Resource Allocation and Adaptive Scheduling for Scalable Video Streaming

THÈSE N^{O} 4121 (2008)

PRÉSENTÉE LE 27 JUIN 2008

À LA FACULTE SCIENCES ET TECHNIQUES DE L'INGÉNIEUR

LABORATOIRE DE TRAITEMENT DES SIGNAUX 4

PROGRAMME DOCTORAL EN INFORMATIQUE, COMMUNICATIONS ET INFORMATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Jean-Paul WAGNER

ingénieur en systèmes de communication diplômé EPF
et de nationalité luxembourgeoise

acceptée sur proposition du jury:

Prof. M. A. Shokrollahi, président du jury
Prof. P. Frossard, directeur de thèse
Dr C. De Vleeschouwer, rapporteur
Prof. P. Thiran, rapporteur
Dr O. Verscheure, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2008

Acknowledgment

First, I would like to thank my PhD advisor, Prof. Pascal Frossard, for giving me the opportunity to work under his supervision. I have learned a lot from him and I am grateful for all of the long discussions we had on a wide range of topics.

I am thankful to the members of my thesis committee, Prof. Patrick Thiran, Dr. Christophe de Vleeschouwer, and Dr. Olivier Verscheure for the time spent in reviewing this manuscript, and for their helpful comments on improving its content. The work presented here has been funded through the Swiss National Science Foundation; I am grateful for this support.

I thank all my current and former colleagues at the LTS4 laboratory for their support and for providing an ejoyable and fun working environment. Special thanks go to Christophe de Vleeschouwer, Jakov Chakareski, Nikolaos Thomos and Dan Jurca for the interesting research discussions and collaborations during the past four years. I thank the staff of the Signal Processing Laboratory, in particular Marianne Marion and Gilles Auric for making the administrative work seem easy.

I am very grateful to my family, for their dedication and the support they have given me during all of my life.

Last but not least, I would like to thank Michelle for the love and support she has given me ever since we have met. I would like to dedicate this thesis to her.

Abstract

The obvious recent advances in areas such as video compression and network architectures allow for the deployment of novel video distribution applications. These have the potential to provide ubiquitous media access to end users. In recent years, applications based on audio and video streaming have turned out to be immensely popular and the Internet has become the most widely used vector for media content distribution, due to its high availability and connectivity. However, the nature of the Internet infrastructure is not adapted to the specific characteristics of multimedia traffic, which presents a certain tolerance to losses, but strict delay and high bandwidth requirements.

In this thesis, our goal is to improve the efficiency of media delivery over the existing network architecture. In order to do so we consider the delivery of scalable video in three main delivery scenarios, namely one-to-one client server architectures, one-to-many broadcasting architectures, and many-to-one distributed streaming architectures.

First, we propose a distributed media-friendly rate allocation algorithm for the delivery of both finely and coarsely scalable video streams. Unlike existing solutions, our algorithm explicitly takes the characteristics of media streams into consideration. As a result, it provides rate allocations that better fit the heterogeneous characteristics of media streams. We outline an implementation that is robust to random feedback delays and that permits a scalable deployment of the algorithm. The rate allocation that is computed by our algorithm achieves network stability and high bandwidth utilization. It moreover allows to maximize the average received quality for all streams that are delivered in the network. While considering the transmission of coarsely layered streams, we derive conditions on the encoding rates of the video layers. These conditions depend on the allowed end-to-end delay and on the rate allocation algorithm that controls the sending rates. They allow us to take full advantage of the allocated transmission rates.

Second, we investigate the problem of jointly addressing the needs of multiple receivers that consume different versions of a layered media stream in a broadcasting scenario. We provide optimal scheduling algorithms that jointly optimize the playback delay and the buffer occupancy at all of these receivers when the used channel is known. Furthermore we analyze low complexity heuristics based optimization techniques, which provide close to optimal results when only limited channel knowledge is available.

Finally, we explore the possibility to exploit the inherent network diversity that is provided by the Internet infrastructure. In particular, we consider media delivery schemes where multiple senders are available for the transmission of a scalable video stream to a single client. Such an architecture is referred to as

a distributed streaming architecture. It has the benefit of aggregating multiple unreliable channels into a single more robust channel with high availability. Through the use of Fountain codes, we are able to transform the distributed streaming problem into a rate allocation problem of lower complexity. The solution to this problem is shown to depend not only on the average packet loss rate, but also on the average length of packet loss bursts that are observed on each of the available channels. The coding scheme that we suggest enables our system to adapt the streamed content to the network characteristics, as well as to the needs of the receiving client.

Keywords: video streaming, distributed streaming, packet scheduling, rate allocation, congestion control.

Résumé

La disponibilité de réseaux de données, ainsi que les progrès indéniables en matière de compression vidéo, donnent lieu à de nouvelles architectures supportant des applications vidéo. Ces architectures ont le potentiel de rendre accessible des fichiers de médias en tout lieu. Au cours des dernières années, des applications basées sur les transmissions audio et vidéo ont connu un succès immense et le réseau internet est devenu le vecteur de choix pour les transmissions multimédiales. Cependant, l'infrastructure du réseau internet est mal adaptée aux besoins du traffic généré par des applications multimédiales. Celui-ci peut tolérer un certain degré de pertes, mais il présente des contraintes strictes en termes de délais et de bande passante.

Dans cette thèse, notre but est de présenter de nouveaux résultats qui permettent d'améliorer les systèmes de transmissions multimédiales en utilisant l'infrastructure du réseau disponible.

Nous allons adresser trois scénarios importants dans cette optique. Il s'agit des transmissions un-à-un, des transmissions un-à-plusieurs, ainsi que des systèmes de transmission distribués plusieurs-à-un.

Nous proposons un algorithme distribué d'allocation de bande passante. Contrairement à des solutions existantes, les allocations calculées par notre algorithme sont aptes à la transmission de vidéos scalables à différents degrés et permettent une maximization de la qualité vidéo moyenne reçue à travers le réseau entier. Il s'ensuit que ces allocations nous permettent de respecter les besoins hétérogènes des vidéos transmises. L'implémentation que nous proposons est robuste aux délais aléatoires qui peuvent se manifester dans le réseau, et permet de déployer l'algorithme à grande échelle. En analysant la transmission de vidéos scalables par couches, nous présentons des conditions sur le débit d'encodage des couches vidéos. Celles-ci dépendent à la fois du délai bout-à-bout que la vidéo peut subir et de l'algorithme d'allocation de bande passante utilisé pour contrôler la transmission. En respectant ces conditions, un avantage maximal peut être tiré des allocations de bande passante que nous proposons.

Ensuite, nous adressons des scénarios de type broadcast, dans lesquels chaque récepteur décode un sous-ensemble différent des couches vidéo transmises. Nous présentons des résultats optimaux en ce qui concerne la minimisation de la mémoire nécessaire au décodage pour chacun des récepteurs, ainsi qu'une optimisation jointe des délais qui s'imposent afin d'assurer un décodage continu quand le canal utilisé est connu.

Finalement, nous présentons une architecture distribuée pour transmettre des vidéos de plusieurs sources à un seul récepteur. Nous utilisons des codes sans rendement de type *fontaine*, ce qui nous permet de transformer ce problème distribué difficile en un problème d'allocation de bande passante à complexité

réduite. La solution de ce problème dépend à la fois du taux de perte moyen de chacun des canaux utilisés, et de la longueur moyenne de pertes séquentielles observées. Le codage que nous proposons facilite l'adaptation des transmissions aux conditions du réseau, et permet en même temps de satisfaire les besoins du client en termes de scalabilité.

Mots-clés: transmission vidéo, transmission de vidéo distribuée, allocation de bande passante, contrôle de congestion.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The advances of technology in the areas of video compression and today's networks permit the deployment of novel video distribution applications. These have the potential to provide the end users with ubiquitous access to media streams, such as audio and video content. In recent years such applications have turned out to be immensely popular. In this thesis we will focus on architectures that allow for video streaming over IP networks, and provide new results in order to improve their efficiency.

An overview picture of today's multimedia communication infrastructure is given in Figure 1.1. Media content is in general encoded or stored in servers that connect to the networking infrastructure through reliable high speed channels. The clients that connect to media streams tend to become more and more heterogeneous and can have a wide variety of access modes to the network. Indeed typical devices that are used today for the consumption of media streams can range from mobile phones with low processing power and small display sizes, up to high performance workstations that drive High Definition displays.

The IP networking infrastructure that connects the senders to the clients provides essentially a best effort service, hence it does in general not give any hard guarantees to the media streaming services in terms of either delay, available bandwidth, or even channel availability. However, there are in general multiple possible paths between any two points that can connect through the network. Also, there are often several servers available to provide the same data to a potential client, by using different channels. These features are generally referred to as *network diversity*.

The area of multimedia communications inherently lies at the crossing of both the signal processing and networking communities, hence efficient solutions to problems in multimedia communications ask for a thorough understanding of both the available networking architectures and the characteristics of encoded video signals.

In order to make sure that a high Quality of Service (QoS) can be provided to the end users, it is essential to efficiently use the available network resources and to carefully adapt the video content to both the network state and the needs of the various clients.

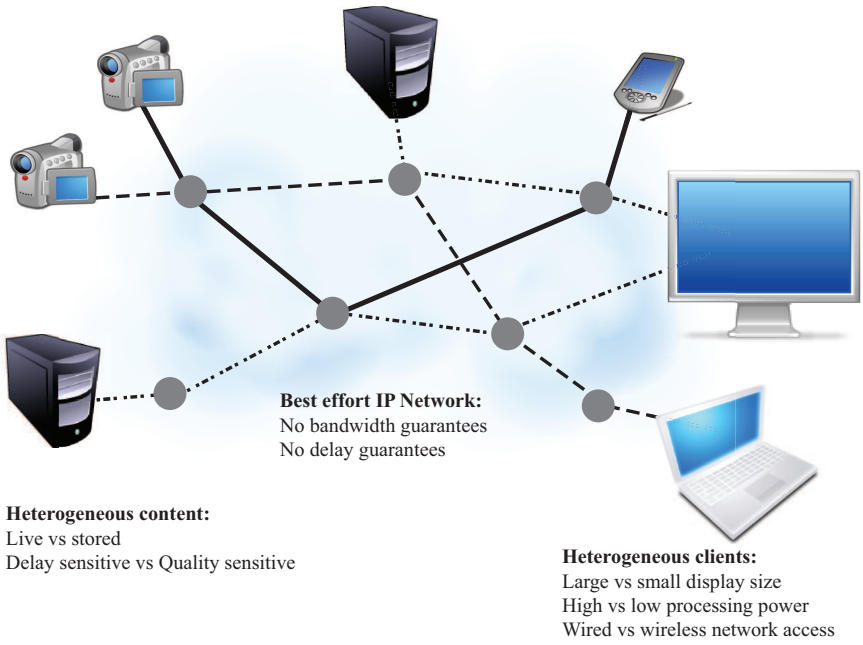

Figure 1.1: A multimedia communication infrastructure and its components.

1.2 Problem statement and contributions

The Internet is nowadays widely used for media content distribution, due to its high availability and connectivity. It is important to realize that the basic underlying network infrastructure of the Internet has been built up through the past decades, essentially for the transfer of data files and to provide basic services such as e-mail. The kind of traffic that is inherently supported is sensitive to losses, but not to delays. The traffic that is generated by media applications today is characterized by some degree of redundancy, which makes it robust to a certain degree of losses, by high transmission rates and generally by low delay tolerance. Although these characteristics are radically different from the initial design goals of the Internet, the underlying infrastructure that is used for media delivery is today essentially the same as that used 20 years ago for simple data transfers. In that sense, the networking infrastructure can be considered to be inherently *media-unfriendly*.

In this thesis, our goal is to improve the efficiency of media delivery over the existing networking architecture. In order to so, we have chosen to investigate three main directions.

The data transport protocols that are used for media transmissions in most of today's architectures, such as the Congestion Avoidance Protocol implemented in TCP [1], or TCP-friendly rate control (TFRC) [2], are inherently media-unfriendly. The goal of these algorithms is to fairly allocate the available network resources to each of the contending data flows. This is achieved without

considering temporal variations in the rate. However, as media streams need to be delivered at a relatively steady rate in order to achieve a good Quality of Service at the receivers, these inherent rate variations severely deteriorate the received media signals. Moreover, each media stream can make potentially different use of the allocated rate: while a stream encoded at a low resolution can be transmitted at the highest decodable quality using a relatively small channel rate, this is for example not true for High Definition motion-rich video content. Hence a *TCP-fair* and *equal* channel resource allocation between such heterogeneous streams is clearly not optimal.

We investigate the feasibility of **media-friendly rate allocation algorithms**. The distributed algorithm we propose and implement takes the specificities of media traffic explicitly into account in order to efficiently allocate the available network bandwidth. This approach results in a system that maximizes the average quality of all the delivered streams in the network, while keeping the latter stable.

In order to adapt the streamed content to the display or decoding capabilities of heterogeneous clients, multiple versions or scalability layers of the same video information need to be made available for streaming. We provide important conditions on the encoding rates of these layers, which need to be considered in order to take full advantage of the smooth rate allocation that is provided by our solution.

In broadcasting scenarios, heterogeneous clients that consume different versions or layers of the same stream have in general different constraints in terms of processing power or buffering capacity. That is why we investigate **scheduling algorithms** that are run at the broadcaster and that jointly consider delay and buffering minimization for multiple receivers of a scalable video stream.

Finally, we explore the possibility to exploit the inherent network diversity that is provided by the Internet infrastructure. In particular, we consider media delivery schemes where multiple senders are available for the transmission of a video stream to a single client. Such an architecture is referred to as a **distributed streaming architecture**. The solution we propose is based on the use of *rateless codes* and avoids coordination between the different senders. It has the benefit of aggregating multiple unreliable channels into a single more robust channel. Through the coding scheme we propose, we are able to make efficient use of such an aggregate channel at a low complexity cost. At the same time, the number of video layers that are delivered can be selected to fit the needs of the client.

The main contributions we provide in this thesis can be stated as follows:

- Based on a theoretical framework that has been initially introduced by Frank Kelly [3], we propose a fully media friendly rate and congestion control algorithm, which runs as a distributed algorithm. In contrast to previous efforts in the research community, our algorithm is robust to heterogeneous and random delays that may be experienced in practical network settings. It is completely scalable and robust while it is implementable using a light-weight application layer protocol. Through the use of utility functions that provide a realistic description of the scalable video streams, we are able to deliver appropriately adapted versions of the delivered streams to clients with different display or decoding capabilities.

- We consider the delivery of coarsely scalable video streams using the afore-

mentioned media-friendly rate allocation algorithm. To this aim we derive conditions on the actual encoding rates of video layers. These conditions depend on the characteristics of the network and on the used rate and congestion control algorithm. Most of the advanced video encoding standards that are available today provide some ways to encode video into several scalability layers. Hence our results are of high practical importance as they specify at which rates these layers should be encoded in order to take full advantage of the allocated network resources.

- Heterogeneous clients, whose difference is marked by different display sizes or processing and storage capabilities for example, ask for different scalability layers of a video stream to be delivered. On the one hand, we provide an optimal scheduling algorithm for broadcast scenarios, which jointly minimizes the buffering needs in a heterogeneous client population when the channel is known. On the other hand, we outline a practical scheme that adapts the computed schedule to potential variations in the channel bandwidth, while still providing close to optimal performance. We give optimal results on joint playback delay minimization for a set of heterogeneous clients that consume various levels of layered video streams. Moreover we outline efficient computing techniques that allow to compute results that are close to optimal when only limited knowledge about the channel is available.

- We provide a low-complexity solution to the difficult problem of distributed streaming. Contrarily to traditional approaches, which use distributed scheduling algorithms running across all participating senders in order to optimally convey video information to a streaming client, we propose to completely decouple senders. This is achieved by encoding the video information on each server using rateless codes. Using this family of codes, we are able to adapt on the fly to potential channel losses. While our scheme avoids us to keep the senders in synchrony, the coding scheme we propose further enables us to adapt the number of layers that are transmitted to both the network availability and the client's needs.

1.3 Thesis Outline

The outline of this thesis is as follows.

In Chapter 2 we will provide an overview of related work in the areas we are considering throughout this thesis.

A distributed media-friendly rate allocation algorithm is presented in Chapter 3. While we use finely scalable video streams to introduce our rate allocation algorithm, we later analyze under which conditions on the encoded layer rates our algorithm can be used to efficiently deliver coarse layered video streams. This is addressed in Chapter 4.

We consider a broadcasting scenario in which a layered scalable video stream is delivered to a population of heterogeneous clients in Chapter 5. Optimal results on playback delay optimization and joint receiver buffer minimization at all the clients are proposed, as well as practical sub-optimal algorithms that provide good results at a lowered complexity cost.

A framework for low-complexity robust distributed streaming from multiple sources to a single client is presented in Chapter 6.

Finally, we conclude and give and outlook on future work in Chapter 7.

Chapter 2

State of the Art

2.1 Background

The availability of high speed computing devices and the expansion of broadband links for Internet access foster a growing demand for multimedia communication services such as video on demand, video conferencing or live video streaming. Examples of this growing demand can be witnessed by the popularity of online services such as YouTube, Joost or Apple iTunes Movie Rental for example.

The main problem that needs to be addressed in order to provide ubiquitous access to media streams at a high Quality of Service (QoS) lies in adapting the data rates of the encoded media streams to the channel rates that are available for their transportation, or vice-versa. The network infrastructure that is used today for media delivery provides essentially *best effort* service, yields highly unpredictable available channel rates, and results in random packet delays and losses. On the other hand, a video stream that is decodable at a high QoS can be characterized by a relatively constant average data rate which can exhibit significant burstiness on shorter timescales. Moreover, as a video that is played out continuously imposes strict timing constraints on the received data, packets that arrive too late are essentially useless.

In this chapter we will give an overview of relevant work with respect to the problems that are addressed in the remainder of this thesis. We will have a look at efforts that consider to exploit the characteristics of current network architectures in order to increase the perceived QoS of transmitted media streams. Then we will present prior work that aims at adapting the available channel rates to the characteristics of the media streams that are to be transported. This is typically done by changing the way data is transmitted at the Transport layer. Next we will provide references on video encoding techniques, before we conclude the chapter with prior work on media packet scheduling algorithms that aim at adapting the transmitted rate to the available channel rate under QoS constraints.

2.2 Networking perspective

One of the main observations regarding channels that connect different end points in today's IP networks lies in the fact that the service they provide is *best effort*. The network does not provide guarantees on the timeliness of received packets, on the rate at which packets are delivered, and even on whether packets are received at all.

2.2.1 Leveraging network diversity

Packets that are transmitted to a single client from different starting points (servers) located at different locations in the network, are very likely to experience uncorrelated channel effects, as the used channels tend to be independent. Hence the idea of leveraging this network diversity by aggregating multiple unreliable channels into a single more reliable one has been proposed in the framework of *distributed video streaming*. Early work in this area is proposed in [4, 5] where the authors propose distributed scheduling algorithms, and in [6], in which a distributed delivery scheme based on distributed Forward Error Coding (FEC) is proposed. These efforts have notably shown that the quality of received media streams can be significantly enhanced when transmitting from several servers to a client compared to traditional server-client models.

Recently, highly scalable data distribution paradigms such as Peer-to-Peer (P2P) overlays have emerged. Although they are mainly used for file sharing and distribution [7] purposes, as in the BitTorrent [8] system for example, these paradigms have sparked a renewed interest in distributed streaming [9]. Several P2P streaming architectures have been proposed. For example, the work in [10] outlines a dynamic peer selection scheme that allows to select an appropriate set of peers for transmission. In [11], a bandwidth adaptation protocol that increases the transmission robustness in P2P streaming scenarios is discussed. The problem of where to replicate the available video streams in a P2P network in order to ensure high content availability is addressed in [12], while the authors in [13] focus on quality adaptive delivery of media streams in such architectures. A comprehensive overview of outstanding issues in this area is provided in [14].

Other distributed architectures comprise for example hybrid client-server and P2P models such as cooperative networking [15]. While the main streaming task is attributed to a single server, clients that have received the media stream earlier on are able to help alleviate the server load by participating in the distribution process. This is particularly useful in the case of severe congestions on the server-client link.

In the case of active networks [16], passive routers that merely forward incoming packets to the next hop are replaced by nodes that can implement several more advanced tasks. Following this paradigm, network diversity can for example be leveraged at an even finer scale. Each active node in the network can be enabled to select a specific next hop for each forwarded media packet according to its relative importance in the decoding process. This way each packet can travel on the path that maximizes the probability for it to reach its destination and to contribute best to the perceived QoS at the end point. An overview on the benefits of using path diversity for video delivery is given in [17]. An efficient implementation of such a transmission scheme that uses Multiple Description Coding (MDC) has been proposed in [18], while the authors in [19]

analyze under which conditions it is optimal for a path diversity scheme to use MDC encoded versus layered encoded media.

The distributed streaming framework we propose in Chapter 6 has the benefit of being less complex than the previous efforts that consider distributed scheduling. Along the lines of [6], we use error correcting codes to alleviate the scheduling problem. In contrast to prior work, we use rateless codes and apply them independently to each of the video layers in the transmitted stream. This approach results in a distributed framework that has the advantage of being able to adapt the transmitted media stream on the fly and without transcoding to potential channel losses. At the same time, the scalability layers that are transmitted are adapted to the needs of the receiving client.

2.2.2 Rate and Congestion Control

Rates that are available for data transmission by a given application and that provide stable network usage are typically computed by Rate and Congestion Control algorithms (RCC). These are in general agnostic of the rate or delay constraints that an application may impose. We will give an overview of recent research efforts that try to remedy to this situation by implementing *media-friendliness* at the level of the RCC algorithm that is used.

Each sender-receiver pair in a network can be characterized by a data flow that it generates. This data flow has a certain data rate and it uses a subset of network resources (typically routers) with heterogeneous capacities, in order to reach its destination. The routing of each packet that constitutes the data flow is done per packet and per hop in a completely distributed way [20], which implies that the network routers may be driven to operate at their maximum workload capacity. If a router reaches this maximum capacity, i.e., it receives more packets than it can possibly forward, then it discards the surplus of packets and forwards only a subset of the packets that it has received. This situation can only be alleviated if the sources that transmit through this particular router adapt their sending rates accordingly. This is only possible if the sources know the actual state of the network resource. The algorithm that adjusts the rate of an application data flow and that, at the same time, tries to resolve such network congestions is called the Rate and Congestion Control algorithm. It controls the rate of a flow based on previously taken decisions and on the state of the network, which has to be fed back to the source. As the routing is performed on a per-hop basis and in general routers drop packets without notifying their respective senders, the feedback is generally generated at the end point of the flow, which notices that it did not receive all packets that were expected. This scenario is called RCC with end-to-end feedback. If routers are *active* elements in the network [16], they can also generate the feedback themselves and send it immediately to the sources that are concerned. Thus we define an RCC Algorithm [21] as a distributed algorithm that takes local rate control decisions based on network feedback and a history of previously taken controls, such that the state of the network globally converges to a stable state which is characterized by high network resource usage and bounded per flow losses.

