing scheduling of PDDM at egress router. The following section will evaluate the performance of these topologies.

6.4.2.4 Mixture of PJDM and PDDM Models

In this simulation, I create the network topologies, which use both of scheduling of PJDM and PDDM models. The RJPS algorithm is chosen for PJDM model, and WTP is chosen for the PDDM model. In the following figure we find the network topologies that use RJPS and WTP at different position (at egress router, at ingress router and at every router):

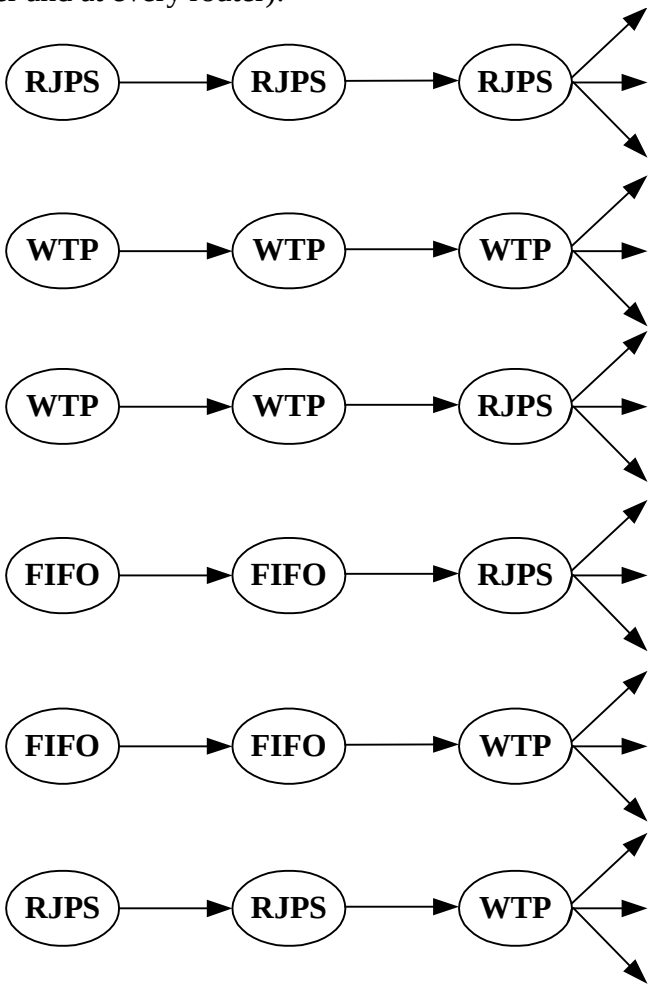


Figure 64 Mixture of PDDM and PJDM models

The topology in simulation is in the following:

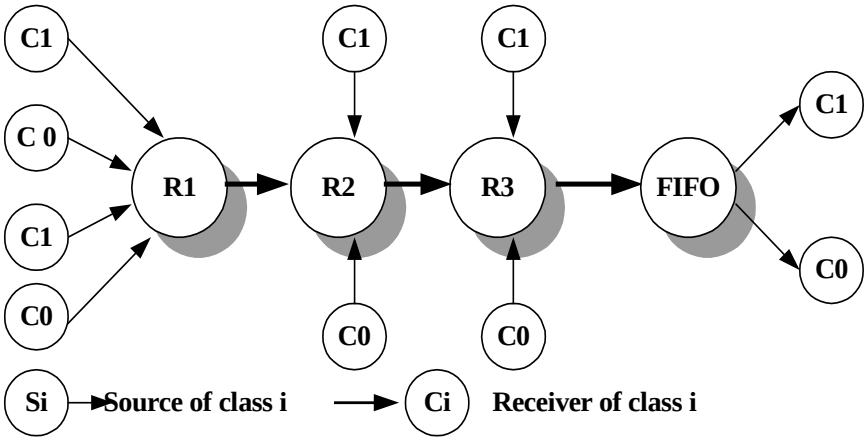


Figure 65 Network topology

The simulation model is as follows. The links are 6 Mbps, 3 Mbps, 1.5 and 0.75 Mbps. The Concord algorithm is used at the receivers. It is necessary to note that I should maintain a window of packets in order to calculate the PDD function. In my simulations, I use PDD taken over a moving packet window of size 3000 packets for Concord algorithm. The two classes 0 and 1 have weights of 1 and 3.

Figure 66 shows the variation of the normalized delay P(s) of these network topologies while the predefined loss rate at the receiver is 5%.

From this graph, I have the following conclusions:

- The RJPS+RJPS+RJPS, which uses RJPS at all the routers, produces the smallest normalized end-to-end delay, and that means the best performance.

- The last topology RJPS+RJPS+WTP, which contains WTP at the egress router and others are RJPS, generates worse quality than RJPS+RJPS+RJPS, WTP+WTP+RJPS and FIFO+FIFO+RJPS, but better quality than WTP+WTP+WTP and FIFO+FIFO+WTP.

- The quality of WTP+WTP+RJPS and FIFO+FIFO+RJPS stays between the quality of RJPS+RJPS+WTP and RJPS+RJPS+RJPS.

- The two other network topologies, WTP+WTP+WTP and FIFO+FIFO+WTP, which use all WTP at all its routers or only at egress router, receive the biggest normalized end-to-end delay. I could say

WTP+WTP+WTP and FIFO+FIFO+WTP generate the worst performance in this case.

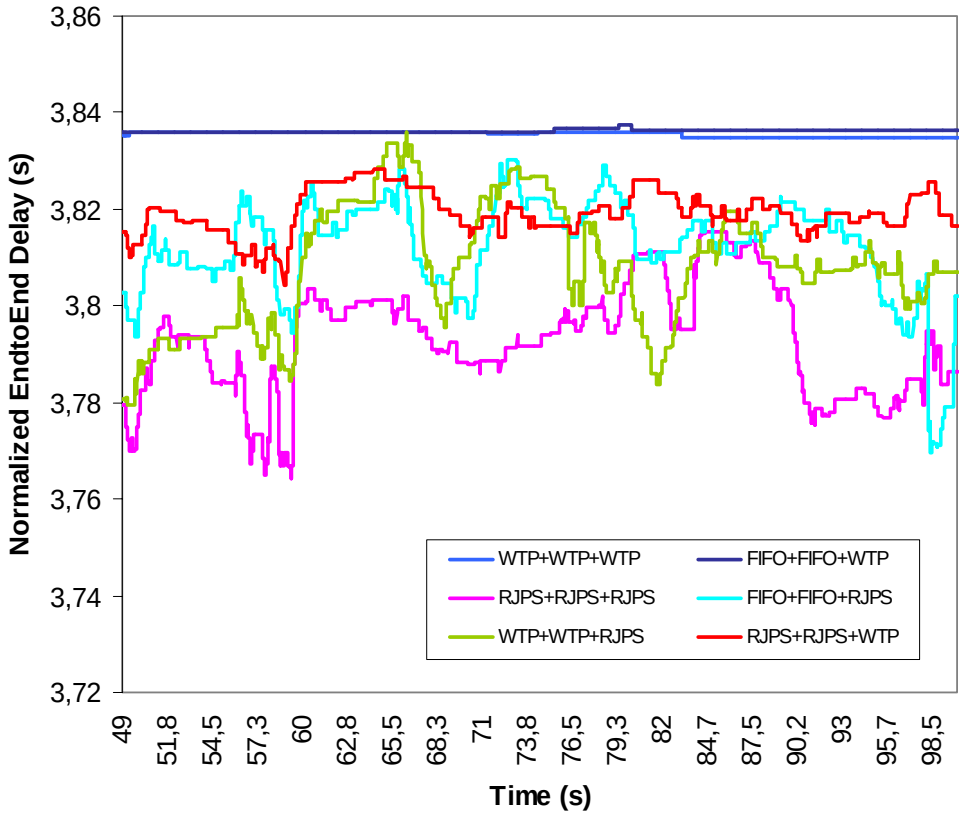


Figure 66 Normalized end-to-end delay

It is very interesting to note that although FIFO+FIFO+RJPS requires only RJPS at its egress router, it could generate better quality in this case than WTP+WTP+WTP, which implements WTP at all its routers. This property allows me to say that implement RJPS at the egress router in a DiffServ network improves the performance of this network while reducing the implementation cost extremely in some cases.

The result in the Table 13 is the average normalized end-to-end delay with different loss rate probabilities from 5% to 15%. Figure 67 plots the quality of these network topologies and puts it in order of the decreasing quality. Results from this figure show that the performance of networks based on PJDM model and based on both PDDM and PJDM increases with the loss rate at the receiver.

It is easy to see that in this case the quality of the networks, which contain RJPS at its routers increases when the loss rate increases, and compared to others topologies, which use WTP at its routers, it achieves higher performance. This

result leads me to a conclusion that in certain cases, the network which uses only RJPS at its egress router receives better performance quality than the network, which uses WTP at all of its routers. On the other hand, the implementation of RJPS at the egress router of networks improves the quality of the network while reducing the cost of implementation extremely.

	RJPS+RJPS+RJPS	WTP+WTP+RJPS	FIFO+FIFO+RJPS	RJPS+RJPS+WTP	WTP+WTP+WTP	FIFO+FIFO+WTP
Loss 5%	3,793339778	3,812344756	3,807713956	3,819351	3,835465444	3,8363002
Loss 10%	3,376311822	3,3807564	3,597968467	3,642769022	3,733399022	3,828120133
Loss 15%	2,897365289	2,927831911	2,943926222	3,306512533	3,382277956	3,8363002