Virtually all Internet connections are regulated by the TCP [1] algorithm which is implemented at the Transport layer. It operates using end-to-end feedback and manages to share available network bandwidth in a fair way between the flows that compete for them. It makes sure that reliable, lossless communi-

cation is achieved between a sender and a client, at the expense of possibly large delays. Due to the additive increase - multiplicative decrease (AIMD) nature of the algorithm and its retransmission policy, it bears however some serious drawbacks in media streaming scenarios. Indeed, whenever a congestion is detected, the available rate is halved resulting in a saw-tooth like rate profile. After a random backoff time, the rate is gradually increased until another congestion is detected. Packets that have been lost in a congestion are retransmitted at a later time instant, implying potential delivery delays. As TCP has been shown to perform poorly in high-speed networks, new algorithms such as XCP [22, 23] have been proposed. They rely mainly on Early Congestion Notifications (ECN) [24] generated at the congested routers. These congestion signals are more precise than end-to-end measurements and can be sent to the senders that caused the congestions earlier. As TCP, these algorithms fail to allocate rates that can be mapped to the rate needs of media streams.

The media communications community has thus focused on implementing RCC algorithms that provide steadier allocated rates than TCP, sacrificing retransmissions of lost packets by gaining in lower delays of the transmitted streams. These algorithms regulate UDP packet flows and are generally designed to compete in a TCP-friendly way along TCP regulated flows for the available network resources. In TFRC [2], this is for example achieved by allocating rates that provide the same average bandwidth as an equivalent TCP flow would use, while providing a smoother rate profile. Although this is an interesting property for media streaming applications, the RCC algorithm still fails in capturing the intrinsic properties of the media streams. TFRC for JPEG2000 video streaming is for example considered in [25] and highlights this problem. The authors truncate the frames in such a way as to match the smooth rate allocated by TFRC. However, TFRC crucially fails to allocate rates that would inherently allow for constant quality video transmission.

A new direction in the design of RCC Algorithms has been started by the seminal paper on the concept of Network Utility Maximization by Frank Kelly [3]. Kelly shows that a stable network state that can provide various definitions of *fairness* among contending flows, can be reached through the use of a simple distributed algorithm. To this end, each flow can be characterized by a function that defines its *Utility* at each transmitted rate. The stable state resource allocation, which is reached through a smooth rate profile, maximizes the aggregate Utility for all flows.

Research in this field has essentially dealt with practical implications for generic data transmissions in end-to-end feedback settings [26, 27] and stability or fairness considerations of the resulting algorithms [28, 29, 30]. But Kelly-controlled systems relying on active network components have also been investigated, for example in the JetMax algorithm outlined in [31].

As the properties of Kelly-controlled flows are closer to the requirements of media streams than those of flows generated by previously proposed RCC algorithms, there has been some recent work on rendering this class of algorithms *media-friendly*. This can be achieved by relating Kelly's concept of Utility specifically to the rate needs of the media streams. In [32, 33, 34] for example, the authors consider the transmission of Fine Granularity Scalable (FGS) video streams. It should be noted that in the proposed systems, correct round-trip time measurements are generally needed in order to make sure that the network remains stable. Having access to exact timing information from the lower

layers in the networking stack asks for added complexity by using cross-layer approaches.

The algorithm we propose in Chapter 3 specifically eliminates the need for exact timing information and is proven to be stable. We show notably that the algorithm can be implemented in a scalable way using a light-weight application layer protocol. Moreover, we show in Chapter 4 that the proposed RCC algorithm can also be used to transmit coarse layered scalable video. This case has not been addressed in previous work.

2.3 Video Encoding Techniques

In this section we provide a brief overview of video encoding standards, while highlighting the capabilities of the media streams they produce in terms of rate adaptation. Most research efforts in the video encoding community have traditionally been targeting efficient data compression.

The recent standards, such as MPEG-4 Part 10 or H.264 AVC [35, 36] all follow the same general compression scheme: first, temporal correlation between neighboring frames is exploited by a prediction loop, then the residual information is decomposed using a transform such as DCT or wavelets. The resulting coefficients are finally quantized and compressed. In general, the prediction loop is reset after a rather small number of frames. This results in a bitstream that exhibits a bursty rate: the first frame in the prediction loop, called key frame, is encoded as a single still picture yielding a relatively high rate. The following frames are then encoded at a much lower rate as the prediction loop exploits the correlation between them and the last encoded key frame. Scalability features have been introduced into the modern encoders. These permit applications to adapt the video rate of the stream that is to be delivered to the available channel rate without re-encoding or transcoding the video information. In general, the more flexibility a scalable encoding provides, the worse is its compression performance.

We will consider three classes of scalable encodings. The first one being Multiple Description Encoding (MDC), which consists in generating a number of low quality representations of a video stream, that can be combined at will to improve the decoded quality. A simple method of generating such representations is provided in [37].

The second, and most widely used class of scalable encodings, is that of hierarchically encoded scalable streams. Video information is encoded into a hierarchical set of layers. Adding a layer can result in higher resolution, higher framerate or higher quality for example. The latest incarnation of layered encoding is given by the scalable video coding (SVC) extension to the H.264 standard [38]. The application of SVC encodings to wireless communication scenarios, where channel conditions are prone to change rapidly, has been advocated in [39]. As an alternative to layered streams, the SVC standard also includes the possibility to encode a stream into multiple versions, providing different scalability features for each version, and to enable an application to switch from one version to another at predefined points in the bitstream [40]. Much finer rate adaptation capabilities are provided by Fine Granularity Scalability (FGS) encoders. They are able to generate bitstreams that can be truncated at any byte level. In this case, an application can obviously match the available channel

rate exactly by truncating the FGS stream to the needed level. An overview of FGS encoding is provided in [41].

2.4 Adaptive Streaming

In this section we outline adaptive streaming techniques, which can in general be deployed either at the sender, or in a network node, such as a proxy server for example. A sender would adapt the rate of an encoded video stream to the available channel rate that the RCC has decided, while a proxy would for example adapt the stream to its outgoing rate.

In order to cope with the characteristic short timescale burstiness of the encoded streams, rate shaping and smoothing techniques have been proposed in [42, 43] for node-to-node scenarios. Similar ideas are presented in [44, 45, 46]. A deterministic Network Calculus framework that can be used to characterize data flows, service policies and network elements such as shapers and smoothers is introduced in [47]. It has been applied to obtain optimal multimedia smoothing and to characterize the minimum playout delay of a single layer stream in [48, 49] and [50], where the authors propose a joint source rate selection and smoothing technique for optimized transmission of media streams.

On a larger timescale, there is a need to adapt the video stream that is transmitted to the available channel rate. Doing so permits to take advantage of a high available rate and transmit the video stream at a higher quality, or to gracefully degrade the transmitted quality when the available rate should drop. The work in [51] provides an interesting early work that exploits the availability of video layers by transmitting them over different multicast groups. Depending on the available rate, a client can then subscribe to a subset of multicast groups in order to receive the corresponding subset of layers. A similar idea has been published in [52]. Transmission policies for layered scalable streams are further discussed in [53], while [54] discusses layered coarse grain adaptivity in the context of TCP friendly rate control. A study of perceived quality variations when adding or dropping layers of a hierarchically encoded stream is presented in [55, 56, 57].

As a result of the scalable encoding process, not all video packets that are available for transmission are equally important in the decoding process. By taking these relative weights of each packet into account, the senders can select to transmit an optimal set of packets, given the imposed bandwidth constraints. Transmission policies that optimize the quality of MPEG-4 FGS video streams in the case of lossy transmission channels and error concealment capabilities at the receiver are extensively studied by the authors of both [58] and [59].

In the Rate-Distortion (RaDiO) scheduling framework [60, 61], video streams are adapted by selecting a subset of packets based on rate constraints, delay constraints and the relative importance of each packet to the overall received quality. Optimal packet scheduling in the case of Multiple Description Coding is discussed in [62].

None of the cited papers addresses the problem of adaptively scheduling a layered stream in a broadcast scenario, where each of the receivers can decode a different subset of the available layers. In Chapter 5 we provide optimal transmission strategies for these scenarios, which have the benefit of jointly minimizing both the playback delays and the buffering capacities at each of the

receivers.

Chapter 3

Media-Friendly Distributed Rate Allocation

3.1 Motivation

In this chapter we address the problem of distributed congestion control in networks where several video streaming sessions share joint bottleneck channels. The proposed algorithm takes into account the specificities of the video streams, as well as the requirements of the heterogeneous streaming clients, in order to determine the actual benefit of additional channel resources, or equivalently the utility of rate increments. The utility-based congestion control framework initially proposed by Kelly [63, 3] is extended to cope with heterogeneous delays in the network as well as with the specific requirements of video streams. We describe an original implementation of such a distributed congestion control algorithm for scalable video streams, using the common RTP/UDP/IP protocol stack, where receiver feedback triggers the adaptation of the streaming rate at each server independently. Finally, we provide extensive simulation results that demonstrate the performance of the proposed solution, which successfully distributes the network resources among the different sessions in order to maximize the average video quality. The proposed scheme is also shown to cohabit fairly with TCP, which is certainly an important advantage in today's network architectures.

The predominance of FTP or HTTP traffic and the scalability of the Internet rely largely on the success of the transport protocol that controls the vast majority of the Internet connections, the Transport Control Protocol (TCP) [1]. One of the design goals of the TCP algorithm is to fairly distribute the bandwidth resources among the different concurrent flows. TCP provides a very efficient and distributed congestion control solution for low-bandwidth data streams, or elastic flows that do not present strict delay constraints.

While these types of data have formed the bulk of the Internet traffic, the Internet is evolving into a transport medium for the distribution of data such as audio and video streams due to the emergence of attractive multimedia applications. Multimedia streaming applications impose different requirements than those underlying FTP or HTTP traffic. On the one hand, the media streams have to be delivered at a rather high and sustained rate in order to deliver an

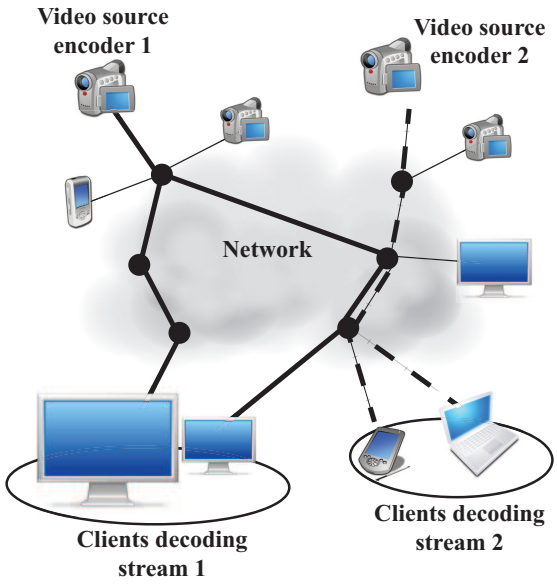

Figure 3.1: System view. The goal is to deliver the best possible media quality to the set of clients, while keeping the network in a stable state in dynamic scenarios.

acceptable Quality of Service (QoS) to the media client. On the other hand, an encoded media stream generally carries significant redundancy and therefore presents some inherent tolerance to packet loss. The experience of the end user is mostly driven by the delay and the fluctuations of quality in media streaming applications. Unsurprisingly, TCP does not perform very well in regulating the rate of media streams since it has not been planned for controlling such data flows. In particular, the typical sawtooth behavior in the controlled transmission rate profiles and the delays experienced by some packets due to the retransmission and exponential back-off policies are not ideal for media streaming. In addition, the importance of the media packets is quite heterogeneous between the different media streams, and even within a given media stream. TCP typically cannot consider these properties and rather targets a fair bandwidth distribution, which does not guarantee optimized average performance in multimedia communications.

In this chapter we will explore an alternative rate- and congestion control algorithm for a bandwidth resources allocation that is adapted to the characteristics of media streams. Instead of compensating the drawbacks of TCP through rather complex scheduling and retransmission schemes, we consider an orthogonal approach and propose a *media-friendly* control algorithm, which adapts the rate in the network to the needs of the media streams. Our work is based on a class of congestion control algorithms that have been first proposed by Kelly [63, 3]. Their general objective is to maximize the aggregate *utilities* that the end users retrieve from their network usage. Given this objective, a candidate algorithm should drive the system close to its optimal state, in which the network utilization is high and where each client receives a rate that is useful for the

media streaming application. The algorithm has also to be run in a distributed way in order to guarantee the scalability of the system. The family of congestion control algorithms described in [63, 3] has the benefit of i) potentially allocating smooth rate profiles without abrupt transitions along time and ii) distinguishing each flow based on a *utility* criterion. This criterion can typically capture the benefit of an increment of bandwidth for a given media streaming client. In the framework of this chapter, it depends on the proper characteristics of a media stream and on the particular requirements of the application.

We consider scenarios described by the framework illustrated in Figure 3.1. The media sources are captured and encoded in various points at the edges of an overlay network. Clients with heterogeneous needs in terms of frame rate or display resolution capabilities for example can connect to any media source from anywhere in the network. Such a scenario could typically represent a video-surveillance system with both high-end decoding stations and low power mobile clients. The objective of the distributed congestion control algorithm presented in this chapter is to jointly achieve network stability and optimized average quality for all decoding clients. In addition, the algorithm has to dynamically adapt to changes in the topology in order to accommodate new clients, or rather to efficiently use increased bandwidth resources.

We extend the class of Kelly-controlled algorithms to practical systems with heterogeneous feedback delays, where we show that the algorithms remain stable. Then we compute utility curves for scalable video streams, which permit to define several levels of quality by changing the spatial, temporal or SNR resolutions. We design a client feedback mechanism in order to infer the network state. Finally, we implement the distributed congestion control scheme for various streaming scenarios, where H.264/SVC streams are transmitted using RTP/UDP/IP protocols. We provide extensive simulation results that demonstrate the convergence of the distributed congestion control algorithm, even in dynamic environments. The results also show that the bandwidth is properly distributed among the clients such that the average quality is optimized. Finally, the proposed algorithm is shown to cohabit fairly with TCP flows.

The remainder of this chapter is organized as follows. In Section 3.2 we provide some background on Kelly's theoretical framework. We extend it in Section 3.3 with an important stability proof. In Section 3.4 we outline the proposed distributed congestion control algorithm and show how we can incorporate media-friendliness through a judicious choice of Utility functions. In Section 3.5 we describe the careful and fully scalable implementation of the proposed algorithm that can significantly boost the performance of the congestion control in practical scenarios. In Section 3.6 we validate our findings through extensive NS-2 simulations. Finally we conclude with Section 3.7.

3.2 Background

In this section, we provide some background and references on Kelly's Network Utility Maximization (NUM) problem that has been first stated in [3]. We give a brief overview on distributed algorithms that solve the aforementioned problem. Finally, we discuss the *fairness* between flows resulting from the rate allocations computed in the NUM framework, and give a brief overview of existing stability results.

3.2.1 Network Utility Maximization

Let $s_i(t)$ be the sending rate assigned to flow i at time t. Further, assume that we assign to each flow i a *utility function*. This function describes by a utility value $U_i(s_i(t))$ the usefulness of the streaming rate $s_i(t)$ for an application that is served by flow i [27]. We suppose that the utility functions are continuous functions of the rate and that they are concave. Let $\mathcal{I}$ be the set of all flows in a given network, characterized in turn by a set of network links l with their respective capacity constraints.

The original NUM problem consists in finding for every time t a set of rates $s_i(t)$ that maximizes the sum of utilities for all flows, $\sum_{i\in\mathcal{I}} U_i(s_i(t))$, while satisfying the rate constraints on each link in the network. The direct solution of this optimization problem is difficult to find for large networks, and it further assumes that a central entity controls the rate of each flow. Instead, Kelly has proposed in [3] to solve a relaxed version of the original problem, by adjusting the rate of the flows in order to satisfy the following differential equations at any time t:

$$\frac{d}{dt}s_i(t) = \kappa_i s_i(t)\left(U_i'(s_i(t)) - p_i(t)\right). \tag{3.1}$$

The term $s_i(t)\cdot U_i'(s_i(t))$ is referred to as the *willingness to pay* for the resource (i.e., the available bandwidth) and $p_i(t)$ is a pricing function, updated by the network, which indicates the price of the offered resource. In practice, $p_i(t)$ is often a congestion indication function that is fed back to the controller from either the network or the end user and κ_i is a constant gain factor.

Note that this system of differential equations drives each rate $s_i(t)$ into a steady state in which there is an equilibrium between the willingness to pay, and the flow's contribution to the resource's price. The former depends on the used utility function $U_i(s_i(t))$ through its gradient, while the latter depends on the current network state, which is expressed by the congestion indication function $p_i(t)$. It is worth noting that Equation (3.1) can be straightforwardly discretized and turned into a rate update equation that leads to a practical implementation of a distributed control algorithm for solving the original NUM problem. To date, this has yielded the most promising implementations of control algorithms, even if there are other ways of solving the original NUM problem in a distributed way [29].

3.2.2 Fairness

The control algorithms based on Kelly's framework provide a fair distribution of the network resources among the different competing flows in the network. The fairness characteristic might however be defined in several different ways. In particular, the fairness could be linked to the distribution of the network bandwidth, or rather to the distribution of network resources that balance utilities among flows. We refer to [64, 28] for a detailed study on the fairness in Kelly's framework. For the sake of completeness we provide a short summary here:

- If the controller relies on end-to-end feedback measures only, the control algorithm provides *proportional fairness*, meaning that flows congesting multiple routers are allocated less bandwidth in the stable state than flows congesting less routers.

- If the controller has access to the congestion levels of each router, which is however not a realistic scenario in today's Internet, the controller can adjust the rate for each flow according to the most congested router, thus providing *max-min fairness* [65].
- If additionally each flow is characterized by a potentially different utility function, the resulting rate allocations are weighted by the respective utility values in the stable state.

The practical system that we propose in this chapter relies on end-to-end measures and uses different utility functions for each flow. Hence, the control algorithm based on Kelly's framework provides in this case a *utility-proportional* fairness with the streaming rates allocated to the different flows.

3.2.3 Stability results

An important characteristic of any distributed control algorithm lies in the stability of the system. Both stability and convergence have been extensively studied for systems based on the differential equations given in (3.1) (see for example [3, 26, 21, 24] and references therein). However, the ideal system described above has only marginal practical relevance since it assumes that network or client feedbacks are immediately available at each source in order to update the rate of the corresponding flow. In practical systems, each flow i has a fixed starting point (the sender) as well as a fixed end point (the receiver) in the network. However, the route that connects the two may change in time, which therefore affects the round-trip delay between sender and receiver. Moreover, the congestion level of each router that is traversed by the flow is dynamically changing. The delay experienced by feedback information is not only heterogeneous across flows in the network, but it has also a random distribution for each flow. When the system experiences some delays in the feedback loops, the differential equations of relation (3.1) read as:

$$\frac{d}{dt}s_i(t) = \kappa_i \left(s_i(t)U_i'(s_i(t)) - s_i(t - D_i^R)p_i(t - D_i^B) \right), \tag{3.2}$$

In this case, D_i^R is the round-trip delay and D_i^B is the delay on the back-trip from the point in the network that created the feedback to the sender. Note that the congestion indication function $p_i(t - D_i^B)$ is synthesized D_i^B time units earlier, and that it relates to the rate allocated by the controller D_i^R time units earlier. In the last years, a substantial amount of research has been devoted to providing stability results for controllers with delayed feedbacks. For example the authors in [26] provide results for the case of equal round-trip delays for each flow. More recently, authors in [66] have studied the stability of systems with arbitrary but constant round-trip delays. Further stability results are provided in [67, 68, 69, 70, 71, 65] and references therein. In each of these works, the scenario corresponds to particular applications: either each flow uses the same utility function, or the delays are different for the various flows but constant in time, or the parameters of the rate update equation are dynamically adjusted with respect to the experienced delay. The latter implies that a precise measurement of the experienced round-trip delay is available at the controller, so that oscillations can be avoided in the system. The application considered

in this chapter however does not correspond to any of these scenarios, and we therefore extend the stability proof to a more generic case in the next section.

3.3 Stability with random feedback delays

The application we are targeting specifically calls for stability results for the scenario in which *general* concave utility functions (i.e. they are different for each media stream) are used, and in which the round-trip delays are arbitrary. In [72], the author provides an elegant stability proof for the case where general utility functions are used, and where the round-trip delays in the network are arbitrary across flows, but constant in time. The author conjectures that the system remains stable if the round-trip delays take on arbitrary values. In what follows we borrow the framework from [72] and we extend the stability result to the case where generic, concave utility functions are used and where the feedback is arbitrarily delayed for each flow.

We consider a network made up of L links, indexed by l. We call $\Delta_{l,i}$ the Round-Trip delay experienced by data transmitted from source i to traverse link l and get back to the source. Hence the experienced Round-Trip delay D_i^R for user i, takes on values in the set $\{\Delta_{1,i}, \ldots, \Delta_{L,i}\}$, depending on the receiver of flow i. Further we denote by Δ_l the maximum Round-Trip delay for data to get sent from any source, traverse link l and get back to the source: $\Delta_l = \sup_i\{\Delta_{l,i}\}$. Let $\overline{D}$ be an upper bound on all experienced Round-Trip delays in the network: $\overline{D} = \sup_i\{D_i^R\}$. Clearly, the following relation holds:

$$\overline{D} \geq \Delta_l, \;\; 0 \leq l \leq L. \tag{3.3}$$

The following Lemma is proven in [72] and is reproduced here as it is of crucial importance in the reasoning that follows.

Lemma 1. *Let A and B be two matrices, the entries of which, indexed by couples (n, m), all satisfy*

$$|A_{n,m}| \leq B_{n,m} \tag{3.4}$$

and hence, the $B_{n,m}$ are real, nonnegative. Then, the spectral radius, i.e., the largest positive eigenvalue of A, is smaller or equal to that of B.