Table 13 Comparison of normalized end-to-end delay

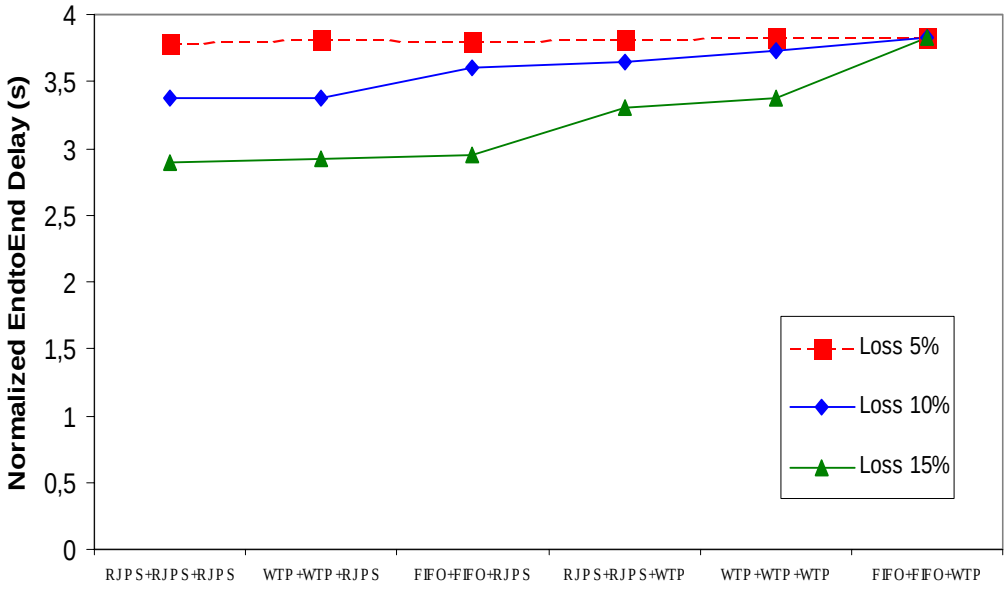


Figure 67 Performance comparison between different network topologies

As shown in Figure 66, normalized end-to-end delay produced by the network topologies based on both of PDDM and PJDM (WTP+WTP+RJPS and RJPS+RJPS+WTP) is better but fluctuate more strongly than the network based on PDDM (WTP+WTP+WTP). Compared to the topology based on PJDM (RJPS+RJPS+RJPS), this normalized end-to-end delay is higher but its jitter is improved.

That means by mixing PDDM and PJDM, the end-to-end quality is improved.

Chapter 7 Summary

In this chapter I give some conclusions on my works and outlines further possible research on the direction.

7.1 Summary

The objective of this thesis is to develop a service-differentiation architecture for the Internet that can improve the end-to-end quality of service as well as easy to deploy and to manage.

Because of non-scalability and complexity issues of the IntServ architecture I have chosen to follow the framework of DiffServ. In addition the absolute differentiation architecture such as Virtual Leased Line or Assured service is not taken into my consideration due to its requirement of admission control and inter-domain resource reservations or careful provisioning.

The architecture that best met my requirements for scalability and simplicity best is the relative proportional differentiation model. Quality of service is dependent on delay, bandwidth or/and loss. Furthermore the model of proportional-delay differentiation PDDM and proportional loss differentiation already exist. This is one of different reasons that lead me to design a new architecture – PJDM, which does not result in proportional delay or loss rate but proportional jitter between classes. Since PJDM controls proportional jitter in the network, it 1) does not require to have scheduling mechanism at every router as PDDM and 2) it can cooperate well with the playout buffer delay adjustment algorithm implemented at the receiver for providing better end-to-end quality of service than PDDM. In other words, PJDM can reduce the implementation cost extremely while providing better end-to-end quality of service than PDDM

In order to provide proportional jitter in PJDM networks, it is necessary to develop different scheduling algorithms that produce proportional jitter between classes. In my thesis four packet schedulers are developed: RJPS, PAJ, Adaptive-RJPS and Adaptive-PAJ. All these four schedulers can be implemented in high-speed networks. Via simulation, I demonstrate that the algorithm RJPS and PAJ can produce accurate proportional jitter ratio under different contexts, as variable link utilization, variable window size, variable packet size. The PAJ algorithm is shown to be simpler and hence easier to deploy than RJPS algorithm especially at high-speed networks. However, these two schemes have a disadvantage: their quality depends on link utilization and traffic profile. For example these schemes

can only perform a good long-term and short-term jitter ratio when the link utilization is more than 80% and the traffic is not bursty. In order to improve the quality of RJPS and PAJ under bursty traffic and small link utilization, Adaptive-RJPS and Adaptive-PAJ have been developed. Results via simulation demonstrate that these adaptive mechanisms improve the quality of jitter ratio, especially under bursty traffic condition.

The performance of PJDM model and PDDM model is examined in this thesis via simulation. My simulative results show that the networks based on PJDM model is simpler and can achieve better end-to-end quality of service than networks based on PDDM model. In some cases, a PJDM network that uses only RJPS at its egress router can produce smaller normalized end-to-end delay than a PDDM network that uses WTP at every router. Compared to PJDM using RJPS, the use of PAJ in PJDM decreases the normalized end-to-end delay but also increases its jitter considerably. In addition, the use of Adaptive-RJPS (or Adaptive-PAJ) increases the performance of PJDM model compared to the use of RJPS (or PAJ).

The models PJDM and PDDM have their own disadvantages and advantages: PDDM produces high and stable normalized end-to-end delay while PJDM produces low but fluctuating normalized end-to-end delay. I then establish a combination of PDDM and PJDM models in order to overcome their disadvantages. Results from my simulation show that the normalized end-to-end delay of new network topologies based on both PDDM and PJDM is smaller than the delay produced by PDDM and fluctuates less than the delay produced by PJDM.