The main result of [72] states that the system of delayed differential equations (3.2) is asymptotically stable if $\kappa_i > 0$, $U_i(\cdot)$ is a concave function for each i, and if all the Round-Trip delays in the network coincide with a single scalar: $D_i^R = D, \forall i$. Specifically, the author presents a linearization of the differential system (3.2) in the form of a delay-differential system of the retarded type:

$$\dot{y}_i(t) = -\sum_l \sum_s M_{is}^{(l)} y_s \left(t - \Delta_{\overleftarrow{l,i}} - \Delta_{\overrightarrow{l,s}}\right). \tag{3.5}$$

Where $M^{(l)}$ is a square matrix that depends on the load of router l, the Utility function that is used and the gain factor κ_i. $\Delta_{\overrightarrow{l,i}}$ is the delay from sender i to router l and $\Delta_{\overleftarrow{l,i}}$ is the delay on the corresponding back-trip. Furthermore, $y_i(t)$ is defined as:

$$y_i(t) = \frac{s_i(t) - \overline{s}_i}{\sqrt{\kappa_i \overline{s}_i}}. \tag{3.6}$$

It is a function of the difference between the actual sending rate $s_i(t)$ of sender i, and the steady-state sending rate that will be achieved at the equilibrium, denoted as $\overline{s}_i$. A sufficient condition for this system to be stable is that the spectral radius of matrix N, given as

$$N = D \sum_{0 \leq l \leq L} M^{(l)} \tag{3.7}$$

is smaller than 1. This is shown to be the case *for any scalar* D, if the matrix $M = \sum M^{(l)}$, which does not depend on the Round-Trip delays, is positive definite. This condition is in turn verified if $\kappa_i > 0$ and the utility functions $U_i(\cdot)$ are concave. We now extend this stability result to arbitrary feedback delays.

Theorem 1. *Assume that the Round-Trip delays D_i^R take on arbitrary values bounded by the scalar $\overline{D}$. Then the system given in Eq. (3.2) is asymptotically stable under the assumptions that $\kappa_i > 0$ and the Utility functions $U_i(\cdot)$ are concave for all i.*

Proof. Following the same reasoning as in [72], a sufficient condition for this to be true is that the corresponding delay-differential system describing the linearization of the control system around the equilibrium is asymptotically stable. This in turn is the case if the spectral radius of matrix N' is smaller than 1, with

$$N' = \sum_{0 \leq l \leq L} \Delta_l M^{(l)}, \tag{3.8}$$

and M the same as in the previous development. Let us introduce the following matrix:

$$\overline{N} = \overline{D} \sum_{0 \leq l \leq L} M^{(l)}. \tag{3.9}$$

From Eq. (3.7) we know that the spectral radius of $\overline{N}$ is less than 1. Using Eq. (3.3) we can further bound the elements of matrix N' as follows:

$$N' \leq \sum_{0 \leq l \leq L} \overline{D} M^{(l)} \tag{3.10}$$

As the spectral radius of the right-hand side of (3.10) is smaller than 1, the proof of this theorem is concluded by the application of Lemma 1. □

Although the system is stable for arbitrarily large but bounded feedback delays, the convergence rate is in general slower for increasing $\overline{D}$. This stems from the fact that the information about the network state takes longer to be integrated in the control loop. This result of stability is important for the design of the congestion control algorithm proposed in the next section. It is moreover most relevant to any practical system as in practice the Round-Trip delays that are observed are always bounded.

3.4 Distributed Rate Allocation Algorithm

We now propose a distributed rate allocation algorithm for video sequences, based on the utility maximization framework described above. We first present

a discretized version of the system of differential equations given in the relation (3.2), which leads to a discrete-time rate update equation. Then we propose utility functions that are adapted to practical scenarios for video streaming.

3.4.1 Rate update equation

The distributed control algorithm can only act at discrete time instants in practice. An approach based on finite differences can be used to obtain a discrete-time version of Eq. (3.2), as suggested in [26]. This results in the following update equation:

$$s_i(t) = s_i(t-1) + \kappa_i\big(s_i(t-1)U_i'(s_i(t-1)) - s_i(t-D_i^R)p_i(t-D_i^B)\big). \quad (3.11)$$

The congestion indication function (pricing function) for flow i can be computed based on the *received* rate $r_i(t-D_i^B)$ measured at the client at time $t-D_i^B$. In this case, it reads

$$p_i(t-D_i^B) = \frac{s_i(t-D_i^R) - r_i(t-D_i^B)}{s_i(t-D_i^R)}. \quad (3.12)$$

Note that there is a temporal shift between the time at which the reference sending rate, the actual sending rate, and the received rate are computed due to delays in the system. This may delay the convergence of the system. Using Eq. (3.11), the sender updates the rate $s_i(t)$ using the last taken control $s_i(t-1)$ as reference rate and a delayed feedback term. This feedback represents pricing information about the control taken D_i^R time units earlier. For example, if there is no congestion, the received rate is equal to the sending rate at time $t-D_i^R$. In that case the update equation will lead to a pure increase of the rate. In the event of a congestion, only part of the sending rate is received so $r_i(t-D_i^B) < s_i(t-D_i^R)$. Hence the price of the resource for flow i is increased and the rate will be re-adjusted downwards accordingly.

In order to cope with delays and improve the stability of the system, we can rather use $s_i(t-D_i^R)$ as the reference rate for the update equation [68, 73], so that the temporal drift between reference rate and pricing function is virtually cancelled. However, differently from [73] we do not intend to use a constant *willingness to pay* throughout the network, as the rate allocation should reflect the relative utilities of the various streams. Hence, in order to eliminate the temporal drift between the term of Eq. (3.11) that increases the reference rate and the feedback term that penalizes the reference rate, we need to evaluate the Utility function at the rate given by $s_i(t-D_i^R)$ as well. Finally, the proposed controller uses the following discrete-time rate update equation for each flow i:

$$s_i(t) = s_i(t-D_i^R) + \kappa_i s_i(t-D_i^R)\left[U'\left(s_i(t-D_i^R)\right) - p_i(t-D_i^B)\right]. \quad (3.13)$$

3.4.2 Role of Utility Functions

In the analysis about the stability of the distributed rate allocation algorithm, the only assumption on the continuous utility functions relies in their concavity. This leaves quite some flexibility in the choice of these functions, such that they can be chosen to correspond to the characteristics of the target application. In particular, the choice of distinct utility functions for each stream directly

influences the rate allocation among all streams congesting a bottleneck in the stable state. From a networking point of view, it can be inferred from the rate update equation (3.13) that the average experienced loss rate $p_i(\cdot)$ is equal to the gradient value of the utility function, evaluated at the rate operation point that is reached in the stable state. The controller then maximizes the system's aggregate utility by iteratively allocating more rate to streams that have a larger benefit at the current rate operation point.

As the normalized congestion signal $p_i(\cdot)$ takes on values in the interval $[0, 1]$, the utility gradient values of any stream in the system are normalized by a system-wide constant. This constant corresponds to the largest gradient value any used utility function throughout the system can take on. It is important to note that the normalization constant should not be stream-dependent in order to avoid distortions between the relative utility curves assigned to different streams. The maximum gradient value in the system also drives the maximum observed stable-state loss rate per stream throughout the system. Hence, by correctly scaling all the gradients, we can bound the maximum loss rate π to the maximum value that can be tolerated by the target application. As π has an effect on the gradient normalization, it also needs to be a global, system-wide constant. This is required in order to avoid distortions between the relative utility curves describing the different streams. Hence π does not allow to tune the loss rates experienced by a particular stream as a function of the specific video signal it transports, but rather represents an upper bound on the worst case loss that should be experienced by any stream. Hence, we can finally rewrite the Eq. (3.13) as:

$$s_i(t) = s_i(t - D_i^R) + \kappa_i s_i(t - D_i^R) \left[\pi \cdot U'\left(s_i(t - D_i^R)\right) - p_i(t - D_i^B)\right]. \quad (3.14)$$

3.4.3 Video Utility

We describe now in more detail the choice of utility functions for video streaming applications. An obvious choice is to relate the utility of a media stream to the rate-distortion characteristics of the encoded sequence. In that case the above control algorithm iteratively allocates the rates among streams proportionally to their contribution to the average video quality computed on all the streams.

We focus in the rest of this chapter on video encoders that provide the possibility to generate traffic that can be tuned to achieve any rate within a bounded rate interval. Such encoders include for example Motion-JPEG2000, in which each frame is progressively intra-coded: the rate of each frame can thus be adapted by selectively dropping wavelet coefficients, while increasing the distortion gracefully. Another example is given by the progressive refinement (PR) slices that were proposed as part of the scalable video coding (SVC) amendment to the H.264/MPEG-4 AVC standard. Using SVC, a video can be decoded at the full encoding rate, yielding the highest possible decoded quality in terms of SNR, framerate and resolution. Aside from this, one has the option to decode a number of sub-streams that have been specifically included while encoding. Each of the sub-streams represents a version of the video that is degraded in either SNR, frame rate or spatial resolution, or any combination of these. Typically it comes down to the targeted application or to the capabilities of the decoding client to decide which sub-stream is most useful. Finally, each of these sub-streams can be encoded using progressive refinement slices, pro-

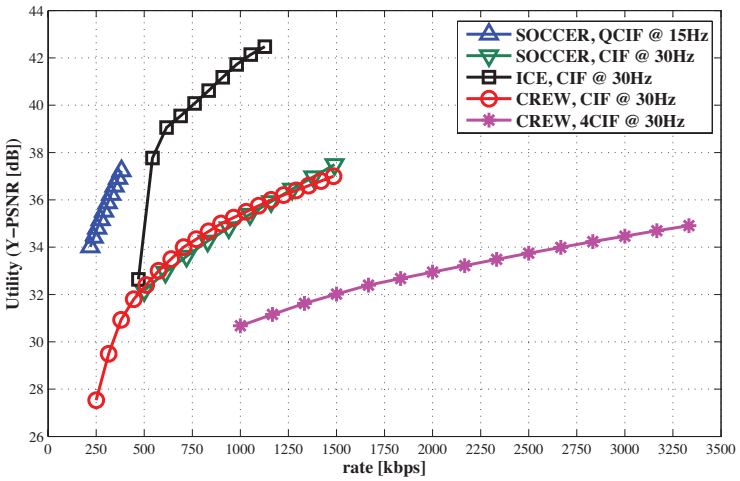

Figure 3.2: Utility curves can reflect the rate-distortion characteristics. Here we show the rate-distortion curves of several H.264-SVC(FGS) encoded test sequences (SOCCER, CREW, ICE).

viding fine granularity scalability (FGS). These slices can be cut at any point in order to finely tune the rate within a substream, while gracefully increasing the distortion. Both of the aforementioned encoding choices rely on fine grained rate adaptation that results in SNR scalability. Hence they are able to generate the kind of traffic we aim for: any rate within a bounded rate interval can be achieved. As this results in SNR scalability, the resulting rate-distortion curves over that interval are concave and can be used as utility functions for the respective streams.

Examples of the utility functions used in this chapter are illustrated in Figure 3.2 for a number of test sequences at different framerates and resolutions. They have been computed on video sequences encoded using the H.264 SVC extension and using PR slices. The sequences have first been segmented into Groups of Pictures (GOPs) of equal size, and each GOP has been encoded independently into PR slices. The utility functions for the complete sequence have finally been extracted by decoding each GOP at a number of fixed rate points, and by averaging the resulting Y-PSNR value at each rate point for all the GOPs of the sequence. Note that we do not rely on an analytical model to specify the utility curves for each stream contrarily to [34, 74], but we rather use the exact rate-distortion information.

Finally, it should be noted that our framework is very generic and that any concave and continuous function is valid as utility function. In particular, priority or service classes can also be incorporated in the utility functions. For example, we can design utility functions for 2 classes of streaming clients decoding the same sequence. The rate for receivers in the lower Priority Class is adapted using the rate-distortion function. The receivers in the higher Priority Class use a different utility function, whose gradient corresponds to an up-scaled version of the rate-distortion curve at any rate point. Clients in the higher Priority Class hence see the quality of their video streams be adapted

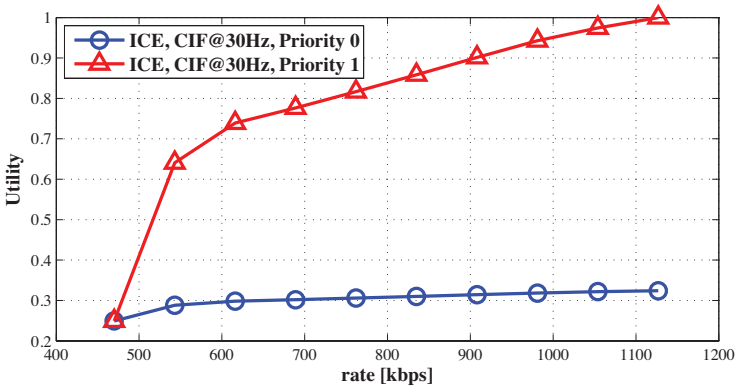

Figure 3.3: Utility curves can also reflect traffic prioritization. Here Priority Class 0 uses the (scaled) R/D information for the ICE sequence, whereas users in Priority Class 1 use a utility function that has a gradient five times as large at each rate point.

faster and reach a higher level. Such utility functions are illustrated in Figure 3.3, where the utility gradient is multiplied by five for the high priority class.

3.5 Implementation

We propose now a practical implementation of the control system described above. We have shown that the distributed control algorithm is stable, even with heterogenous feedback delays, which are likely to happen in real scenarios. We outline a scalable implementation of the proposed framework, and we explain in detail the design choices for the rate update equation and the distributed control in a video streaming system.

3.5.1 Design issues

From the above development, it is clear that the rate update equation (3.14) has ideally to be applied as often as possible in order to emulate a continuous time control system. This requires the availability of accurate feedback at each moment in time and at each end-point in the network. At the same time, it implies that the sending rate can be adapted at any time instant. However, when we are dealing with real video streams, it is clear that we cannot adapt the video rate at any arbitrary time instant. For example, dropping random parts of the bitstream results in large and uncontrolled losses of quality due to the inherent decoding dependencies between video elements. Scalable video coding provides an interesting solution for flexible adaptation of the bitstream. For example, encoding formats such as H.264-SVC(FGS) form independently decodable compressed units such as Groups of Pictures (GOP), which are typically groups of 16 or 32 frames. They further offer the possibility to extract a substream of a given rate from any independently decodable entity of the stream. The rate control has therefore to be performed on GOPs, and the rate update equation

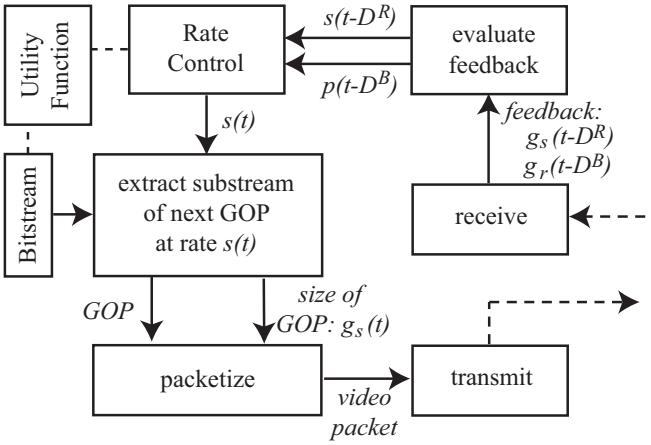

Figure 3.4: Schematic view of our sender implementation.

(3.14) of the control system should be synchronized with GOP boundaries. It is applied after each transmitted GOP, and defines the rate of the substream to be extracted from the next GOP.

The decision of the control algorithm relies on the feedback received from the clients, about the state of the streaming session. In addition, it uses the result of the decisions taken in previous iteration of the algorithm, as seen in the update equation (3.14). The controller has to know the sending rate that was computed $(t - D_i^R)$ time units earlier, where D_i^R is the *arbitrary* experienced Round-Trip delay. Maintaining the history of earlier decisions for each flow at each sender does however not provide a viable solution as it does not scale. In addition, it relies on very accurate Round-Trip Time measurements in order to avoid any drift between the sending rate that is actually used in the update equation and the feedback that is received. The utility function is evaluated on the sending rate, yielding the willingness to pay for the bandwidth that is offered, while the congestion signal that is fed back gives the price of the bandwidth resources. Any temporal drift between the computation of these values slows down the convergence of the distributed algorithm. A scalable system has therefore to avoid any temporal drifts, by gathering together the congestion signal, and the sending rate it corresponds to. In the next section, we propose a light-weight application layer protocol that respects the above constraints using the time-stamp information included in the transported video streams.

3.5.2 Proposed control system

We propose now an implementation of the distributed control system at the application layer, for streaming scalable video over the classical RTP/UDP/IP protocol stack. For the sake of clarity we will drop the index i that specifies a particular flow and sender/receiver pair in what follows. We describe now in details the behavior of a client and sender pair, and all the concurrent streaming sessions in the system adopt the same strategy.

A schematic view of the sender implementation is first given in Figure 3.4.

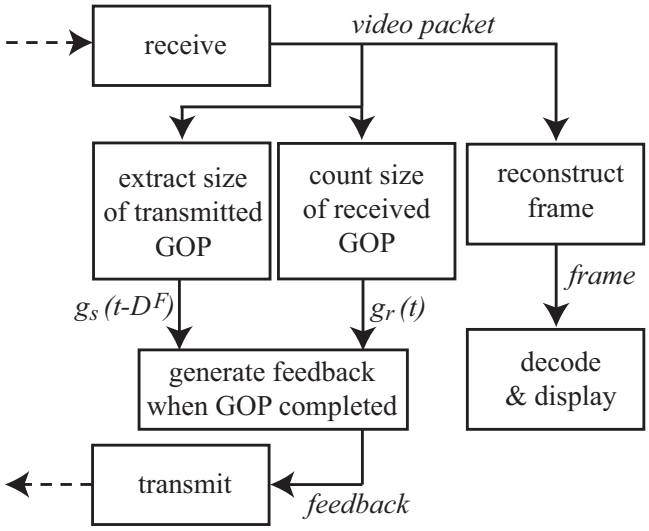

Figure 3.5: Schematic view of our receiver implementation.

Upon reception of a feedback message, the rate controller given by Equation (3.14) updates the targeted sending rate to $s(t)$, which depends on the media stream that is being transmitted through the chosen utility function. The next GOP of the scalable bitstream is then optimally truncated so that its rate matches the target rate. The resulting substream is then packetized according to the video frames and Network Abstraction Layer (NAL) units and injected into the network. The exact size of the extracted GOP, denoted by $g_s(t)$, is added to the header of all the packets in the GOP. Note that the sender knows the framerate f [Hz] of the stream as well as the length k of the GOP in frames, since those are usually negociated by the sender and the receiver. The sending rate is therefore simply given by

$$s(t) = \frac{g_s(t)}{k} f \ . \tag{3.15}$$

The client behavior is then represented in Figure 3.5. It receives the packet stream and detects potential packet losses. It further reassembles the media bitstream and sends the reconstructed decodable parts of the bitstream to the decoder. At the same time, the receiver counts the number of bits $g_r(t)$ that are received for the current GOP. After the current GOP has been completely received, the receiver sends a feedback to the sender, where it includes the received size $g_r(t)$ as well as the GOP size $g_s(t - D^F) \geq g_r(t)$ that is read from packet headers. When the feedback eventually reaches the sender, it accurately reconstructs the control decision taken $D^R = D^F + D^B$ time units earlier. Even if the Round-Trip time D^R is arbitrary, the sender can replicate the past control decision and compute the previous sending rate as

$$s(t - D^R) = \frac{g_s(t - D^R)}{k} f \ . \tag{3.16}$$

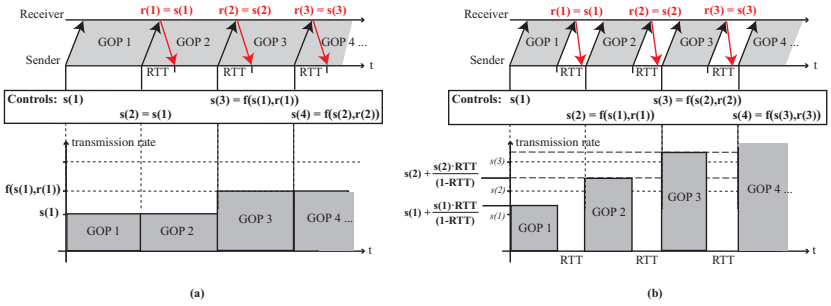

Figure 3.6: Guard interval illustration. *Left:* When no guard interval is used, the available feedback is only considered in the control loop at a later stage. For example, the rate for GOP 3 is dependent on the rate and feedback of GOP 1. *Right:* By using a guard interval, the feedback can be integrated more rapidly in the control decisions.

Similarly it reconstructs the rate that is actually received by the client as:

$$r(t - D^B) = \frac{g_r(t - D^B)}{k} f \ . \tag{3.17}$$

The sender has thus access to all the information necessary to evaluate the congestion signal as given in Equation (3.12). Note that it uses the reference sending rate $s(t - D^R)$ without relying on a stored history of controls nor on exact Round-Trip delay measurements. All the information is rather extracted from the received feedback packets, and the media stream characteristics that are known at the sender. Furthermore, the information that is transmitted in the feedback packet can be accurately synthesized at each client even when media packets are lost, since all the packet headers contain the GOP size information $g_s(t)$. It is therefore sufficient to receive one media packet of the GOP for the client to generate an accurate feedback signal that stabilizes the control system.

Even if such a system is scalable and robust, it does not perform optimally yet due to network latency. Consider for example the scenario depicted in Figure 3.6(a). The feedback is only sent after a GOP n is completely received. Hence, it is available at the sender at the earliest one Round-Trip time after the last packet of the GOP has been transmitted. At the time the controller has to decide on the sending rate for the next GOP $n+1$, it does not have access to any feedback yet. It can thus only decide to keep on transmitting at the same sending rate. Even though an accurate feedback becomes available during the transmission of the GOP $n+1$, it can not be incorporated in the control system due to the structure of the video streams. Therefore, it only affects the sending rate of the GOP $n+2$, which introduces a latency that is almost equal to the duration of one GOP.

In order to avoid such a latency that slows down the convergence of the control system to a steady-state solution, we propose to slightly increase the actual transmission rate. The data of a GOP with rate $s(t)$ is thus transmitted at a rate $s(t) + \frac{s(t) \cdot RTT}{1 - RTT}$. Here RTT is an estimate or an upper bound of the Round Trip Time experienced by the stream. We express RTT in seconds and suppose without loss of generality that it is always less than 1. This imple-

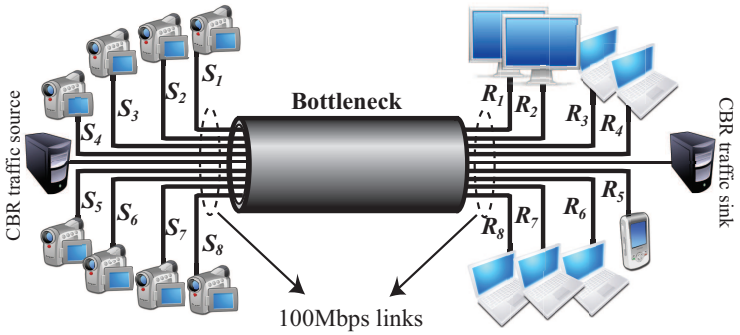

Figure 3.7: Simple simulation topology.

mentation therefore creates a guard-interval between the transmission of two adjacent GOPs n and $n+1$, so that the feedback information about the GOP n is likely to be received on time for controlling the sending rate of the GOP $n+1$. Figure 3.6(b) sketches the improved control sequence. The small increase in the transmission rate permits to increase the sending rate of the streaming session faster compared to the case illustrated in Figure 3.6(a).

3.6 Simulation results

3.6.1 Setup

We illustrate now the behavior of the distributed rate allocation algorithm with simulation results in different streaming scenarios. We consider the general network topology depicted in Figure 3.7, where eight different sender-receiver pairs connect to a network through high-speed links but share a common bottleneck link. Each sender-receiver pair runs its own rate-control loop independently of the other pairs. It relies only on end-to-end feedbacks and the Utility information relative to the video sequence that is streamed. In particular, each sender and receiver knows neither the capacity of the bottleneck link, nor the number and nature of the other streams that are using the same bottleneck link. Finally, we also consider a constant bit rate (CBR) source-sink that also uses the same bottleneck link without any adaptive rate control.

The sequences transmitted by the senders are encoded using the H.264 SVC reference software (JSVM). In the encoding, we constrain each GOP of a given stream to span the same range of encoding rates. Unless otherwise stated, the Utility functions are given by the Rate-Distortion information extracted from the encoded sequences, as depicted in Figure 3.2. The distribution of the test video sequences between the different sender-receiver pairs is given in Table 3.1 for the scenarios considered in our simulations.

The proposed rate allocation algorithm is implemented in the application layer of the NS-2 simulator platform, and controls the rate of the underlying UDP transport protocol. The sender runs the control algorithm using the rate update equation (3.14) based on the utility functions of the transmitted stream, and the feedback it gathers from the receiver. Unless otherwise stated, the

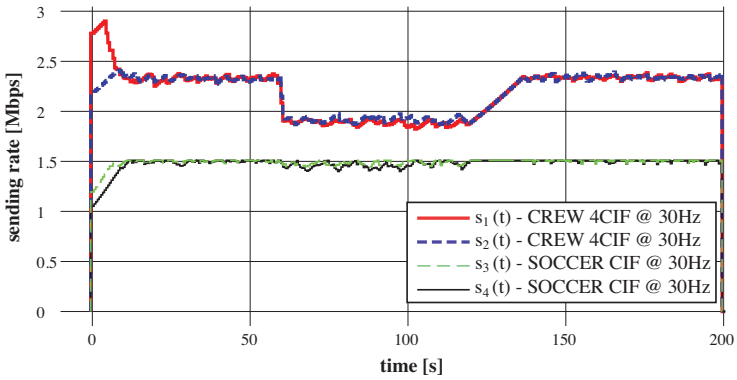

Figure 3.8: Resulting sending rates/controls for scenario 1.

maximum loss rate factor π is set to 0.05 and the gain factor κ equals 1. The sender then extracts a SVC substream at the target sending rate with help of the tools available in the JSVM software distribution and sends the appropriate packets to the receiver. Upon reception of each packet, the receiver checks for losses and eventually reconstructs a received packet trace for each GOP. It forms a feedback packet based on the information it has received. The video sequence is finally decoded with the packets that have been correctly transmitted.

Due to the lack of a proper error concealment implementation in the H.264 SVC reference code, we chose to implement an *active node* in the bottleneck router, following the general idea from [75]. If the router has to drop packets due to a congestion, it will first drop those packets that are of least importance to the decoded stream, and which are discardable. For this to work we added the layer index and a discardability flag, retrieved from the bitstream syntax, to each packet header. It is important to note that this approach does no change anything to the rate-update control loop. It results merely in a shift of the observed PSNR values to more realistic values, which are in accordance with those that would be observed in end-systems that use proper concealment techniques.

3.6.2 Adaptation to changing bottleneck capacity

We analyze the effect of a changing bottleneck capacity, or equivalently the effect of varying background traffic in Scenario 1. We set the bottleneck bandwidth

$S_{1,2} \rightarrow R_{1,2}$	CREW, 4CIF @ 30Hz
$S_{3,4} \rightarrow R_{3,4}$	SOCCER, CIF @ 30Hz
$S_5 \rightarrow R_5$	SOCCER, QCIF @ 15Hz
$S_{6,7} \rightarrow R_{6,7}$	ICE, CIF @ 30Hz
$S_8 \rightarrow R_8$	ICE, CIF @ 30Hz (Priority 1)

Table 3.1: Distribution of the test video sequences among the sender-receiver pairs.

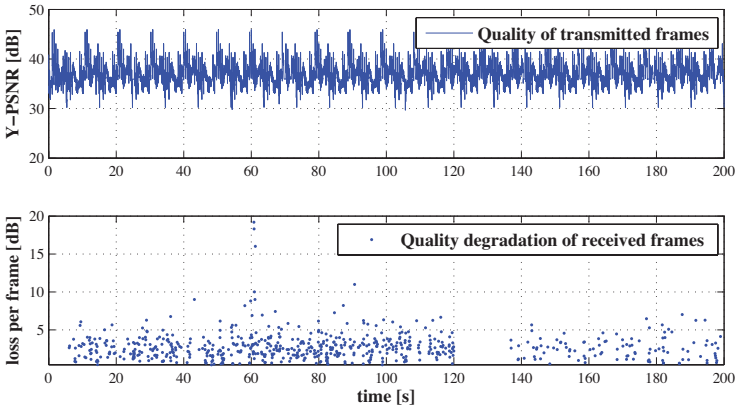

Figure 3.9: Results for scenario 1. *Top:* Y-PSNR of each frame the SOCCER CIF@30Hz sequence transmitted by sender S_3. *Bottom:* Quality loss for each frame of the sequence, as received by client R_3.

to 8 Mbps and let senders S_1, S_2, S_3 and S_4 start transmitting simultaneously their respective video streams at time $t = 0$. After 60 seconds, we add a 1 Mbps CBR background traffic stream in the same bottleneck channel, which translates into a sudden bottleneck capacity reduction for the four sender-receiver pairs. The background traffic is switched off again after one further minute.

The resulting sending rates are shown in Figure 3.8 where they illustrate some key properties of the proposed algorithm. Clearly, the algorithm converges quickly to a stable state at the beginning of the transmission, as well as around the changes in background traffic. The stable state corresponds to a maximization of the utilities of the different streams. Even though each stream is regulated independently of all the other ones, the same sequences converge to the same rate in the stable state. Once the CBR traffic joins the bottleneck, the streams react quickly to the new situation and settle in a new stable state almost immediately. Note that this behavior is very different from the sawtooth behavior seen in TCP for example. As the CREW sequences carried by S_1 and S_2 have a lower utility gradient than the SOCCER sequences transmitted by S_3 and S_4, the rates of the latter are hardly affected by the drop in bottleneck capacity. In other words, as the background traffic lowers the bottleneck bandwidth, the distributed control system allocates less rate to the CREW streams, as this results only in a minor overall quality degradation. A rate reduction for the SOCCER streams would result in a larger quality drop. Once the background traffic is switched off, the four streams return rapidly to their original stable rates. The average quality of the streams sent by the senders are reported in Table 3.2 in terms of Y-PSNR, along with the respective quality reduction due to packet loss during the transmission.

Finally, we show the temporal evolution of the quality for one of the test streams in Figure 3.9. We report the Y-PSNR quality for each frame of the stream transmitted by S_3. We also compute the quality loss at the receiver by substracting the Y-PSNR of each received frame from the Y-PSNR of the corresponding frame transmitted by the sender. Unsurprisingly, there is no loss

when there is spare bandwidth available on the bottleneck link, i.e. during the first 10 seconds and after the CBR source switches off. There is however more packet loss due to congestion when the background traffic is active (i.e., in the timespan from 60 to 120 seconds). However, the steep gradient of the SOCCER Utility function prevents the rate from dropping. We note that loss cannot be completely avoided even in the steady state, as all the streams simultaneously compete for bandwidth shares until saturation of the bottleneck bandwidth is reached. The small quality degradation resulting from these losses could be further reduced by the use of error resiliency tools at the decoder or through forward error protection.

3.6.3 Adaptation to new streams

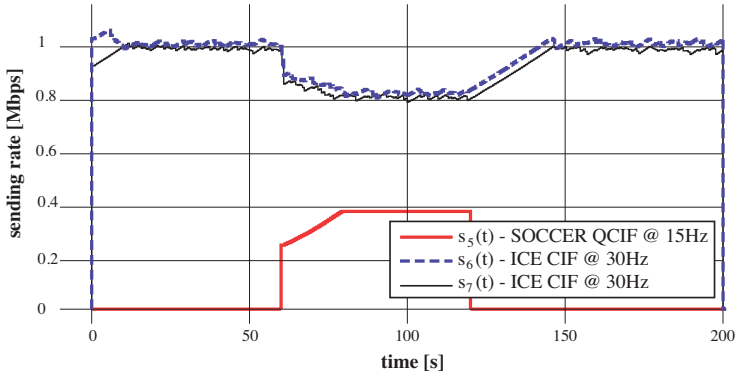

Figure 3.10: Resulting sending rates/controls for scenario 2.

In Scenario 2 we analyze the allocation of bandwidth resources when streams join and leave the bottleneck channel. We set the bottleneck bandwidth to 2 Mbps, and the sources S_6 and S_7 both start transmitting at time $t = 0$. After 60 seconds, the sender S_5 starts streaming during one minute, and then leaves the bottleneck again. The sending rates for this dynamic join/leave scenario are depicted in Figure 3.10.

It can again be seen that the system converges rapidly to a stable state, where the two equivalent streams get an equal share of the bottleneck capacity. Once the third source initiates its streaming session, the sending rates of the ICE sequences are gracefully brought down in order to adapt to the new situation. The new stream that contains the SOCCER QCIF sequence has a steep utility function and is therefore aggressive in getting shares of the bottleneck bandwidth. The quality of the transmitted streams and the corresponding quality drops due to packet loss are given in Table 3.2.

In order to illustrate the benefit offered by a slight increase in the transmission rate, we have run the same simulation where the senders however do not implement the guard interval presented in the previous section (see Figure 3.6). The corresponding sending rates are given in Figure 3.11. It can be seen that in this case the algorithm has a slower convergence to the steady state due to the delay introduced by late feedbacks. This penalizes the average quality by

about 1 dB per stream with respect to the same simulation scenario where the guard interval is used.

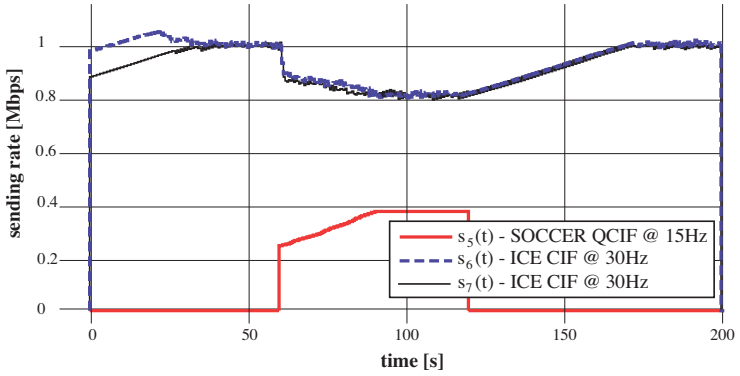

Figure 3.11: Resulting sending rates/controls for scenario 2 when the guard interval scheme is not used.

3.6.4 Adaptation to priority classes

As stated earlier, the concept of Utility function is very general and extends beyond the common characterization of video streams by their rate-distortion characteristics. To illustrate this, we consider in Scenario 3 a situation similar to the previous one, but where the 2 streams from S_6 and S_8 correspond to the same video content with different priority classes. They are characterized by the 2 Utility curves depicted in Figure 3.3, which could for example model two clients with differential treatment. The resulting sending rates for this simulation run are shown in Figure 3.12. They illustrate that the choice of Utility function directly drives the performance of the streaming application, which clearly favors the high priority client. We report again in Table 3.2 the average quality of the transmitted streams, along with the quality drops due to packet loss. We see that the high priority stream benefits from a 2dB quality gain compared to the low priority stream with the same video content.

Finally, we have run the same scenario several times using different values of π in the rate update Equation (3.14). Remember that this factor scales all the Utility gradients in the system and should thus bound the loss rate experienced by any stream in the stable state. The result of this simulation is given in Figure 3.13 where we show the experienced loss (which is equal to the congestion signal $p(t)$), as seen by receiver R_8 for the three cases where π equals 0.03, 0.05 and 0.1 respectively. As expected the packet loss rate is bounded by π in each of these cases. The parameter π is therefore an essential tool in order to adapt the rate allocation algorithm to a given decoder. A decoder can be characterized by a loss rate that it can cope with while decoding a stream, mostly through the use of error-concealment techniques. By matching π to the maximum tolerable loss rate at the decoder, we therefore ensure that the received stream is decodable when the system is in the stable state.

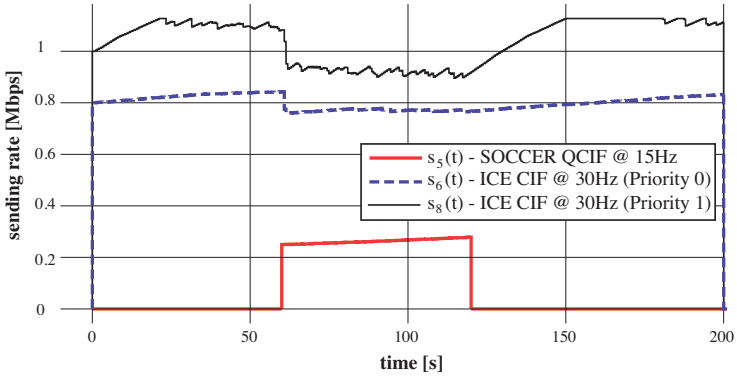

Figure 3.12: Resulting sending rates/controls for Scenario 3. Both S_6 and S_8 transmit the same ICE sequence, but using different Utility curves.

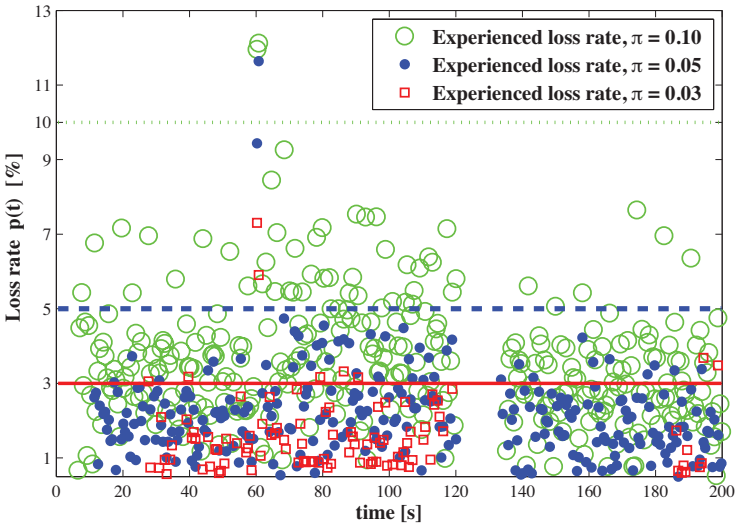

Figure 3.13: Results for scenario 3: experienced loss rates $p(t)$, expressed in percentages, at receiver R_8 for different values of π.

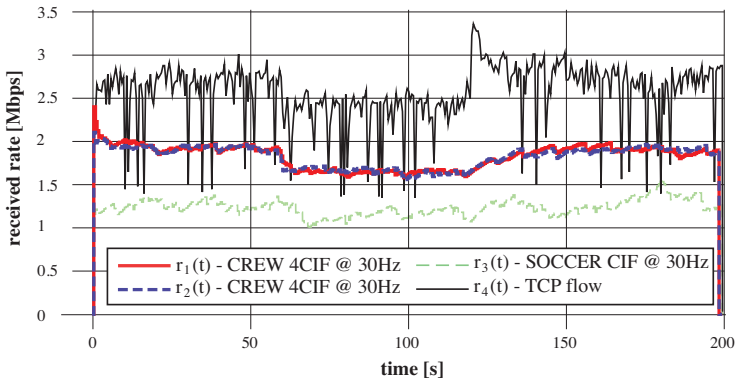

Figure 3.14: Received rate in Scenario 1, where the stream from S_4 is replaced by a TCP-regulated flow.

3.6.5 Adaptation to TCP traffic

We finally illustrate the behavior of the rate allocation algorithm when the bottleneck channel is shared with flows controlled by TCP. We have considered a scenario similar to Scenario 1, but we replace the 4th stream in the system by an FTP data flow, which is regulated by TCP (Tahoe implementation). The results of this experiment are shown in Figure 3.14, which presents the received rates at each of the clients. The goodput of the TCP flow is shown as stream 4. Interestingly, these results show that our proposed congestion control protocol can coexist with TCP on the same bottleneck channel. Compared to the Scenario 1, the sending rates of the media streams converge slower and show slightly larger oscillations around their stable state values. This is due to the rapid changes in available bandwidth, which is mostly driven by the TCP flow. However, one can observe that the rates are still smooth and that the CREW and SOCCER streams are still handled appropriately, according to their respective Utility functions.

While this simulation does not represent any formal proof of proper distribution of resources between TCP flows, and the streams controlled by the algorithm proposed in this chapter, it still shows that our algorithm does not starve TCP flows of the network resources. This corresponds to the results presented in [30], which state that the long term behavior of TCP Tahoe is equivalent to the one of a controller that maximizes a Utility function of the form $U(s) = \arctan(s)$. As this is a concave utility function, we should thus expect TCP Tahoe streams to be able to compete with streams regulated by any other concave Utility functions in a stable system.

Finally, we shall note that our algorithm is not *TCP-friendly* in the sense that the allocated rates yield an average per-flow bandwidth that is not equivalent to the one allocated by a TCP connection. TCP is in general more aggressive and tends to fairly share the average rate of each session, without considering the characteristics of each stream. Our algorithm targets a different objective, which is the effective allocation of the resources in order to maximize the average utility, or equivalently to make the best use of bandwidth resources in terms of application requirements.

3.7 Conclusions

We have presented a distributed rate allocation algorithm that targets the optimal distribution of bandwidth resources for improving the average quality of service of media streaming applications. We have first extended the framework of Network Utility Maximization with an important stability proof, which states that the algorithm is stable under general concave Utility functions and randomly delayed feedback. This result is particularly interesting for the implementation of rate allocation algorithms in real scenarios. We have then proposed an effective and scalable implementation of the distributed control algorithm with a light-weight application-layer protocol. We have further proposed a few practical utility functions for streaming video sequences. Finally, we have analyzed the behavior of the proposed solution with extensive NS-2 simulations, where we have considered H.264 SVC-FGS encoded sequences as an illustration. We have shown that the bandwidth allocation actually respects the constraints imposed by the utility functions, and that the system converges quite rapidly to a stable state after changes in the network. The proposed algorithm therefore provides an interesting solution for rate allocation in distributed streaming systems.

The proposed framework could be extended to the usage of step-like Utility functions rather than continuous concave functions. Although such functions would be able to model the characteristics of layered video streams that do not have the FGS property, they would also imply to recast the NUM problem as an integer optimization problem. This topic being beyond the scope of the present text, we will in the following Chapter propose to use smoothing and scheduling techniques in order to map the traffic generated by such layered video streams to the RCC controls that are decided by our rate update equation.

Scenario 1	Mean Y-PSNR at sender	Mean loss at receiver
$S_1 \rightarrow R_1$	33.30 dB	0.65 dB
$S_2 \rightarrow R_2$	33.28 dB	0.75 dB
$S_3 \rightarrow R_3$	36.62 dB	0.52 dB
$S_4 \rightarrow R_4$	36.56 dB	0.49 dB
Scenario 2		
$S_5 \rightarrow R_5$	36.12 dB	0.05 dB
$S_6 \rightarrow R_6$	41.06 dB	0.82 dB
$S_7 \rightarrow R_7$	41.24 dB	0.80 dB
Scenario 3		
$S_5 \rightarrow R_5$	34.01 dB	0.11 dB
$S_6 \rightarrow R_6$	39.99 dB	0.53 dB
$S_8 \rightarrow R_8$	41.92 dB	0.91 dB

Table 3.2: PSNR quality for the different streaming scenarios [dB].

Chapter 4

Smoothing of Layered Video

4.1 Background

In Chapter 3 we have shown that a media-friendly Rate and Congestion Control (RCC) Algorithm can be implemented straightforwardly using the framework of Network Utility Maximization [3]. A key condition for such a system to be efficient is the capacity to match the sending rates of each source in the network to their respectively available channel bandwidth, or equivalently, to the rate control taken by the RCC Algorithm. We have been able to provide this feature in the framework of Chapter 3 through the use of video encodings that have the Fine Granularity Scalability (FGS) property: such streams can be adapted to match any bitrate, hence the *source rate* of the transmitted stream can be adapted to the available *channel rate*. In that case, the source rate equals the *sending rate* of the sending server.

In this Chapter we will consider the more practical case that consists in streaming a hierarchically layered video stream that does not have the FGS property. Hence the video can only be adapted by adding or removing a complete video layer to or from the stream. This *coarse granularity scalability* feature is today the only scalability feature that has been adopted into any encoding standard, hence its practical importance. It should be noted that during the standardization procedures of both the MPEG-4 standard [41], and the scalable video coding extension (SVC) to the H.264 AVC standard [38], FGS scalability methods have been proposed. These features have however in both cases failed to be adopted into the final version of the standard, due to the bad coding efficiency of FGS schemes: in order to have the FGS property, a stream needs to carry a lot of redundancy, which prohibits efficient compression.