Finally, the loss rate at the Concord playout buffer can influence the performance of networks based on PJDM and on both PJDM and PDDM: when the loss rate increases, their performance is also increased.

7.2 Suggestion for Future Work

Throughout the thesis I pointed out specific issues that deserve further research. Here I give some suggestion for future work:

- A valuable simulative study would be carried out larger topologies of more than 3 nodes with different loss rates at receiver end and different traffic profiles. Can my topologies based on PJDM model achieve better performance than the other topologies, which are based on PDDM model in such conditions?

- More experimental study would be carried out to combine the networks (based on the PJDM model) with some wireless access networks. It would

be an interesting direction since jitter produced in wireless domain is considerable so that through my new mechanisms, the gain in terms of end-to-end delay would not be small.

Bibliography

[Alv93] Alvarez-Cuevas, F., Bertran, M., Oller, F., Selga, J., "Voice Synchronization in Packet Switching Networks," *IEEE Network Magazine*, 7(5), pp. 20-25, Sept. 1993.

[Bak96] F. Baker, R. Guerin, and D. Kandlur, "Specification of Committed Rate Quality of Service," draft-ietf-intserv-commit-rate-svc-00.txt, Jun. 1996.

[Bald00] M. Baldi and Y. Ofek, "End-to-End Delay Analysis of Videoconferencing over Packet Switched Networks," *IEEE/ACM Trans*, Net., vol. 8, no.4, Aug. 2000.

[Ban96] A. Banerjea, D. Ferrari, B. A. Mah, M. Moran, D. C. Verma, and H. Zhang, "The Telnet Real-Time Protocol Suite: Design, Implementation, and Experiences," *IEEE/ACM Transactions on Networking*, vol. 4, no. 1, pp. 1-10, February 1996.

[Beg99] A. Begel, S. McCanne and S. L. Graham, "BPF+: Expoiting Global Data-flow Optimization in a Generalized Packet Filter Architecture," *In Proceedings of ACM SIGCOMM*, Sept. 1999.

[Bha99] N. Bhatti and R. Friedrich, "Web Server Support for Tiered Services," *IEEE Network*, pp. 64-71. Sept. 1999.

[Bier96] E. Biersack, W. Geyer, and C. Bernhardt, "Intra and Interstream Synchronization for Stored Multimedia Streams," *ICMCS*, Hiroshima, Japan, June 1996.

[Bol99] G. Bolch, S. Greiner, H. Meer, and K. S. Trivedi. QueuingNetwork and Markov Chains. John Wiley and Sons, 1999.

[Bol93] Bolot, J., "End-to-end Packet Delay and Loss Behavior in the Internet", *In Proceeding of ACM SIGCOMM'93*, pp.289-298, San Francisco, CA, Sept. 1993.

[Bod01] Ulf Bodin, Andreas Jonsson and Olov Schelen, "On Creating Proportional Loss-rate Differentiation: Predictability and Performance," *In Proceeding of IWQoS'2001*, Karlsruhe, Germany, June 6-8, 2001.

[Bou99] J. L. Boudec, M. Hamdi, L. Blazevic and P. Thiran, "Asymetric best- effort Service for Packet Networks," *In Proceeding Global Internet Symposium*, December 1999.

[Brad94] R. Braden, D. Clark, and S. Shenker, "Intergrated Services in the Internet Architecture: an Overview," July 1994. RFC 1633.

[Brad97] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP) – Version 1, Functional Specification," September 1997. RFC 2205.

[Carl97] G. Carle and E. W. Biersack, "Survey of Error Recovery Techniques for IP-Based Audio-Visual Multicast Applications," *IEEE Net*., Vol. 11, no. 6, Nov. 1997, pp. 24-36.

[Card02] Cardoso, K. V., Rezende, J. F., and Fonseca, N. L. S. "On the Effectiveness of Push-out Mechanisms for the Discard of TCP Packets,", *International Conference on Communications 2002*, New York City, NY, USA, April 2002.

[Chao92] H. Chao and N. Uzun,"A VLSI Sequencer Chip for ATM traffic shaper and queue manager," *IEEE Journal of Solid State Circuits*, vol. 27, no. 11, pp. 1634-1643, November 1992.

[Charn00] A. Charny and J. L. Boudec, "Delay Bounds in a Network with Aggregate Scheduling," *In Proceeding QOFIS*, October 2000.

[Chua00] J. Chuang, "Distributed Network Storage Servcie with Quality-of-Service Guarantees," *J. Net. Comp*. App., 2000.

[Chua99] S. T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching Output Queuingwith a Combined Input Output Queued Switch," *In Proceeding of INFOCOM'99*, pp. 1169-1178, New York, CA, March 1999.

[Clark88] D. Clark, "The Design Philosophy of the DARPA Internet Protocols," *In Proceedings of ACM SIGCOMM'88*, pp. 106-114, Stanford, CA, August 1988.

[Clark98] D. D. Clark and W. Fang, "Explicit Allocation of Best-effort Packet Delivery Service," *IEEE/ACM Transactions on Networking*, vol. 6, pp. 362-373, Aug. 1998.