From Chapter 3 it should be clear that the actual sending rate of a source directly influences the rate that will be allocated to it by the RCC Algorithm in future controls. This dependence is in general not taken into account in the design of media streaming systems. Most of the time the assumption that the channel bandwidth varies independently of the sending rates prevails in the media communication community. Such a situation would allow a source to adapt its source rate by adding/dropping video layers, once it observes that a

corresponding channel bandwidth is available. We argue in this Chapter that sending rate and channel rate are not independent, and that a sender needs to carefully adapt both its source and sending rates in order to contend for available transmission resources. Indeed, if a source sends at a lower rate than that decided by the RCC Algorithm, other sources in the network will tend to contend for the share of bandwidth that has been left unused by this behavior. It is thus crucial to implement a sending rate that is as close as possible to the channel rate decided by the RCC Algorithm, in order to correctly implement the sequence of controls that is proven to drive the system into a stable state. Senders that transmit at higher rates tend to create congestions and behave unfairly, while senders that transmit at lower rates are unable to probe for potentially available increased transmission resources.

While the use of FGS video streams allows to handle this problem inherently, the situation is different when it comes to streaming hierarchically layered streams that do not have the FGS property. Only a discrete set of *video source rates* is available for transmission, given by $v = \sum_{i=1}^{l} \lambda^i$, for $1 \leq l \leq L$. Here each operand λ^i of the sum corresponds to the rate of one of the L available layers, which are all given by the set $\mathcal{L}$:

$$\mathcal{L} = \left\{\lambda^1, \ldots, \lambda^L\right\}. \tag{4.1}$$

In order to efficiently use the available network resources and to keep the system stable, the RCC Algorithm may however decide to use a transmission rate that can not be matched by any of the L available source rates. In this Chapter we analyze how we can temporarily generate these intermediate sending rates from a buffer of available video data by using smoothing techniques [42, 48, 49, 50]. This will enable us to have some flexibility in adapting the source rates to the channel rates. Following this approach, we will derive conditions on both the encoding rates in $\mathcal{L}$ and the incurred prefetch delays. These conditions will depend on the characteristics of the underlying RCC Algorithm, which computes the available transmission rates.

The remainder of this chapter is organized as follows: in Section 4.2 we provide some details on Rate and Congestion Control as well as smoothing techniques, which will be useful in our developments. Our contribution, which enables us to adapt the source rate of layered video streams to available channel rates, is detailed in Section 4.3. In Section 4.4 we address practical scheduling issues that arise using our solution, and propose a practical algorithm which efficiently addresses all of these considerations. In Section 4.5 we apply our findings to the media-friendly rate and congestion control algorithm presented in Chapter 3. Finally we provide extensive simulations stemming from our NS-2 implementation of the proposed system in Section 4.6.

4.2 Preliminaries

4.2.1 Rate and Congestion Control

In the setup of Chapter 3, we were always able to match the rate that was decided by the RCC Algorithm exactly with a source rate $s(t)$, through the use of FGS video streams. As will be seen in what follows, this is no longer the case when we consider video streams that do not have the FGS property. That is

why we need to differentiate the sending rate $s(t)$ explicitly from the rate that is computed by the RCC Algorithm. In the remainder of this chapter we will denote the available rate at which an application can transmit (i.e., the available channel rate computed by the RCC Algorithm) by $c(t)$.

We provide a brief high-level description on the principles of Rate- and Congestion Control Algorithms (RCC). In general, the channel rate that is available for use by any networked application is computed by an RCC algorithm. This rate is periodically recomputed and the resulting sequence of rates forms a control sequence that drives the network into a stable state. RCC algorithms can be implemented in lower layers of the network stack, as it is the case in TCP for example [1], which runs on the transport layer, or in the application layer itself, such as for example in TCP-frienldy Rate Control (TFRC) [2]. These are distributed algorithms that take local decisions in order to converge to a globally stable network state. In TCP, a stable state is characterized by a *fair* sharing of the available network resources among all competing flows. Based on previous local decisions and feedback on the current network state, the RCC at each sender decides iteratively at which rate it should transmit data to its client. If each sender in the network applies their local sequence of controls (i.e., sending rates) that are generated in this way, the distributed algorithm will converge to a *stable* network state, which is characterized by high resource utilization and bounded per-flow losses.

If senders do not comply to the controls taken by their RCC instance, there is no guarantee for the network to reach a stable state. Hence non-complying sources tend to drive the network into an unstable state in the worst case and to bad resource utilization in the best case.

In traditional data transfers over the Internet, the above observations are of minor relevance: data is transferred without delay constraints and the Transport Layer takes care of both implementing the RCC and adjusting the sending rate accordingly. In media-streaming applications however, the RCC is preferentially implemented in the Application layer. This stems from the fact that on the one hand the application needs to have some control on the rate that is allocated. On the other hand, the application needs to know the rate decided by the RCC in order to adapt the bitstream that is to be transmitted accordingly due to timing constraints. Hence the sending rate needs to be matched to the channel rate at the Application layer. The source rate needs to be carefully selected or smoothed if the sequence of achieved sending rates should conform to the rate control decisions.

4.2.2 Scheduling using prefetching

A key component in any rate adaptation mechanism is the scheduler. It builds a transmission schedule by deciding which packet is to be transmitted at which time during the transmission process. This decision typically depends on previously taken decisions, as well as on information about the currently available channel bitrate. In streaming scenarios involving layered video, the scheduler decides when to transmit which layer of which frame.

While streaming video, there are timing constraints that need to be satisfied at any time t. Let k denote the index of the frame with the most stringent timing constraint. If frame k, and any frame leading up to k, has not been scheduled for transmission by time t, it will not reach the decoder by the time its decoding

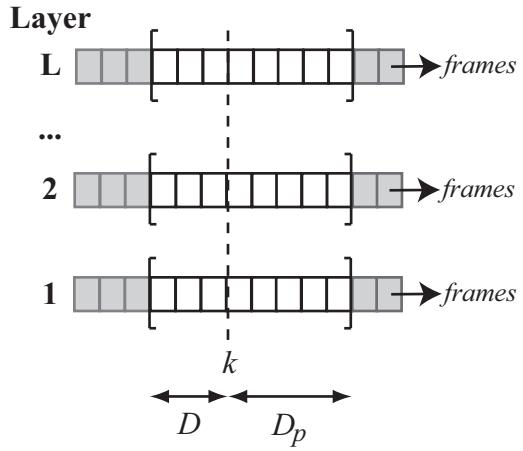

Figure 4.1: Illustration of the playback delay and prefetch delay concepts for a set of L layers.

timestamp expires. In that case, the client will experience a buffer underflow. Let D denote the constant *playback delay* at the receiver. For the sake of simplicity, we express delays in a number of frames throughout this chapter, i.e., if $D = 1$, then its duration is equal to the inverse of the stream's framerate. This assumption generally holds if the video is encoded using forward-prediction, i.e. the structure in a Group of Pictures (GOP) is $(IPPP\ldots)$. A receiver will wait until it has received D frames before it starts decoding, or playing out, the video stream it receives. Hence D corresponds to a buffer on the receiver side, which is used to compensate for sudden bandwidth outages for example [76]. Let then D_p denote the *prefetch delay*. It corresponds to a buffer at the sender side, in which the latter stores D_p frames that are available for transmission, but whose transmission deadline has not yet expired. The data in this buffer can be exploited to temporarily increase the sending rate above the actual source rate. This results in a partial transfer of the buffer to the receiver end. Obviously, the dimensioning of D_p is important for any smoothing algorithm. Note that the total delay experienced between encoding and decoding is the summation of these two delay components, in addition to a transmission delay that we suppose to be negligible.

Figure 4.1 illustrates these concepts for L layers of a stream. Consider for example that only layer 1 is selected for transmission. If the scheduler decides not to prefetch data, it will transmit each frame once its timing constraint becomes the most stringent. After the transmission of each frame, k will thus be incremented by 1. This will generate an outgoing sending rate of mean λ^1 bps. If a higher rate $\lambda^1 < c(t) < \left(\lambda^1 + \lambda^2\right)$ is decided by the RCC, the scheduler can fill the rate surplus $c(t) - \lambda^1$ by transmitting data from the next D_p layer-1 frames that it holds in the prefetch buffer.

In the following section, we will analyze more thoroughly which are the conditions on the layer rates and on the prefetch delay that need to be satisfied in order to adapt the source rates to the available channel rates.

4.3 Adaptation to congestion control constraints

In general, the rate profiles that are allocated by a Rate and Congestion Control algorithm can be partitioned into two disjoint sets of periods. Periods of *stable state* are characterized by a steady, relatively constant and smooth rate. In contrast, during periods of *convergence*, the RCC takes controls that allow the system to go from one stable state to the next. This happens whenever the availability of the network resources changes, due to different flows joining or leaving the network, or to topology changes for example.

If we can make sure that during both of these characteristic periods we will be able to generate the sending rates that match the controls dictated by the RCC, we can make sure that the network remains stable. A stable network state is characterized by an efficient utilization of the network resources and bounded per flow losses.

4.3.1 Stable state periods

We suppose, without loss of generality, that the stable state rate c which is reached by the RCC is a random variable that can take on values in a rate range given by $[c_{min}, c_{max}]$, and which is characterized by a probability distribution $p_c(x)$ defined on the same range. Using a set of layers (4.1), we can however only transmit at L rates for a sustained period of time, corresponding to the L achievable source rates. If the RCC Algorithm converges to a rate c that can not be matched by any of the source rates, $v = \sum_{i=1}^{l} \lambda^i$, for $1 \leq l \leq L$, the sender should select to transmit at the source rate that is closest to, but smaller than c.

These observations lead us to formulate the following problem: what is the number of layers L, and which are the layer rates λ^l, $1 \leq l \leq L$ that should be encoded in order to minimize the error made when sending at a rate $\tilde{c} = \sum_{i=1}^{l^*} \lambda^i \leq c$, instead of the steady rate c? Here l^* denotes the index of the highest layer that can be transmitted given the steady rate c.

This problem can be readily recast into a quantization problem: which is the optimal quantizer that minimizes the mean squared error (MSE) between the input c and the quantized value $\tilde{c}$? Without any specific assumptions on the actual network topology and dynamics, the distribution of c can safely be regarded as being uniform on $[c_{min}, c_{max}]$. The optimal quantizer, which minimizes this MSE for a uniform source, is a uniform quantizer with constant quantization step size $\Delta\lambda$, given by [77]:

$$\Delta\lambda = \frac{c_{max} - c_{min}}{L}. \tag{4.2}$$

If the input c lies in the bin $[i\Delta\lambda, (i+1)\Delta\lambda[$, $1 \leq i \leq L-1$, then the quantizer output reads as $\tilde{c} = i\Delta\lambda$. The MSE between the input c and its quantized version is given as:

$$MSE(c, \tilde{c}) = \frac{\Delta\lambda^2}{12}. \tag{4.3}$$

After combining Equations (4.2) and (4.3), and setting the rate of each layer l to:

$$\lambda^l = \Delta\lambda = \lambda \texttt{ for } 1 \leq l \leq \text{Ł}, \tag{4.4}$$

it becomes apparant that the more layers of equally small rate are available, the smaller the error between the steady rates computed by the RCC Algorithm and the achievable sending rates will be. We refer to Figure 4.2 for an illustration of this quantizer.

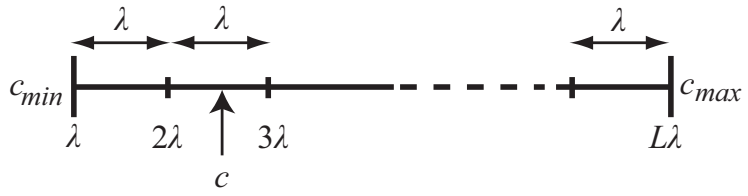

Figure 4.2: The optimal quantizer for a uniform source is a uniform quantizer.

To achieve the optimum performance of the RCC Algorithm, we should thus resort to encoding all video streams into FGS-like bitstreams, as we did in Chapter 3. However, as mentioned in Section 4.1, FGS encoding schemes that are available today present bad coding efficiencies and are not standardized, which makes them bad solution candidates in practice. That is why we will study in the next section how we can partially overcome the drawback that is represented by thicker layers.

4.3.2 Convergence periods

The duration of a convergence period depends in general on the RCC algorithm, and is called the *convergence time*. It is a random variable that we denote by δ and which has a probability distribution $p_\delta(x)$. This probability distribution can in general not be expressed analytically, but it can be inferred from statistics [78, 79]. From above, we know that in the case of transmitting a layered stream, the steady states that are achieved correspond in practice to the rates that can be achieved using the layers that are available. In a convergence period, the RCC will take controls $c(t)$ that typically do not correspond to any of these rates, but which need to be taken to move the transmission rate from one steady state to the next. Moreover, the controls taken during the convergence periods may exceed the rate of the new steady state by an overshoot rate of ϵ_{RCC} bps for a limited amount of time. Figure 4.3 sketches the evolution of the available channel rate $c(t)$ from an initial state in which layer 1 of average rate λ^1 can be transmitted, to a new stable state which allows for two layers of combined rate $\left(\lambda^1 + \lambda^2\right)$ to be delivered. In this example $\delta = (t_2 - t_1)$. Obviously we can straightforwardly transmit layer 1 up to time t_1, and both layers 1 and 2 starting from time t_2. However, in order to drive the channel up to rate $c(t_2)$, we need to send additional data in between the time instants t_1 and t_2, as indicated by the grey-shaded area. Failing to do so will leave the sending rate at λ^1. This in turn will not allow the sender to contend for the available transmission resources. As a result of which, these resources would get allocated to other contending flows. The additional available sending rate, as given by the grey-shaded area in Figure 4.3, can be filled by either one of the three following approaches:

- sending dummy data in order to probe the channel. Although this solution is always feasible it will have a negative impact on the goodput of the

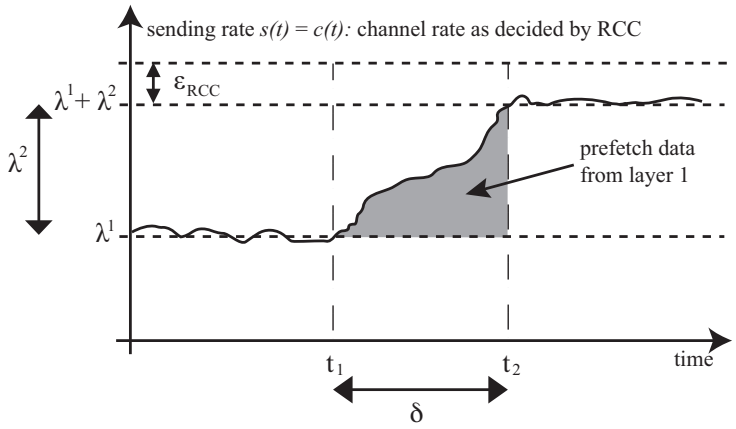

Figure 4.3: Illustration of the rate profile during a convergence phase.

application, meaning that network resources are used without yielding any application-level utility gain.

- sending redundant stream data such as Forward Error Correction (FEC). This is also feasible, but it reverts to the same situation as above if all data are in fact received.

- sending a part of the bitstream ahead of time. Using this scheme, all data that is transmitted will result eventually in an application-level utility gain, and it thus enhances the goodput of the application.

We opt for the latter solution, which consists in prefetching part of the video bitstream into a buffer at the client, whose decoder will only consume this part at a later time. This buffer undergoes thus a *filling phase* during the convergence time δ. As the stream we are considering is hierarchically encoded, layer 1 must be available at the decoder anyway in order for the corresponding layer 2 data to be decodable. Hence by prefetching layer 1 data ahead of time, the benefits are threefold:

- we can achieve the sending rate imposed by the RCC algorithm and increase the network stability,

- we make the decoding process more robust as layer 2 will be transmitted only when layer 1 is already available,

- we maximize the goodput of the streaming application by avoiding to send dummy or redundant data.

This general idea is similar to work that is presented in [54], with the exception that the authors therein do not consider a bounded prefetch delay. They imply that the client has unlimited buffering capacity and that the effective delay experienced by the client can grow arbitrarily large, which may represent practical limitations.

4.3.3 Computing the number of layers and their respective rates

We will use the concept of *smoothing* that we have outlined earlier in Section 4.2.2. Typically the exact shape of $c(t)$ in between t_1 and t_2 in Figure 4.3 is not known analytically, hence we apply the following integral upper bound on the possible traces of $c(t)$:

$$\int_{t_1}^{t_2} \left(c(\tau) - \lambda^1\right) d\tau \leq \left(\lambda^2 + \epsilon_{RCC}\right) \delta. \tag{4.5}$$

Conceptually this means that if, during the timespan δ we have $\left(\lambda^2 + \epsilon_{RCC}\right) \delta$ bits available to be transmitted in addition to the $\lambda^1\delta$ bits needed for layer 1 itself, then we can generate all sending traces $c(t)$ during the timespan δ that are bounded by Equation (4.5). This includes the traces generated by the RCC in order to converge to the next stable state.

The prefetch delay gives us a bound on the time horizon, and hence on the amount of data, which we can use to prefetch video data. Once the streaming session has started, the scheduler is not able to send data at any given time instant t that is further than D_p frames, respectively time units, into the future. From Figure 4.1 it can easily be seen that the maximum amount of data that is available for prefetching is given by $\lambda^1 D_p$. If this amount of data allows the sender to increase the transmission rate to $\left(\lambda^1 + \lambda^2\right)$ during the timespan δ, then layer 2 can be transmitted later on. Otherwise the sender continues to transmit only layer 1. Combining all of the above we get:

$$\lambda^2 \leq \lambda^1 \frac{D_p}{\delta} - \epsilon_{RCC}. \tag{4.6}$$

This is a condition on the layer thickness of layer 2, which depends on the allowed prefetch delay D_p as well as on the used RCC algorithm. Indeed, the latter can be characterized by its convergence time δ and the potential overshoot during convergence periods, ϵ_{RCC}. We can generalize this result straightforwardly to higher layers by recursion, i.e., for layer $(i+1)$ we have:

$$\lambda^{i+1} \leq \lambda^i \frac{D_p}{\delta} - \epsilon_{RCC}. \tag{4.7}$$

Using relation (4.7) with equality, i.e.,:

$$\lambda^{i+1} = \lambda^i \frac{D_p}{\delta} - \epsilon_{RCC}, \tag{4.8}$$

we can thus iteratively compute the minimum number of layers that are needed in order to make sure that we can converge from one stable state to the next, as well as their specific rates λ^l, once the statistical properties of the RCC Algorithm are known. If the channel rate can take on values in $[c_{min}, c_{max}]$, we set $\lambda^1 = c_{min}$ and apply Equation (4.8) iteratively until at the n^{th} iteration we have $\lambda^n > c_{max}$. The minimum number of layers that need to be encoded is thus given by $L = n - 1$ and their respective average source rates are given by $\mathcal{L} = \left\{\lambda^l\right\}_{l=1}^{L}$.

Note that if the probability distribution of δ is known, either analytically or as an approximation through statistics, we can dimension the above layers

by setting δ large enough so that is covers a high percentage of $p_\delta(x)$. Indeed, if $P(x < \delta)$ goes to 1, which represents a conservative approach, encoding the layers as outlined in this section will make sure that we can generate the sending rates dictated by the RCC algorithm with high probability.

Finally, we can also solve the inverse problem, which consists in finding the minimum prefetch delay D_p, given a set of layers $\mathcal{L}$ and a RCC Algorithm that is characterized by its convergence time δ and its potential overshoot rate ϵ_{RCC}. From above we immediately get:

$$D_p = \delta \cdot \left(\sup \left\{ \frac{\lambda^2 + \epsilon_{RCC}}{\lambda^1}, \frac{\lambda^3 + \epsilon_{RCC}}{\lambda^2}, \ldots, \frac{\lambda^L + \epsilon_{RCC}}{\lambda^{L-1}} \right\} \right). \tag{4.9}$$

It is important to note that, although multiple finer layers tend to increase the overall stability of the RCC Algorithm and allow for smaller prefetch delays, they also typically exhibit a worse compression efficiency as compared to coarser layers. This will be illustrated in Section 4.6 where we present our simulation results. Moreover the complexity of scheduling algorithms is typically increased when more layers are available for transmission.

4.4 Practical scheduling aspects

In this section we will discuss some important practical aspects of the smoothing approach that we have outlined in Section 4.3. Let us consider once more the simple scenario that we have used earlier in order to outline some practical issues that may arise. We suppose that we have reached a steady state at rate λ^1 and that the RCC algorithm converges to a new steady state at rate $(\lambda^1+\lambda^2)$. From the previous section, we know that if the layers are encoded so as to satisfy relation (4.7), then we can generate with high probability the sending rates that match the control decisions taken in the convergence period. For the sake of clarity, we will set the playback delay D to 0 in what follows, without loss of generality. Note that the importance of D and its optimal computation will be analyzed in the following chapter of this thesis [76].

4.4.1 Repeatable prefetching

It is important to notice that, in the worst case, if we succeeded to attain the transmission rate $(\lambda^1 + \lambda^2)$, we have exhausted the complete prefetch delay D_p for layer 1. In that case, the prefetch buffer for layer 1 has been completely filled during the timespan δ, corresponding to a *filling phase*. By prefetching at most D_p frames, we have thus been able to ramp up the transmission rate to $(\lambda^1 + \lambda^2)$, and from there on we can transmit both layers if the network status does not change for some time. Clearly, if the same operation is to be repeated at a later time, each *filling phase* needs to be followed as soon as possible by a corresponding *draining phase*, which liberates the previously filled prefetch buffer.

If after the *filling phase* of layer 1, the rate $(\lambda^1 + \lambda^2)$ has been reached and should be sustained, the *draining phase* of the prefetch buffer for layer 1 data needs to be accompanied by another *filling phase* of layer 2 data. During this phase, as much data as is drained from the prefetch buffer of layer 1 will have

to be prefetched into the buffer of layer 2 – keeping the overall prefetch buffer for both layers 1 and 2 at a constant filling level.

So, once the transmission rate of $(\lambda^1 + \lambda^2)$ is reached, the scheduler should in a first step make sure that for both layers all frames up to k are scheduled. Then it should preferentially select frames from the higher layer(s) in order to fill the rate surplus.

Another case that needs to be considered in this light is the settling into a stable state that corresponds to a rate that can not be achieved by a source rate constructed from the set given in Eq. (4.1). Suppose that the highest layer that can be transmitted at a given time is given by the index l^*, and that the RCC Algorithm converges to a steady state c, $\lambda^{l^*} < c < \lambda^{l^*+1}$. Using the mechanism outlined above, the scheduler will try to achieve any rate decided by the RCC Algorithm by exploiting the prefetch buffers for each of the layers up to l^*. The above loop will end up by first exploiting all of the available prefetch buffers. At this point, the actual transmission rate has not increased to the point where a further layer could be transmitted, but has settled at c. The scheduler will be unable to sustain the channel rate decided by the RCC Algorithm. It needs to drop its actual sending rate and settle into a steady rate that equals to the average video source rate that is closest to the channel rate, as discussed in Section 4.3.1. The actual sending rate for the l^* selected layers will read as:

$$s(t) = v = \sum_{l=1}^{l^*} \lambda^l. \tag{4.10}$$

However, as all the prefetch buffers are completely filled, the scheduler would no longer be able to increase its sending rate at any later time instant. Therefore the scheduler needs to settle into a steady state which is slightly smaller than that in Equation (4.10), in order to allow for a *draining phase* of the prefetch buffers, hence:

$$s(t) = \sum_{l=1}^{l^*} \lambda^l - \epsilon_\lambda \tag{4.11}$$

Note that as soon as an amount of data equal to $\lambda^* D_p$ has been drained from the aggregate prefetch buffer consisting of the prefetch buffers for layers 1 up to l^*, the scheduler can try to follow the decisions of the RCC Algorithm once more. According to Equation (4.7), this free prefetch buffer will allow the sending rate to be potentially ramped up to the average video rate given by

$$v = \sum_{l=1}^{l^*+1} \lambda^l, \tag{4.12}$$

which allows the next layer to be transmitted. In what follows we call this condition the *draining condition* for the prefetch buffer of layer l^*. In absence of other changes on the used channel, this condition will be satisfied $(\lambda^* D_p) \,/\, (\lambda^* - \epsilon_\lambda)$ time units after the rate was set to (4.11). This timespan corresponds to the time it takes to drain the previously filled prefetch buffer in the worst case where it was completely filled. In practical scenarios, the factor ϵ_λ should be kept small, so that the sending rate is still close to the steady source rate of the transmitted layers. While this means that the prefetch buffer takes a longer

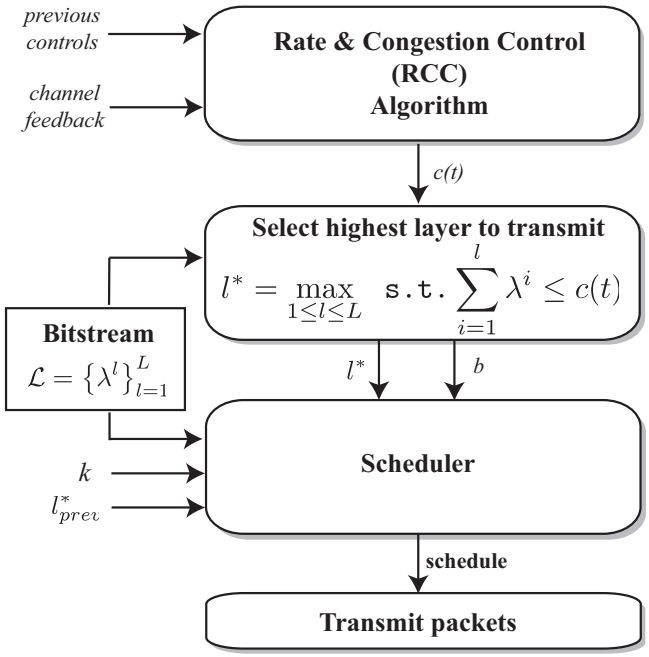

Figure 4.4: Overview of a typical sender architecture including RCC module and adaptive scheduler.

time to be completely drained, it also makes sure that the sender does not allow a large rate ϵ_λ to be available for contention by other flows in the network.

The considerations that we have outlined in this section do not apply if D_p is very large as the prefetching process will always be repeatable.

4.4.2 Implementation

Figure 4.4 gives a generic view of a sender architecture. Based on channel feedback and previously taken rate controls, the RCC algorithm decides to send data at rate $c(t)$ during the next control interval of length Δ_c. Given this new control, and knowing the set of available rates $\mathcal{L}$, the sender computes the index l^* of the highest layer that can be sustained using the sending rate $c(t)$. This is computed as:

$$l^* = \max_{1 \le l \le L} \texttt{s.t.} \sum_{i=1}^{l} \lambda^i \le c(t). \tag{4.13}$$

This index is provided as input to the scheduler, along with l^*_{prev}, which is the index of the highest layer that was selected in the previous control interval. The scheduler also knows the frame index k which indicates up to which frame all of the selected layers need to be scheduled by the end of the current control interval, in order to meet all timing constraints. For control interval j, k reads as:

$$k = j \cdot \Delta_c \cdot f, \tag{4.14}$$

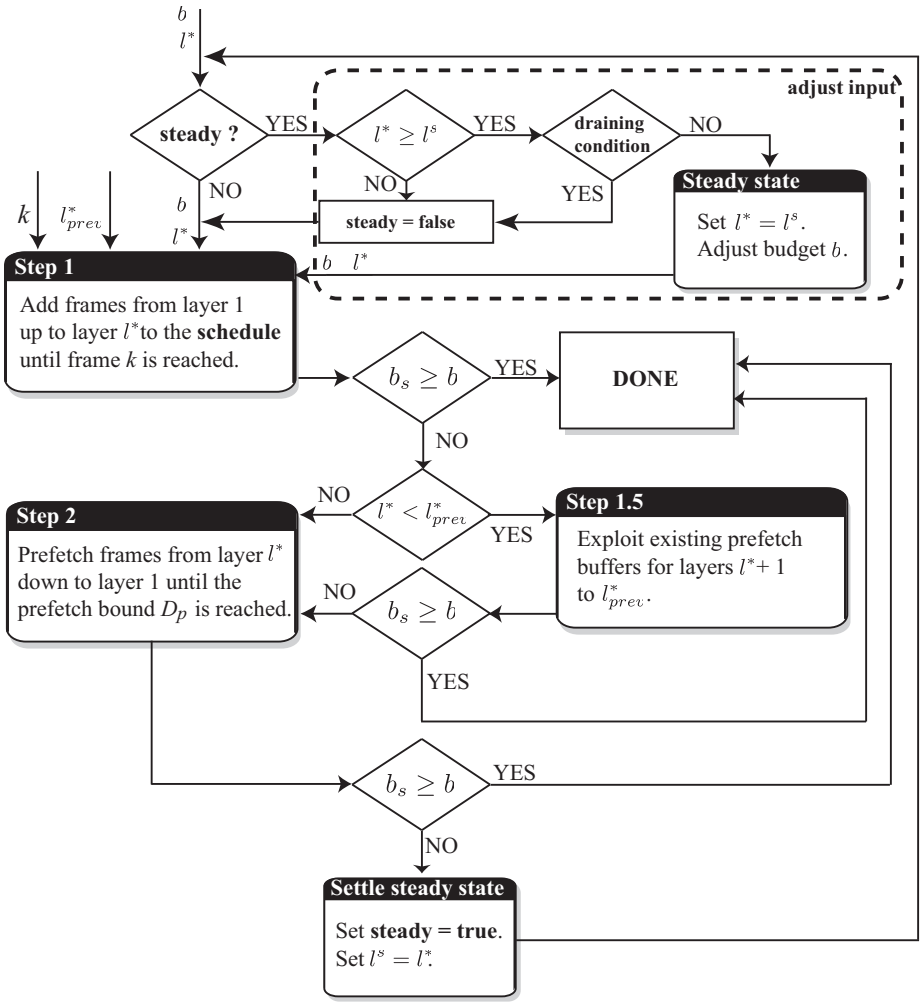

Figure 4.5: Workflow of the adaptive scheduler that implements our algorithm.

where f denotes the stream's framerate in frames per second. Finally, the control decision $c(t)$ is communicated to the scheduler in the shape of an available bit budget b for the schedule that is to be computed.

Figure 4.5 provides a more detailed view of the scheduler's inner life. Here, b_s denotes the number of bits that has already been added to the schedule. Note that we allow $b_s > b$, as we choose to add only *complete* frames of each layer to the transmission schedule. The scheduler keeps a state variable that indicates whether it needs to wait for the prefetch buffers to be emptied or whether it can try to ramp up the rate as dictated by the RCC algorithm. This variable is denoted as *steady* in Figure 4.5 and it is initialized to *false* for all streams that start transmission. This flag allows the scheduler to verify the *draining condition* that was discussed in the previous Section. The work flow of the scheduler is quite straightforward:

- In Step 1, all the frames up to k are added to the schedule, for all of the selected layers from layer 1 to layer l^*.
- In Step 2, we allocate the surplus of available rate by prefetching data from higher layers first. As mentioned earlier, this makes sure that the prefetching operation used to increase the rate is repeatable throughout streaming session.
- There is an intermediate Step 1.5 which is performed if the number of layers chosen for transmission in the current control interval is smaller than the number of layers l^*_{prev} that has been selected in the previous control interval. In that case, the scheduler checks if it can still allocate frames for previously sent layers, up to the frame indexed by k. Note that this might be possible if a prefetch buffer has been built up for layer l^*_{prev} earlier on. In case this step fails, it is rolled back.
- If at the end of Step 2, the bit budget has not been exhausted, then the computed rate $c(t)$ cannot be matched with a sending rate. This happens if the RCC computes a stable state for which there is no matching source rate, or if the prefetching fails due to an unexpectedly long convergence time of the RCC. In that case, the sender settles into a steady state and sets its transmission rate to (4.11). The complete scheduling loop is rolled back and run again with an adjusted bit budget (i.e., an adjusted rate $c(t)$) that accounts only for the layers that have been selected for transmission and further allows for the prefetch buffers to be drained if necessary.

  The scheduler stores the highest selected layer as $l^s = l^*$ and will adjust the sending rate down to the same rate (4.11) during the subsequent control intervals, until the draining condition is fulfilled. If that is the case, the scheduler is again able to ramp up the sending rate high enough to potentially deliver an additional higher layer.

4.5 Application to Media-Friendly Rate Allocation

So far we have presented the results of this chapter in a generic form. By choosing δ and ϵ_{RCC} appropriately, the rates in the layer set $\mathcal{L}$ can be computed to match a wide variety of Rate and Congestion Control Algorithms.

In our simulations, we will however focus our attention on applying the findings from the current chapter to the Media Friendly Rate and Congestion Control Algorithm that we presented and validated in Chapter 3. This algorithm provides the capability to inherently distinguish the transported streams and their rate needs, while allocating smooth rate profiles that allow for high and steady quality video delivery. One of the main assumptions in Chapter 3 has been that any rate that can be computed by the RCC Algorithm can be exactly matched with a corresponding video rate. This has been the case for FGS-like video encodings. Such encodings generate a class of traffic that can take on any rate within a finite rate interval. If a video sender is however constrained to sending a discrete subset of complete layers for each frame, the traffic it generates no longer has that property and can typically only take on L distinct sending rates, corresponding to the L available video source rates.

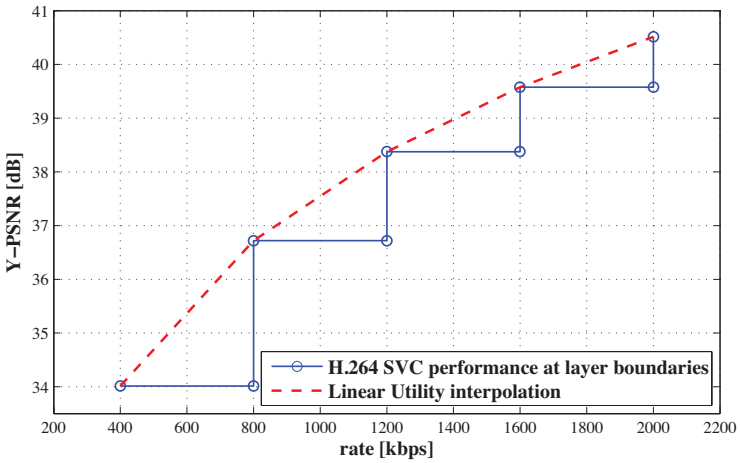

Figure 4.6: Utility function for the SOCCER sequence (CIF @ 30Hz), encoded into 5 SNR layers using H.264 SVC.

We will show in the remainder of this Chapter that by setting the encoding parameters and the prefetch bounds as indicated by our results, the scheduler will be able to generate the sending rates needed to drive the system into a stable state, by adapting the source rates through the use of smoothing.

4.5.1 Video Utility for layered streams

We have encoded the SOCCER (CIF, 30Hz) and CREW (4CIF, 30Hz) test sequences into a set of layers that provide SNR scalability using the H.264 SVC reference codec. Decoding additional layers will thus decrease the distortion in the received signal. We have considered 3 encodings in particular:

- SOCCER (coarse) encodes the SOCCER CIF@30Hz sequence into 5 layers. Each layer having an average bitrate of 400kbps. See Figure 4.6 for an illustration of its rate/distortion performance.
- SOCCER (fine) encodes the same sequence into 8 layers. The base layer having an average bitrate of 400kbps, while all of the enhancement layers provide an average rate of 200kbps, see Figure 4.7.
- CREW encodes the CREW 4CIF@30Hz sequence into 8 layers. The base layer having an average bitrate of 1Mbps, while all of the enhancement layers provide an average rate of 300kbps, see Figure 4.8.

In order to generate concave and continuous Utility functions for these streams, we interpolate the utility values in between the layer boundaries by linear segments. According to our scheduling algorithm, if a layer l^* is being transmitted, higher rates are generated by prefetching information from l^* until $l^* + 1$ can be delivered. We hence motivate our choice of Utility functions by the fact that due to the hierarchical relationship between layers l^* and $l^* + 1$, the Utility of layer l^* increases linearly with the amount of prefetched layer l^* data.

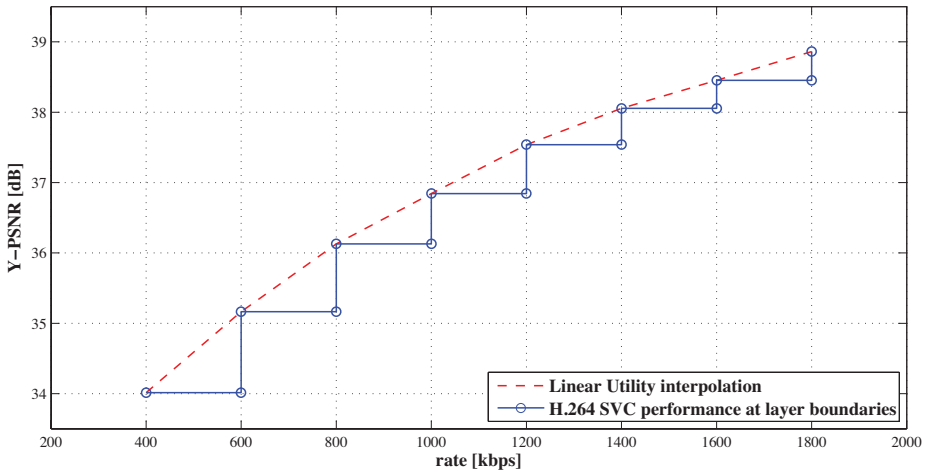

Figure 4.7: Utility function for the SOCCER sequence (CIF @ 30Hz) , encoded into 8 SNR layers using H.264 SVC.

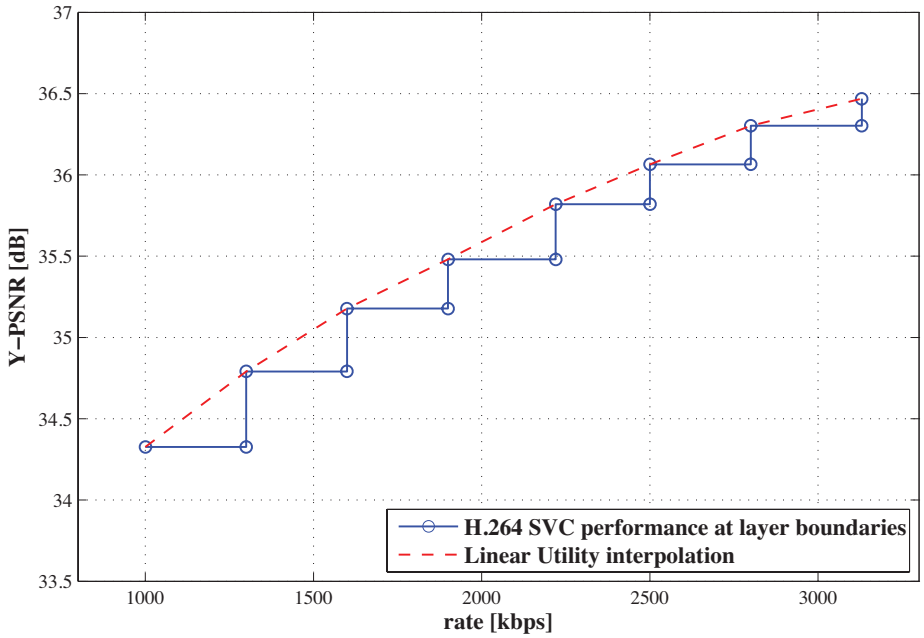

Figure 4.8: Utility function for the CREW sequence (4CIF @ 30Hz) , encoded into 8 SNR layers using H.264 SVC.

As discussed in Section 4.4, the RCC Algorithm may converge to a rate allocation which can not be achieved by all of the flows in the etwork.Whenever this happens, the corresponding flows will send at the highest rate they can achieve, which is smaller than the rate decided by the RCC Algorithm. It is important to note that whenever this happens, the resulting rate allocation will be *sub-optimal*. The aggregate Network Utility will thus not be necessarily maximized, and consequently the fairness property that we have shown in Chapter 3 may not always be enforced.

We will illustrate using our simulations that these situations can be avoided by using more layers of smaller rate, validating the results of Section 4.3, or by allowing large prefetch delays. The latter allows more flexibility in the sending rates, at the expense of a larger used buffer at the receiving end of the stream and an inevitable increase in end-to-end delay.

4.5.2 Convergence time

One interesting feature of our RCC Algorithm is that due to the smooth rate profiles it generates, the overshoot rate ϵ_{RCC} is in practice negligible. This leaves the convergence time δ to be computed. Our findings indicate that for our RCC Algorithm, δ will depend heavily on the scenarios that are considered and mostly on the Utility functions that are used. According to our results in Chapter 3, the system will converge slower when the Utility functions that are used have a slope that is less steep. This indicates that the convergence time depends on the nature of the traffic that is being transmitted. As this situation makes it hard to estimate a meaningful pdf $p_\delta(c)$, we have resorted to computing the maximum expected value of δ for each of the considered simulation scenarios. In order to do so, we have run each scenario first using the appropriate Utility functions, but by sending traffic that is always able to match the computed channel rates, as in the FGS scenario from Chapter 3. We have then measured the durations it has taken for the RCC to bridge the rate gap between two layers. The maximum of these values is then selected as the convergence time δ for the considered scenario. The such learned value provides a good starting point in order to validate our findings. Although this learning phase is not achievable in practice, a conservative estimate on the convergence time can always be extracted from experiments.

4.6 Simulations

4.6.1 Setup

We have extended the NS-2 implementation of our media friendly rate and congestion control algorithm to include the scheduler depicted in Figure 4.5. The average loss rate in the stable state π is set to 5% for all simulations and κ equals 1. Once more, we use the network topology that is shown in Figure 3.7. Each sender delivers a given video stream to its client, as given in Table 4.1.

There is one important difference to our new scheduler. When in Chapter 3 the sender would send exactly one finely adapted GOP worth of video data during each control interval, this is now no longer true. During each control interval, the sender will try to match the current rate control by possibly

prefetching data from frames whose decoding deadline will only expire later. Hence, we no longer add the size of the transmitted GOP to each packet, but rather the number of bytes b_s that have been scheduled on aggregate during the current control interval. The receiving client will count the number of bytes it receives from the batch of data that was sent, and transmit the resulting byte count together with b_s as feedback to the sender. As the sender knows the (constant) duration of each control interval, using this feedback it will be able to recover the previously taken rate control. Moreover it will be able to evaluate the congestion signal, which is all the input that is needed in order to evaluate the controller's rate update equation. For all of the following simulation runs, we have set the factor ϵ_λ in Equation (4.11) to 5% of the selected source rate. This means that once a sender needs to fall back to transmitting at the video source rate $v = \sum_{l=1}^{l^*} \lambda^l$, because it fails to achieve the steady state rate computed by the RCC Algorithm, the actual transmission rate is computed as:

$$s(t) = 0.95 \cdot \sum_{l=1}^{l^*} \lambda^l. \tag{4.15}$$

As outlined in Section 4.4.1, this allows the potentially full prefetch buffers to be drained, while keeping the sending rate close to the selected source rate.

4.6.2 Sub-optimal behavior and fairness

In the first set of simulations we consider some very simple topologies in order to illustrate some key characteristics that stem directly from our findings. Throughout this set of simulations, we have set $D_p = 90$ frames, unless otherwise stated. This is in line with equation (4.9). Indeed the convergence time we expect for these simulations was computed to be 3 seconds (90 frames), and the maximum ratio between the rates of 2 subsequent layers is 1.

In Scenario 1, we use the sender-receiver pairs 1 and 2, which transmit the coarse layered version of the SOCCER test sequence. We set the bottleneck capacity to 3.5Mbps. The resulting sending rates and the received rates are plotted in Figures 4.9 and 4.10 respectively. In addition we plot the number of transmitted layers for each sent frame in Figure 4.11. As expected, the two streams end up sharing the available network resources adequately, and each flow gets an equal average rate which allows for the transmission of all layers up to layer 4 once the steady state is reached.

In the next simulation run, which we describe by Scenario 2, we consider the setup to be exactly the same as in Scenario 1, but this time we set the bottleneck capacity to 3.7Mbps. The resulting sending rates are plotted in Figure 4.12 and the number of delivered layers for each of the streams is shown in Figure 4.13.

$S_{1,2} \rightarrow R_{1,2}$	SOCCER, CIF @ 30Hz (coarse)
$S_{3,4} \rightarrow R_{3,4}$	SOCCER, CIF @ 30Hz (fine)
$S_{5,6} \rightarrow R_{5,6}$	CREW, 4CIF @ 30Hz

Table 4.1: Distribution of the test video sequences among the sender-receiver pairs.