[Clay99] M. Claypool and J. Tanner, “The Effects of Jitter on the Perceptual Quality of Video,” *ACM Multimedia’ 99*, Orlando, FL, 1999.

[Cruz1] R. L. Cruz, “A Calculus for Network Delay, Part I: Network Elements in Isolation,” *IEEE Transactions on Information The*ory, vol. 37, no. 1, pp. 114-131, January 1991.

[Cruz2] R. L. Cruz, “A Calculus for Network Delay: Part II: Network Analysis,” *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 132-141, January 1991.

[Dem90] A. Demers, S. Keshav, and S. Shenker, “Analysis and Simulation of a Fair Queuing Algorithm,” *In Journal of Internetworking Research and Experience*, pages 3-26, October 1990. (Also in Proceedings of ACM SIGCOMM’89, pages 3-12).

[Dov1] C. Dovrolis and P. Ramanathan, “A Case for Relative Differentiated Services and the Proportional Differentiation Model,” *IEEE Network*, October 1999.

[Dov2] C. Dovrolis and P.Ramanathan, “Proportional Differentiated Services, Part II: Loss Rate Differentiation and Packet Dropping,” *In Proceedings of the 2000 International Workshop on Quality of Service (IWQoS)*, Pittsburgh PA, June 2000.

[Dov3] C. Dovrolis, D. Stiliadis and P. Ramanathan, “Proportional Differentiated Services: Delay Differentiation and Packet Scheduling,” *In Proceedings of the 1999 ACM SIGCOMMM*, Cambridge, MA, Sept. 1999.

[Dov4] C. Dovrolis and P. Ramanathan, “RAFT: Resource Aggregation for Fault Tolerance in Intergrated Services Packet Networks,” *ACM Computer Communication Review (CCR)*, April 1998.

[Edel99] R. Edell and P. Varaiya, “Providing Internet Access. What we learn from INDEX,” *IEEE Network*, pp. 18-25, Sept. 1999.

[Enw95] A. Enwalid, D. Heyman, T. V. Lakshman, D. Mitra, and A. Weiss, “undamental Bounds and Approximations for ATM Multiplexers with Applications to Video Teleconferencing,” *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 6, pp. 1004-1016, August 1995.

[Ferr00] T. Ferrari and P. F. Chimento, "A Measurement-based Analysis of Expedited Forwarding PHB Mechanisms," *In Proceedings International Workshop on QoS (IWQoS)*, June 2000.

[Ferg98] P. Ferguson and G. Huston, "Quality of Service: Delivering QoS on the Internet and in Corporate Networks," *John Wiley and Sons*, January 1998.

[Sally1] S. Floyd and V. Jacobson, "Link-sharing and Resource Management Models for Packet Networks," *IEEE/ACM Transaction on Networking*, vol. 3, pp. 365-386, August 1993.

[Sally2] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397-413, Aug. 1993.

[Fran93] D. Frankowski and J. Riedl, "Hiding Jitter in an Audio Stream," Tech. Rep. TR-93-50, Univ. of MN Dept. Comp. Sci., 1993.

[Gar99] S. –H. Gary Chan and F. A. Tobagi, "Caching Schemes for Distributed Video Services," *In Proceeding IEEE ICC*, Vancouver, Canada, June 1999.

[Gey96] W. Geyer, C. Bernhardt, and E. Biersack, "A Synchronization Scheme for Stored Multimedia Streams," Lect. Notes Comp. Sci., vol. 1045, 1996.

[Gop98] R. Gopalakrishnan and G. M. Parulkar, "Efficient User-space Protocol Implementations with QoS Guarantees using Real-time Upcalls," *IEEE Transactions on Networking*, vol. 6, no. 4, pp. 374-388, August 1998.

[Guer97] R. Guerin, S. Kamat, and H. Herzog, "QoS Path Management with RSVP," March 1997. Internet Draft: draft-qos-path-mgmt-rsvp-00.txt.

[Guer00] R. Guerin and V. Pla, "Aggregation and Conformance in Differentiated Services Networks: A Case Study," *In Proceeding ITC Specialist Seminar on IP traffic Modeling, Measurement, and Management*, September 2000.

[Gup99] Pankaj Gupta and Nick McKeown,"Packet Classification on Multiple Fields," *In Proceedings of ACM SIGCOMM'99*, pp. 147-160, Cambridge, MA, September 1999.

[Hart00] F. Hartanto and L. Tionardi, “Effects of Interaction between Error Control and Media Synchronization of Application-level Performances,” *In Proceeding of IEEE GLOBECOM*, 2000, pp. 298-303.

[Jac99] V. Jacobson, K. Nichols and K. Poduri, “An Expedited Forwarding PHB,” June 1999. RFC 2598.

[Jam97] S. Jamin, P. B. Danzig, S. J. Shenker, and L. Zhang, “A Measurement-based Admission Control Algorithm for Intergrated Services Packet Networks,” *IEEE/ACM Transactions on Networking*, vol. 5, no. 1, pp. 56-70, February 1997.