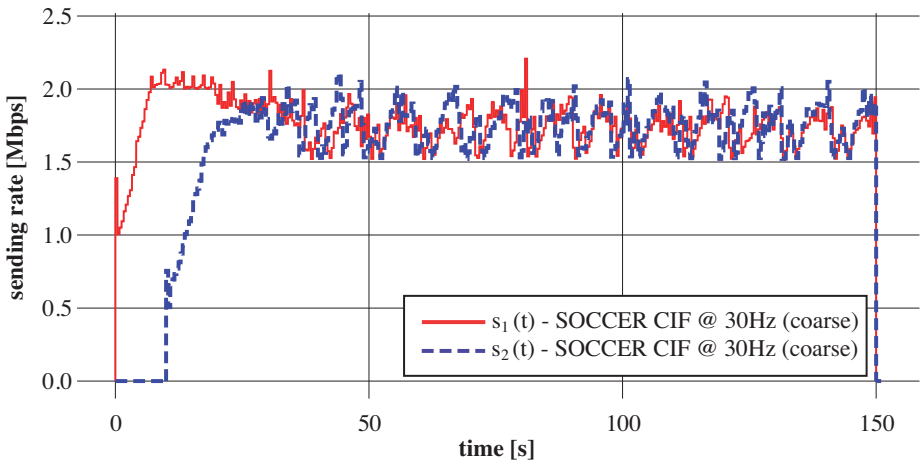

Figure 4.9: Allocated transmission rates for Scenario 1.

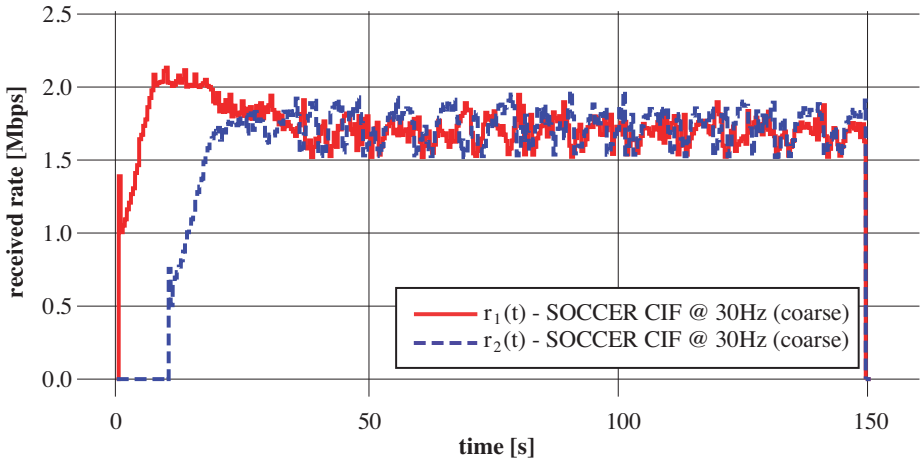

Figure 4.10: Received rates for Scenario 1.

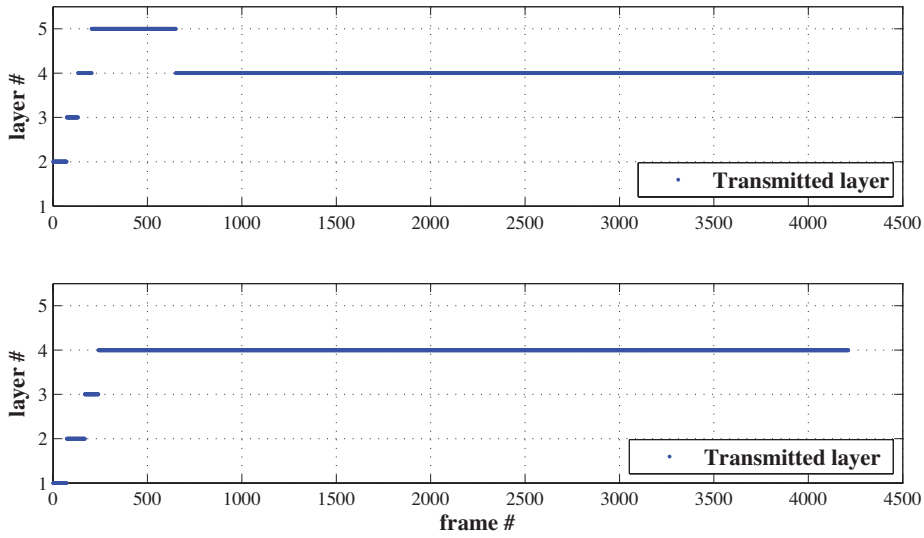

Figure 4.11: Transmitted layers for senders 1 (top) and 2 (bottom) in Scenario 1.

Obviously, the small change in bottleneck bandwidth has an important effect on the steady state distribution. Even though both streams are characterized by the same Utility function, not all of the competing streams can transmit at the rates that would allow for a fair and adequate sharing of the network resource. Hence the RCC algorithm ends up by allocating more rate to stream 1 than to stream 2, in an effort to maximize the resource utilization. This result illustrates our discussion in Section 4.5: if the rates that are available from the set of video layers $\mathcal{L}$ do not match all of the possible steady state rates, and if the prefetch delay is not large enough to smooth the available source rates, then the sender will have to converge to a suboptimal state in which the Utility-weighted fairness property can no longer be enforced all the time.

In Scenario 3 we consider once more the setup from Scenario 2, but we remove any bound on the prefetch delay D_p: the scheduler can prefetch any data from the future if this is needed to fill the rate dictated by the RCC Algorithm. This allows each of the senders to allocate much smoother rate profiles, thus enhancing the stability from a networking perspective, as shown in Figure 4.15. However, as illustrated in the number of layers that are delivered for each frame and each of the two streams, see Figure 4.15, this does not address the fairness issue that we have mentioned. The only way to alleviate this situation is to consider finer layers in order to be able to match the steady state rate that is achieved.

We illustrate this situation in Scenario 4, where the bottleneck capacity is again set to 3.7Mbps, but this time we let the sender-receiver pairs 3 and 4, transmitting the fine layered version of the SOCCER sequence, share the bottleneck. The resulting transmission rates are shown in Figure 4.16 and indicate that using finer layers, we can indeed increase the stability of the algorithm. Moreover, as depicted in Figure 4.17, both streams end up being treated in a

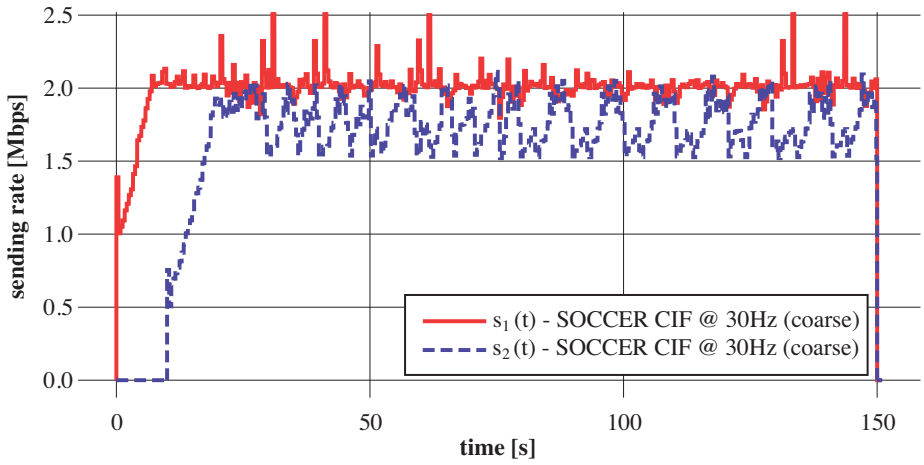

Figure 4.12: Allocated transmission rates for Scenario 2.

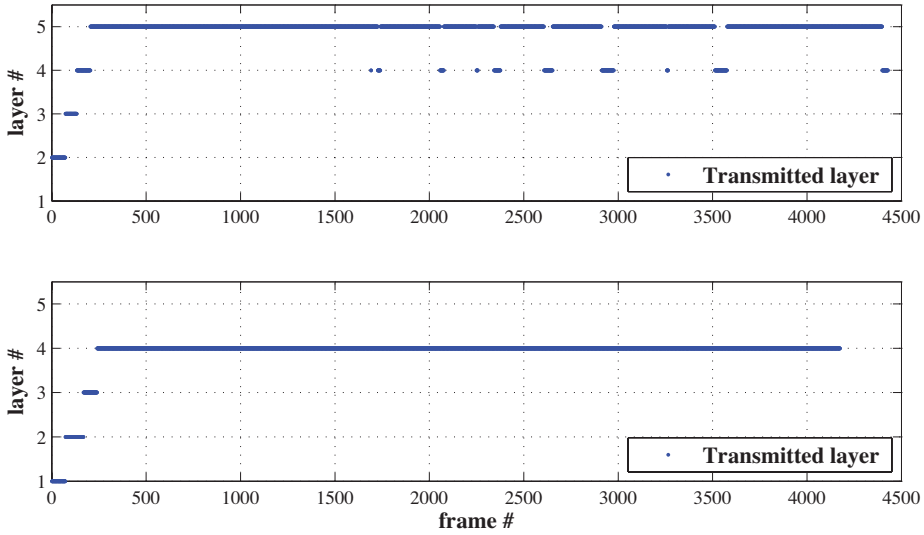

Figure 4.13: Transmitted layers for senders 1 (top) and 2 (bottom) in Scenario 2.

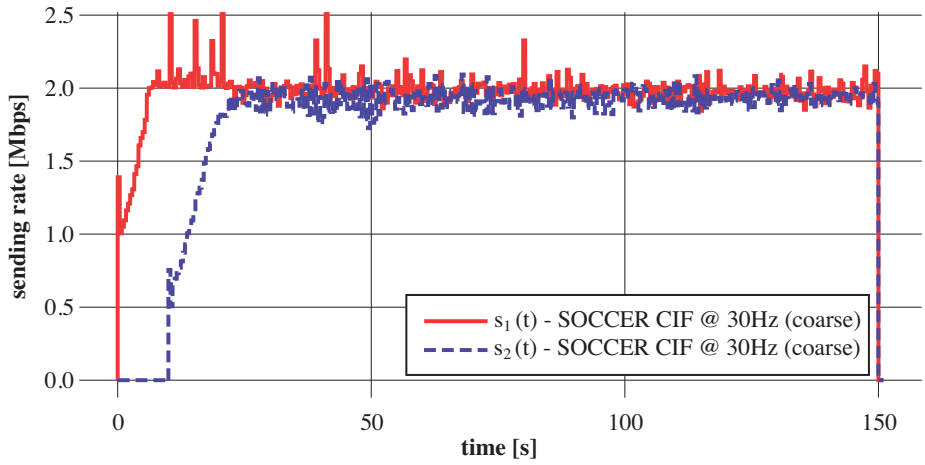

Figure 4.14: Allocated transmission rates for Scenario 3.

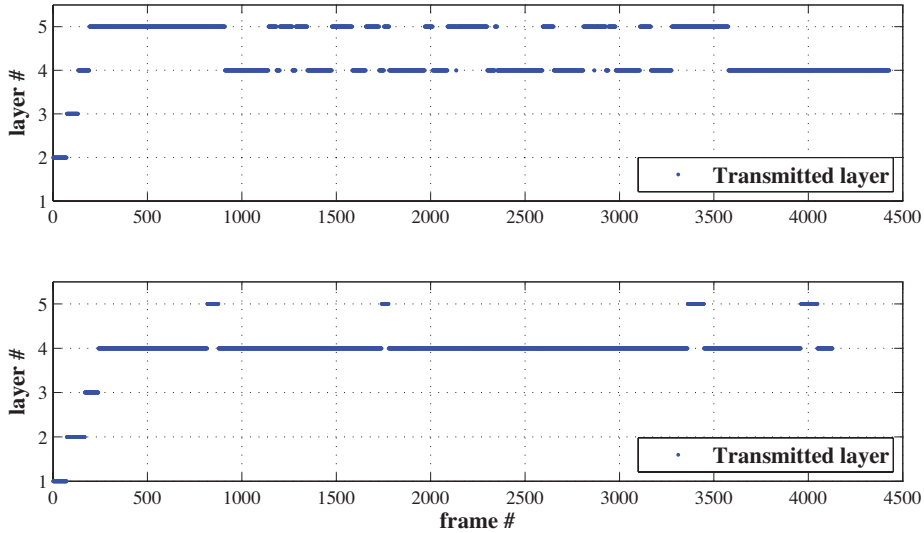

Figure 4.15: Transmitted layers for senders 1 (top) and 2 (bottom) in Scenario 3.

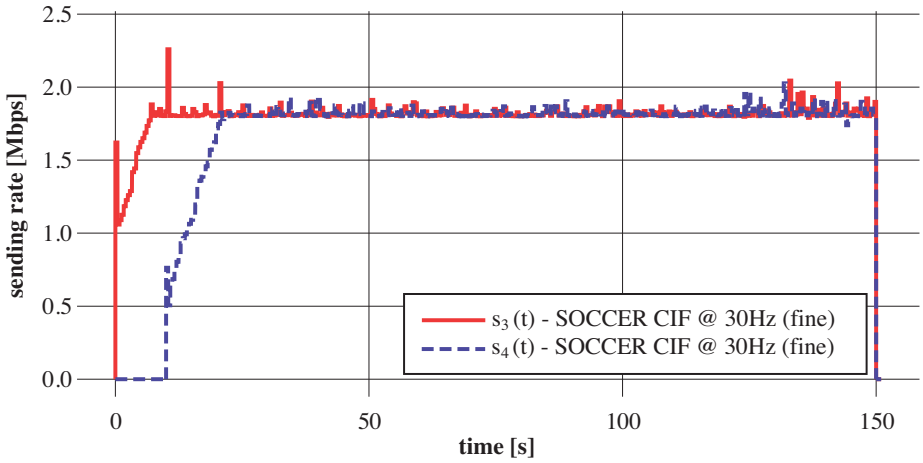

Figure 4.16: Allocated transmission rates for Scenario 4.

similar way and fairly share the resource, as the optimal rate controls can now be matched by a corresponding video rate.

The mean Y-PSNR of the transmitted streams and the mean loss in Y-PSNR due to congestion losses are given for reference in Table 4.2. As laid out in Chapter 3, the proposed rate and congestion control algorithm is characterized by steady state losses which are tightly bounded by the factor $\pi = 5\%$. Moreover we have implemented an *active node* at the bottleneck router which discards first the packets of least importance whenever it has reached its forwarding capacity.

4.6.3 Adaptation to network dynamics

In the second set of simulations, we provide results on the performance of our proposed system in a more dynamic network setting. We consider a bottleneck capacity of 6Mbps and set the prefetch bound first to $D_p = 130$, which allows to satisfy relation (4.9). The bottleneck is shared by 4 sender-receiver pairs: senders 3 and 4 transmit the fine layered version of the SOCCER test sequence, while senders 5 and 6 transmit the CREW sequence. The simulation run lasts for 120 seconds. During 30 seconds, and starting 50 seconds into the simulation, we inject some constant bitrate (CBR) background traffic into the bottleneck, thus reducing its forwarding capacity for the media streams. The CBR rate is set to 1Mbps. We refer to this simulation run as Scenario 5.

Figure 4.18 shows the resulting sending rates for all of the 4 streams. Most importantly, it can be noticed that our system is still able to distinguish the 2 types of traffic on the bottleneck through the use of their Utility functions. It allocates the rates accordingly in an effort to maximize the overall Network Utility. Moreover the rates converge quickly to a new stable state once the bottleneck capacity drops, and recover to the initial steady state as soon as the background traffic is switched off.

The number of transmitted layers for each frame is plotted in Figure 4.19 for senders 3 and 4, and in Figure 4.20 for senders 5 and 6 respectively. On average the similar streams get the same number of layers throughout the simulation

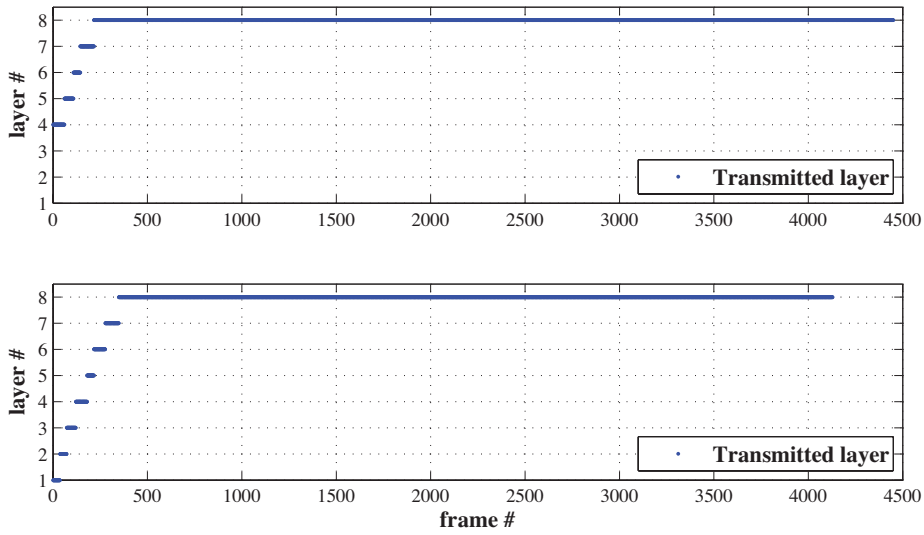

Figure 4.17: Transmitted layers for senders 3 (top) and 4 (bottom) in Scenario 4.

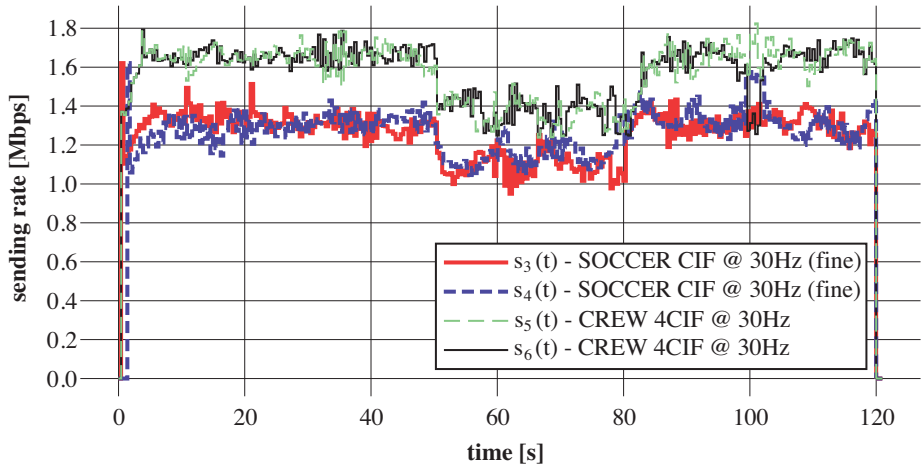

Figure 4.18: Allocated transmission rates for Scenario 5.

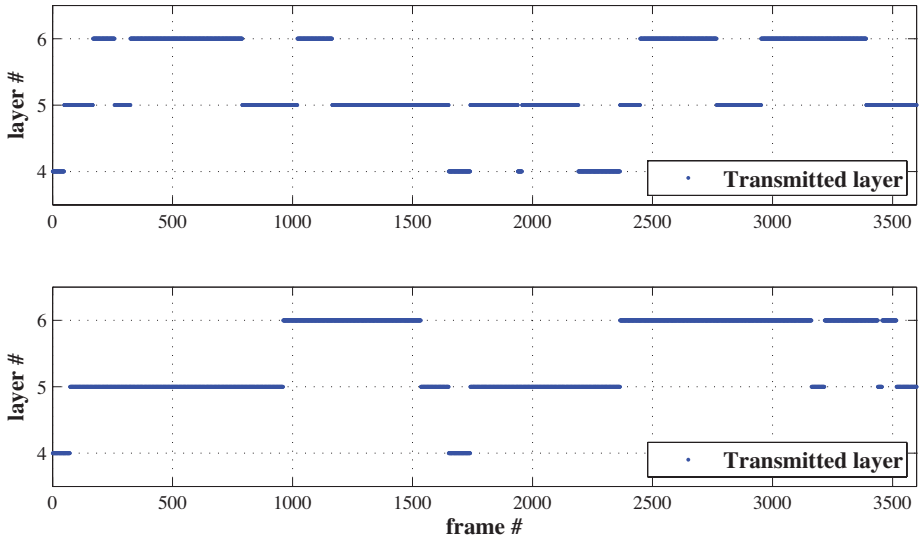

Figure 4.19: Transmitted layers for senders 3 (top) and 4 (bottom) in Scenario 5.

run, which is confirmed by the resulting Y-PSNR values listed in Table 4.2.

In Figure 4.21 we illustrate the filling level of the prefetching buffers for the 3 layers that are delivered by sender 6. This shows how higher layers are preferentially used for prefetching in an effort to keep the prefetching process repeatable. It can also be seen that around the 100th second of the simulation run, no data for layer 3 is sent, but as the corresponding information has been prefetched earlier, all of layer 3 is still transmitted for all of the frames after the bottleneck capacity is restored.

Finally, in Scenario 6, we consider the same topology as in Scenario 5, but we illustrate the behavior of the system if the layer rates and the chosen prefetch delay do not comply to the condition given in Equation (4.9). This is achieved by setting a smaller prefetch delay $D_p = 100$ for senders 5 and 6, which are streaming the CREW sequence. The resulting sending rates for each of the 4 streams are depicted in Figure 4.22. Both senders 5 and 6 start to transmit at relatively high sending rates. Once the background traffic sets in, it becomes apparent that the senders do not have enough data in their prefetch buffers in order reach the previously achieved stable state. This results from the fact that D_p is too small, or conversely that given D_p, the rate of the encoded layers is too large. Once the background traffic is switched off, senders 3 and 4 are able to contend for the available bottleneck bandwidth and transmit the complete set of available layers. Meanwhile, senders 5 and 6 are stuck at transmitting layer 1. Although the bottleneck is not fully used, they alternate filling and draining phases of their prefetch buffers, but are never able to reach a sending rate that would allow for the transmission of further layers.