[Kali94] S. Kalidindi and M. J. Zekauskas, “Surveyor: an Infrastructure for Internet Performance Measurements,” *In Proceeding of INET*, San. Jose, CA, June 1999.

[Kang01] J. Kangasharju et al., “Distributed layered encoded Video through Caches,” *In Proceeding of IEEE INFOCOM*, Anchorage, AL, Apr. 2001.

[Kesh98] S. Keshav and R. Sharma, “Issues and Trends in router Design,” *IEEE Communications Magazine*, pp. 144-151, May 1998.

[Klein76] L. Kleinrock. QueuingSystems: Vol 2. Wiley-interscience, 1976.

[Kron91] H. Kroner, G. Hebuterne, P. Boyer, and A. Gravey, “Priority Management in ATM switching Nodes,” *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 418-427, Apr. 1991.

[Kum98] V. Kumar, T. V. Lakshman, and D. Stiliadis, ”Beyond best-effort: Router Architectures for the differentiated Services of tomorrow’s Internet,” *IEEE Communications Magazine*, pp. 152-164, May 1998.

[Lak98] T. V. Lakshman and D. Stiliadis, “High Speed policy-based Packet Forwarding using Efficient Multi-dimensional Range Matching,” *In Proceeding of ACM SIGCOMM’98*, pp. 203-214, Vancouver, Canada, September 1998.

[Laout1] N. Laoutaris and I. Stavrakakis, “Adaptive Playout Strategies for Packet Video Receivers with finite buffer Capacity,” *In Proceeding of IEEE ICC*, Helsinki, Finland, June 2001.

[Laout2] N. Laoutaris and I. Stavrakakis, "An analytical Design of optimal Playout Schedulers for Packet Video Receivers," *Comp. Commun. J., Special Issue on the Quality of Future Internet Services*; an earlier version was presented at 2nd Int'l. Wksp. Quality Future Internet Services, Coimbra, Portugal, 2001.

[Leung00] M.K. H. Leung, J.C.S. Lui and D. K. Y. Yau, "Characterization and Performance Evaluation for Proportional Delay Differentiated Services," *In Proceeding of 8th International Conference on Network Protocols ICNP2000*, Osaka, Japan, Nobember 2000.

[Lieb01] J. Liebeherr and N. Christin, "JoBS: Joint Buffer Management and Scheduling for Differentiated Services," *In Proceeding of IEEE/IFIP Ninth International Workshopon Quality of Service (IWQoS 2001)*, June 2001.

[Lin91] A.-M. Lin and J. A. Silvester, "Priority Queuingstrategies and Buffer Allocation Protocols for Traffic Control at an ATM intergrated broadband Switching System," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 1524-1536, Dec. 1991.

[Liu99] H. Liu and M. El Zarki, "Delay and Synchronization Control Middleware to support real-time Multimedia Services over Wireless PCS Networks," *IEEE JSAC*, vol. 17, no. 9, Sept. 1999, pp. 1660-71.

[Liu97] H. Liu et al., "Error Control Schemes for Networks : an Overview," *Mobile Nets. and Apps.*, vol. 2, no. 2, Oct. 1997, pp. 167-82.

[Liu96] C. Liu et al., "Multipoint Multimedia Telecoference System with Adaptive Synchronization," *IEEE JSAC*, vol. 14, no. 7, Sept. 1996.

[Mank97] A. Mankin, F. Baker, B. Braden, S. Bradner, M. O'Dell, A. Romanow, A. Weinrib, and L. Zhang, "RSVP Version 1: Applicability Statement, some Guidelines on Deployment," IETF RFC 2208, September 1997

[May99] M. May, J. C. Bolot, C. Diot and A. Jean-Marie, "Simple Performance Models of Differentiated Services Schemes for the Internet," *In Proceeding of IEEE INFOCOM*, March 1999.

[Mc00] McCreary, S. Claffy, ‘’Trends in wide area IP traffic patterns – A view from ames Internet Exchange‘’, *In the ITC Specialist Seminar.*

[Mills91] D. L. Mills, “Internet Time Synchronization: the Network Time Protocol,” *IEEE Trans. Commun*., vol. 39, no. 10, Oct. 1991, pp. 1482-93.

[Mont83] W. Montgomery, “Techniques for Packet Voice Synchrnoization”, *IEEE Journal on Slected Areas In Communications*, 6(1), pp. 1022-1028, December 1983.

[Moon98] S. Moon, J. Kurose and D. Towsley, “Packet Audio Playout Delay Adjustment: Performance Bounds and Algorithms,” *Multimedia Systems*, Vol. 6, pp. 17-28, January 1998.

[Nayl82] W. E. Naylor and L. Kleinrock, “Stream Traffic Communication in packet switched Networks: Destination Buffering Constraints,” *IEEE Trans. Commun*., vol. COM-30, no. 12, Dec. 1982, pp. 2527-34.

[Nett79] A. Netterman, I. Adiri, “A Dynamic Priority Queue with General Concave Priority Functions,” Operation Research, 28(6), pp. 1088-1100, 1979.

[Ngo1] T. Ngo-Quynh, A. Wolisz, K. Rebensburg, “Adaptive Jitter Differentiation Parameters for Proportional Jitter Scheduling in Differentiated Services Networks,” Accepted Paper in the *4th IEEE Conference on Mobile and Wireless Communications Networks MWCN 2002*, Stockholm, Sweden.

[Ngo2] T. Ngo-Quynh, H. Karl, A. Wolisz, K. Rebensburg, “New Scheduling Algorithm for Providing Proportional Jitter in Differentiated Services Network,” *In Proceeding of IST Mobile & Wireless Telecommunications Summit 2002*, 16-19 June 2002 Thessaloniki – Greece.

[Ngo3] T. Ngo-Quynh, H. Karl, A. Wolisz, K. Rebensburg, “Relative Jitter Packet Scheduling for Differentiated Services,” *In Proceeding of 9th IFIP Working Conference on Performance Modeling and Evaluation of ATM&IP Networks IFIP ATM&IP 2001*, Budapest Hungary, Juny, 2001.

[Ngo4] T. Ngo-Quynh, H. Karl, A. Wolisz, K. Rebensburg, "The Influence of Proportional Jitter and Delay on End-to-End Delay in Differentiated Services Network," *In Proceeding of IEEE International Symposium on Network Computing and Application NCA'01*, Cambridge, MA, USA. Februar 2002.

[Ngo5] T. Ngo-Quynh, H. Karl, A. Wolisz, K. Rebensburg, "Using only Proportional Jitter Scheduling at the boundary of a Differentiated Services Network: simple and efficient," *In Proceeding of 2nd European Conference on Universal Multiservice Networks ECUMN'02*, April 8-10, 2002,Colmar, France.

[Nguy01] H. T. Nguyen and Helmut Rzehak, "An Adaptive Bandwidth Scheduling for Throughput and Delay Differentiation," *In Proceeding of ICN'01*, July 11-13, 2001, Colmar, France.

[Nich98] K. Nichols, S. Blake, F. Baker and D. L. Black, "Definition of the Differentiated Services Field (DS Field) in the Ipv4 and Ipv6 Header," December 1998. IETF RFC 2474.

[NS2] The Network Simulator ns-2. http://www.isi.edu/nsnam/ns/

[Odl99] A. M. Odlyzko, "Paris Metro Pricing: The Minimalist Differentiated Services Solution," *In Proceedings IEEE/IFIP International Workshop on Quality of Service*, June 1999.

[Orion00] Orion Hodson, Colin Perkins and Vicky Hardman**, ''**Skew detection and compensation for Internet audio applications,'' *In Proceedings of the IEEE International Conference on Multimedia and Expo*, July 2000, New York.

[Par93] A. K. Parekh, Robert G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Intergrated Services Networks: The single node case," *IEEE/ACM Transactions on Networking*, vol 1, no 3(1993).

[Patr94] C. Patridge, Gigabit Networking, Addison-Wesley, 1994.

[Paxs97] V. Paxson, "End-to-End Internet Packet Dynamics," *SIGCOMM Symp. Commun. Architectures and Protocols*, Cannes, France, Sept. 1997.

[Paxs96] V. Paxson, "End-to-End Routing Behavior in the Internet," *In Proceedings SIGCOMM Symposium*, pp. 25-38, August 1996.

[Perk98] C. Perkins, O. Hodson, and V. Hardman, “A Survey of Packet Loss Recovery Techniques for Streaming Audio,” *IEEE Net*, vol. 12, no. 5, Sept. 1998, pp. 40-48.

[Raj99] R. Rajan, D. Verma, S. Kamat, E. Felstaine, and S. Herzog, “A Policy Framework for Intergrated and Differentiated Services in the Internet,” *IEEE Network*, pp. 36-41, September 1999.

[Ram94] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne, “Adaptive Playout Mechanisms for Packetized Audio Applicationis in wide-area Networks,” *In Proceedings of IEEE INFOCOM*, Toronto, Canada, pp. 680-686, June 1994.

[Robert] L. G. Roberts. Internet growth trends. http://www.ziplink.net/

[Rocc01] M. Roccetti et al., “Design and Experimental Evaluation of an AdaptivePplayout Delay Control Mechanism for Packetized Audio for Use over the Internet,” *Multi. Tools Apps*., vol. 14, no. 1, May 2001.

[Ros00] J. Rosenberg, L. Qiu, and H. Schulzrinne, “Integrating Packet FEC into Adaptive Voice Playout Buffer Algorithms on the Internet,” *In Proceeding IEEE INFOCOM*, Tel Aviv, Israel, Mar. 2000

[Roth95] K. Rothermel and T. Helbig, “An Adaptive Stream Synchronization Protocol,” *In Proceeding of NOSSDAV*, Durham, NH, Apr. 1995, Lect. Notes Comp. Sci., pp. 189-202.

[Sahu00] S. Sahu. P. Nain, D. Towsley, C. Diot and V. Firoiu, “On Achievable Service Differentiation with Token Bucket Marking for TCP,” *In Proceedings of ACM SIGMETRICS*, June 2000.

[Sahu99] J. S. Sahu, D. Towsley, “A Quantitative Study of Differentiated Services for the Internet,” *In Proceedings Global Internet Symposium*, December 1999.

[San93] H. Santoso et al., ‘’Preserving Temporal Signature: a Way to Convey Time Constrained Flows,’’ *In Proceeding of IEEE GLOBECOM*, Houston, TX, Nov. 1993.

[Shenk12] S. Shenker, C. Patridge and R. Guerin, “Specification of Guaranteed Quality of Service,” September 1997. Internet RFC 2212.

[Shenk15] S. Shenker and J. Wroclawski, "General Characterization Parameters for Intergrated Services Network Elements," September 1997. RFC 2215.

[Shree95] M. Shreedhar and G. Varghese, "Efficient Fair Queuing using Deficit Round Robin," *In Proceeding ACM SIGCOMM*, pp. 231-242, 1995.

[Shiv95] N. Shivakumar, C. J. Sreeman, B. Narendran and P. Agrawal, "The Concord Algorithm for Synchronization of Networked Multimedia Streams," *International Conference on Multimedia Computing and Systems*, 1995.

[Srin99] V. Srinivasan, S. Suri, and G. Varghese, "Packet Classification using Tulpe Space Search," *In Proceedings of ACM SIGCOMM'99*, pp. 135-146, Cambridge, MA, September 1999.

[Step99] D. C. Stephens, J. C. R. Bennett, and H. Zhang, "Implementing Scheduling Algorithms in high-speed Network," *IEEE Journal on Selected Areas of Communications*, September 1999

[Stil98] D. Stiliadis and A. Varma, "Latency Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms," *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 611-625, October 1998.

[Stoika1] I. Stoika, S. Shenker, and H. Zhang, "Core-stateless Fair Queuing: Achieving Approximately Fair Bandwidth Allocations in high speed Networks," *In Proceedings ACM SIGCOMM*, September 1998.

[Stoika2] I. Stoika and H. Zhang, "Exact Emulation of an Output Queuing switch by a Combined Input Output Queuing Switch," *In Proceeding of IWQoS'98*, pages 218-224, Napa, CA, 1998.

[Stoika3] I. Stoika and H. Zhang, "LIRA: An Approach for Service Differentiation in the Internet," *In Proceedings NOSSDAV*, 1998.

[Stoika4] I. Stoika and H. Zhang, "Providing Guaranteed Services without per flow Management," *In Proceedings of ACM SIGC*OMM, September 1999.

[Ston95] D. L. Stone and K. Jeffay, “An empirical Study of a Jitter Management Scheme for Video Teleconferencing,” Multi, Sys., vol. 2, no. 2, 1995.

[Suter98] B. Suter, T. Lakshman, D. Stiliadis, and A. Choudhury, “Design Considerations for supporting TCP with per-flow Queuing, “ *In Proceedings IEEE INFOCOM 1998*.

[Teit99] B. Teitelbaum, S. Hares, L. Dunn, R. Neilson, V. Narayan, and F. Reichmeyer, “Internet2Qbone: Building a Testbed for Differentiated Services, “ *IEEE Network*, pp. 8-16, September 1999.

[Terz99] A. Terzis, L. Wang, J. Ogawa, and L. Zhang, “A Two-Tier Resource Management Model for the Internet,” *In Proceedings Global Internet Symposium*, December 1999.

[Wang98] Z. Wang. A Case for Providing Fair Sharing. *In International Workshop on QoS*. May 1998.

[Wang01] B. Wang, S. Sen, M. Adler, and D. Towsley, “Proxy-based-Distribution of Streaming Video over Unicast/multicast Connections,” *Tech. Rep. UMASS TR-2001-05*, Univ. MA, Amherrst, 2001.

[Whit97] P. P. White, “RSVP and Intergrated Services in the Internet: a Tutorial”. *IEEE Communications Magazine*, Pages 100-106, May 1997.

[Wrocl11] J. Wroclawski, “Specification of the Controlled-load Network Element Service,” September 1997. RFC 2211.

[Wrocl10] J. Wroclawski, “The Use of RSVP with IETF Intergrated Services,” September 1997. RFC 2210.

[Yau97] D. K. Y. Yau and S. S. Lam, “Adaptive Rate Controlled Scheduling for Multimedia Applications,” *IEEE/ACM Transactions on Networking*, vol. 5, no. 4, pp. 475-488, August 1997.

[Yeom00] I. Yeom and Y. N. Reddy, “Modeling TCP Behavior in a Differentiated Services Network,” Technical Report, Texas A&M University, February 2000.

[Yua96] M. C. Yuang et al., “Dynamic Video Playout Smothing Method for Multimedia Applications,” *In Proceeding IEEE ICC*, Dallas, TX, June 1996, p. 544.

[Yua97] M. C. Yuang, P. L. Tien, and S. T. Liang, “Intelligent Video Smoother for Multimedia Communications,” *IEEE JSAC*, vol. 15, no. 2, Feb. 1997, pp. 136-46.

[Zhang1] H. Zhang, “Service Disciplines for Intergrated Services Packet-switching Networks,” *PhD thesis*, University of California at Berkeley, Computer Science Division, November 1993. Technical Report UCB/CSD-94-788.

[Zhang2] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, “RSVP: a new Resource Reservation Protocol,” *IEEE Network*, pp. 8-18, September 1993.