The number of transmitted layers for each frame is plotted in Figure 4.23 for senders 3 and 4, and in Figure 4.24 for senders 5 and 6 respectively. On average the similar streams get the same number of layers throughout the simulation

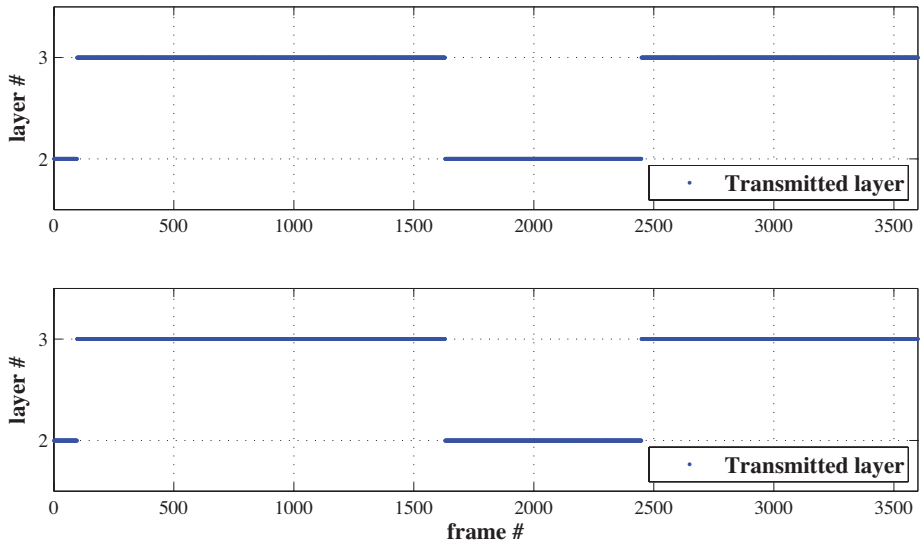

Figure 4.20: Transmitted layers for senders 5 (top) and 6 (bottom) in Scenario 5.

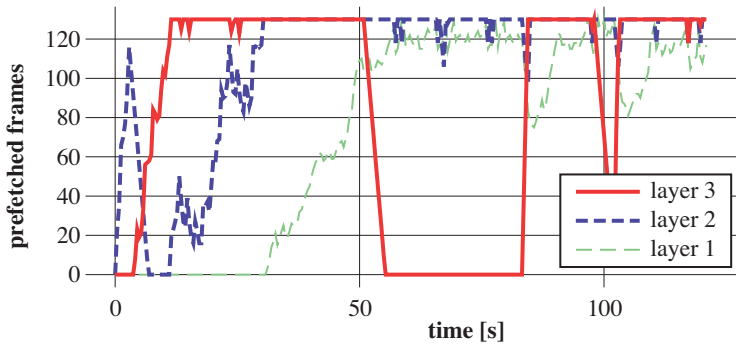

Figure 4.21: Evolution of the prefetching buffer of sender 6 in Scenario 5.

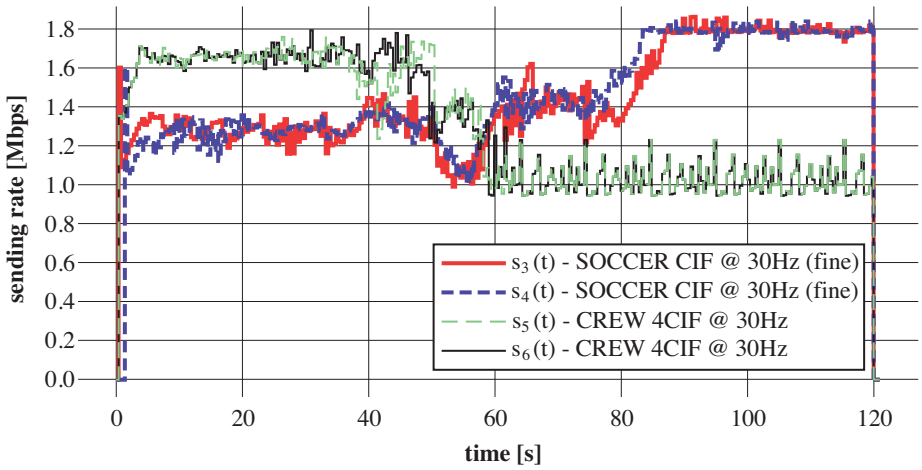

Figure 4.22: Allocated transmission rates for Scenario 6.

run, which is again confirmed by the resulting Y-PSNR values listed in Table 4.2. It can be noticed that the average losses measured at the receivers are somewhat lower as compared to the previous Scenario. This is due to the fact that during the last part of the simulation the bottleneck is not fully used, and hence there are no congestion losses during the corresponding period.

4.7 Conclusions

In this Chapter we have considered the problem of adapting the source rates that are generated by layered video streams, to the available channel rates. By proposing to use smoothing in order to achieve this adaptation, we have derived bounds on both the encoding rates of the video layers and the prefetch delay that can be used for smoothing. These conditions depend on parameters that reflect the behavior of the underlying Rate and Congestion Control Algorithm. We have discussed practical scheduling aspects related to the transmission of these video layers. Furthermore, we have given a practical implementation of a scheduler that meets all of our design criteria.

Finally, we have illustrated that using layers that are properly encoded to the precomputed rates, it is possible to extend the media friendly rate allocation algorithm from Chapter 3 to the case of layered streams. The hurdle that needs to be taken in order to make this possible lies in the fact that a layered video stream does in general not generate traffic that can achieve any rate within a given interval. We address this problem by allocating the available sending rates in a meaningful way on the one hand, and by correctly scheduling that available data on the other hand. Our results indicate that in general, having multiple layers of smaller average bitrate is highly beneficial in terms of both network stability and incurring delay. Although the coding performance of the video sequences is degraded when more layers of smaller average rate are encoded, there is an important trade-off to be made in terms of network stability and delay.

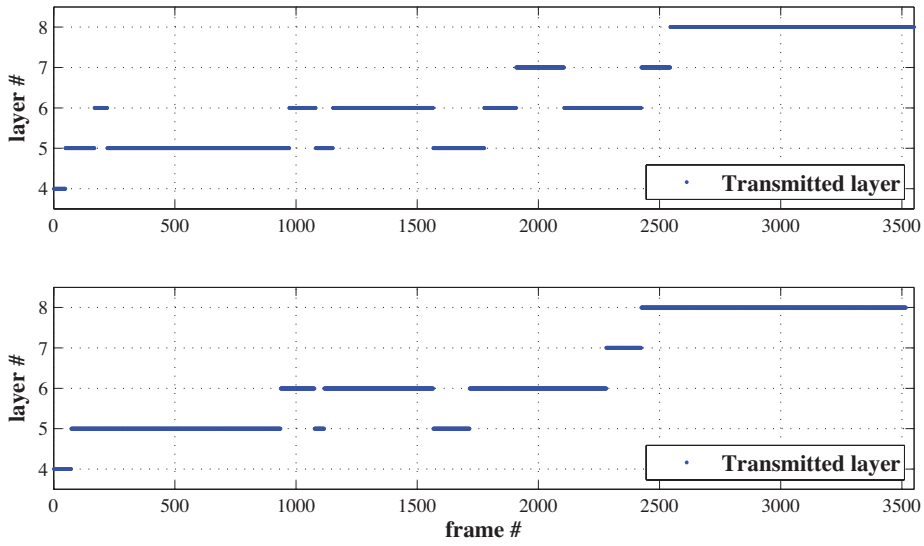

Figure 4.23: Transmitted layers for senders 3 (top) and 4 (bottom) in Scenario 6.

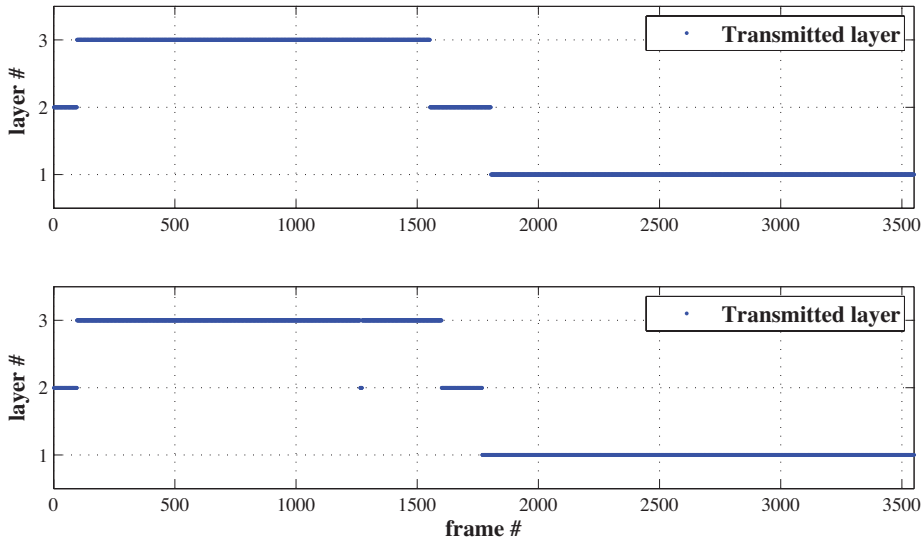

Figure 4.24: Transmitted layers for senders 5 (top) and 6 (bottom) in Scenario 6.

Scenario 1	Mean Y-PSNR at sender	Mean loss at receiver
$S_1 \rightarrow R_1$	38.08 dB	0.96 dB
$S_2 \rightarrow R_2$	37.85 dB	0.98 dB
Scenario 2		
$S_1 \rightarrow R_1$	39.44 dB	1.02 dB
$S_2 \rightarrow R_2$	39.00 dB	0.97 dB
Scenario 3		
$S_1 \rightarrow R_1$	39.15 dB	0.96 dB
$S_2 \rightarrow R_2$	38.89 dB	1.10 dB
Scenario 4		
$S_3 \rightarrow R_3$	38.79 dB	0.90 dB
$S_4 \rightarrow R_4$	38.74 dB	0.82 dB
Scenario 5		
$S_3 \rightarrow R_3$	37.59 dB	1.15 dB
$S_4 \rightarrow R_4$	37.72 dB	1.18 dB
$S_5 \rightarrow R_5$	35.12 dB	0.40 dB
$S_6 \rightarrow R_6$	35.11 dB	0.51 dB
Scenario 6		
$S_3 \rightarrow R_3$	38.18 dB	0.21 dB
$S_4 \rightarrow R_4$	38.21 dB	0.19 dB
$S_5 \rightarrow R_5$	34.72 dB	0.19 dB
$S_6 \rightarrow R_6$	34.74 dB	0.18 dB

Table 4.2: PSNR quality for the different streaming scenarios [dB].

Chapter 5

Playback Delay and Buffer Optimization

5.1 Background

Due to the rapid evolutions in consumer electronics, the possibility to adapt to client preferences or to customize services becomes predominant in multimedia applications. We consider in this chapter the problem of the simultaneous delivery of a scalable media stream to heterogeneous clients that present different computing capabilities, different access bandwidths, or different user requirements. Each one of these clients selects to receive an appropriate subset of scalability layers, providing the needed and decodable stream. A typical example of such a system is given in Figure 5.1, where a streaming server connects to heterogeneous clients directly or through a streaming proxy and broadcasts a stored scalable media stream. Scalable video streaming systems have generally to respect a bottleneck channel bandwidth, which prevents the immediate delivery of the media data to all the clients. This limitation may be imposed by channel or disk bandwidth constraints, admission control or pre-determined traffic specifications (e.g., TSPEC in the 802.1e wireless standard). Client buffering capabilities may help to smooth the discrepancies between the video source rate and the available bandwidth with a sustained quality, at the price however of an increased playback delay at the client. It becomes therefore important to derive efficient packet scheduling strategies such that smooth playback can be ensured at each client and that the overall quality of service is maximized.

Several works have studied the problem of efficient packet scheduling in different streaming scenarios. The problem of minimizing the playback delay and buffering needs for a single receiver and non-scalable streams under guaranteed rate constraints has been previously addressed in [44, 43, 45]. The problem has been nicely formalized in more general terms in [49] and [48]. In [53] the authors discuss optimal streaming of layered video under random bandwidth models, when the buffer is not constrained at the decoder. The error concealment at the decoder is further considered in [58] where the scheduling decisions are based on a Markov Decision Process to cope with unpredictable bandwidth variations. The authors of [60] address a similar scenario, where the number of layers that are transmitted are computed from local decisions based on expected

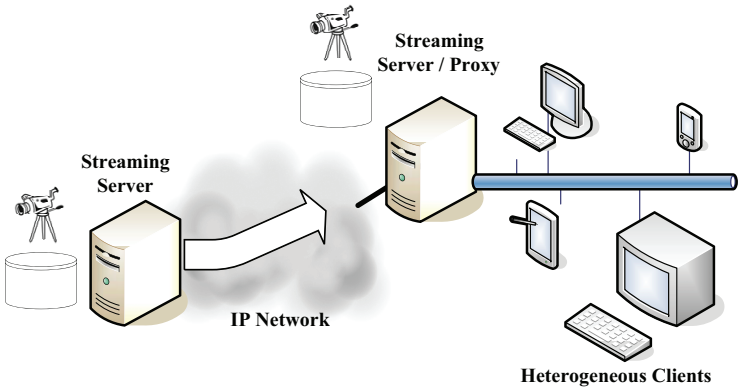

Figure 5.1: Example of a scalable video streaming system.

run-time estimation. None of the above work however considers the problem of multiple clients that participate together in the streaming session.

A scheduling algorithm that minimizes the buffer occupancy of a single client that receives a single stream has been proposed in [42]. Our work extends this algorithm to provide jointly minimal buffer occupancy at heterogeneous clients that decode different subsets of the same layered stream. Optimal multiplexing for continuous media streaming is discussed in [80]. However the authors focus on bandwidth efficiency and do not discuss the delay nor the buffer occupancy experienced by a client. Layered video streaming has been studied in relation with multicast delivery schemes in [51, 52], without addressing the specific problem of heterogeneous receivers with delay and buffer constraints. None of the cited papers addresses the problem of multiplexing a layered video stream onto a broadcast channel by targeting on one hand a set of minimal playback delays for heterogeneous clients, and on the other hand the minimum buffer occupancy at each one of these clients.

In this chapter, we propose to optimize the selection of the playback delays for the different clients in order to have a fair distribution of the delay penalty induced by the broadcast-like media transmission. We show that the minimal playback delays cannot be jointly achieved for all the clients, and we derive low cost optimization algorithms for computing playback delay sets under different client prioritization policies. Once the playback delays are given, we prove that minimum buffer occupancy can be simultaneously attained for all the clients. There is moreover a unique sending trace that attains the optimal solution in this case, and we propose an algorithm that implements the optimal transmission of the packets from the different layers. When both optimization problems are solved sequentially, the system can design a mechanism that jointly optimizes both delays and buffers. To the best of our knowledge, this work is a first effort to address the playback delay optimization problem, together with the buffer minimization problem for broadcast to heterogeneous clients. Finally, we show how the optimal scheduling solution can be modified with a simple adaptive rate control algorithm when the knowledge about the channel bandwidth is limited. This solution provides an interesting alternative to conservative scheduling schemes in some practical scenarios with unpredictable bandwidth,

while it offers an improved average quality but minor and controllable quality variations.

The chapter is organized as follows: we provide an overview of the system under consideration and discuss media scheduling properties in Section 5.2. We present the delay optimization solutions in Section 5.3, and we analyze the buffer minimization problem in Section 5.4. Section 5.5 introduces an adaptive rate control algorithm to cope with unexpected bandwidth variations and discusses its performance in practical scenarios.

5.2 Scalable Video Broadcast

5.2.1 System Overview

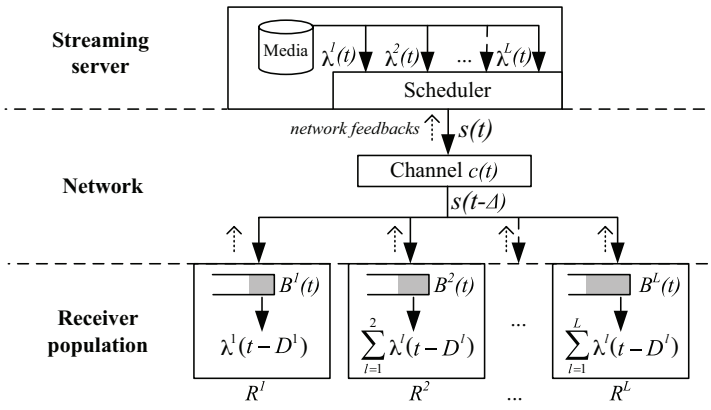

Figure 5.2: Formal view of the system.

We present an overview of the system under consideration in this chapter. A formal representation of the system is given in Figure 5.2. A streaming server sends a scalable media stream to a population of receivers through a common bottleneck channel. This bottleneck represents for example a shared channel or network segment with limited bandwidth, or the disk bandwidth limitations in a video-on-demand server. The bottleneck channel is given by its bitrate $c(t)$, which indicates how many bits the channel is able to transmit at any time t, and possibly by a maximum network latency Δ.

Generally, the server's knowledge about the channel availability is extracted from client or network feedback. In this chapter we will assume perfect channel knowledge at the server, which leads to an upper bound on achievable performance for any predictive scheme, where the server estimates the available channel. In particular, this assumption is verified for constant bit rate channels or when the bandwidth is controlled by deterministic guarantees (e.g., TSPEC in the recent 802.11e wireless protocol). Moreover, as discussed in Chapter 4, while delivering a layered video stream, a sender has often to settle for a sending rate which equals to the average rate of a subset of the available layers. In the rest of this chapter, the channel rate $c(t)$ is the rate available for the broadcast

application, and we do not limit the study to any particular congestion control or rate allocation strategy.

The scalable video stream is built on several hierarchical layers. Each of the L layers is completely determined by its source or playout trace $\lambda^l(t), 1 \leq l \leq L$, which indicates the size of the layer l at time t. When a hierarchy exists between video layers, the decoding of layer l is made contingent on the correct decoding of all inferior layers, from 1 up to $l-1$. Batches of clients simultaneously access the same video sequence, possibly with different scalability levels. The receivers are grouped together into L sets, based on the number of video layers or the resolution that they have requested. We denote as R^l, $(1 \leq l \leq L)$ the set of clients that receive all layers up to the l^{th} layer.

After the first bit is sent by the server, each receiver in R^l buffers the video data for a *playback delay* D^l. The video bits are stored in the receiving buffer, whose content at any time is further denoted as $B^l(t)$. After the initial playback delay, the receiver decodes and plays continuously a video whose resolution corresponds to the series of additive layers it has requested. The playback delay can be different for each group of receivers R^l. However, the set of playback delays $\mathcal{D} = \{D^l\}_{l=1}^{L}$ should be chosen such that non-disruptive playback of the sequence can be achieved for any set of receivers R^l, i.e., such that no buffer underflow occurs at any receiver [1]. At the same time, the choice of the playback delay impacts the quality-of-service perceived by the end-user. It therefore requires an efficient packet scheduling strategy in order to reach a proper trade-off between user experience and resilience to underflows and bandwidth limitations. Before addressing the problems of the selection of the playback delay and packet scheduling, we describe below the problem of packet scheduling in media streaming applications that generally impose strict timing constraints.

5.2.2 Media Scheduling Formalism

We now describe in more details the packet scheduling characteristics in media streaming applications. When the channel is constrained, the streaming server has generally to implement effective scheduling algorithms, in order to ensure timely delivery of media packets and to avoid buffer underflow at receivers. The packet transmission strategy is chosen in order to meet criteria such as desired distortion or delay [61, 59], or maximum utilization of the available channel bitrate. The packet scheduler outputs a stream at a *sending rate* $s(t) \leq c(t), \forall t$ that indicates the number of bits fed into the channel at any time instant t.

We denote the cumulative source (video), sending and channel rate functions with capital letters (V, S, C), where a cumulative rate function is defined as the total number of bits that have been counted since time $t = 0$. For example, $C(t) = \int_0^t c(u)du$ is the number of bits the channel can transmit up to time t. Note that the cumulative rate functions are all wide-sense increasing in t. We further define $v_D(t)$ as the number of video bits consumed by the decoder that starts playing the video stream after a playback delay D. It reads as:

$$v_D(t) = \begin{cases} 0 & , 0 \leq t < D \\ v(t-D) & , t \geq D \end{cases}$$

[1] In the remaining of the chapter, we use R^l to design a set of receivers that subscribe to the resolution level l or one of the receivers in this set, interchangeably.

$V_D(t) = V(t - D)$ is the corresponding cumulative function.

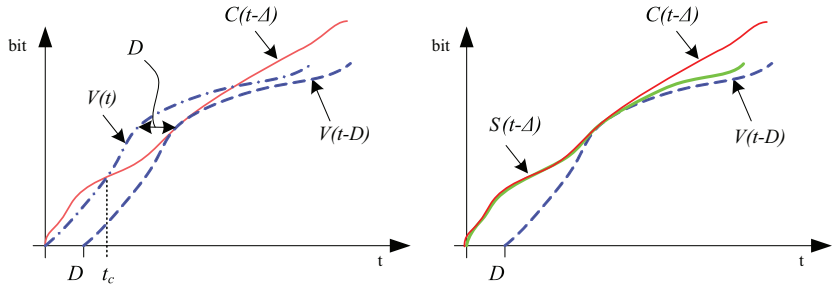

Figure 5.3: *Left:* Playback delay and buffer underflow prevention. *Right:* Schedulable play-out trace and a corresponding sending rate trace.

The scheduler has to ensure that the client does not experience any buffer starvation, when the video decoding starts after a playback delay that has been selected a priori. Figure 5.3 illustrates the importance of the playback delay for smooth video decoding in a scenario with a single client. If the client starts playback at the reception of the first bit, a buffer underflow occurs at time t_c. Starting playback at the client after D time units makes sure that the buffer underflow does not occur. We say that a source trace $v(t)$ is *schedulable* over a channel with available bandwidth $c(t)$, with a playback delay D, if the following *schedulability condition* holds for all t:

$$V_D(t) \quad \leq \quad C(t - \Delta) \tag{5.1}$$

In the case where the channel trace $C(t)$ is completely known and its usage is not further constrained, if the condition (5.1) is met, this implies that the server can find a scheduling solution or equivalently a sending trace $s(t)$ such that each of the following conditions are satisfied for all t:

$$V_D(t) \quad \leq \quad S(t - \Delta) \tag{5.2}$$

$$s(t) \quad \leq \quad c(t). \tag{5.3}$$

The first inequality makes sure that the server transmits a number of bits that is sufficient for decoding the video stream at all time instants t, with a playback delay D and a maximum network latency Δ. As a constant value of Δ implies a simple time shift in each of the above inequalities, we will consider that $\Delta = 0$ for the sake of clarity and without loss of generality in the remainder of this chapter. The second condition simply imposes that the number of bits transmitted by the server at any time instant is not larger than the channel rate. Note that the latter condition implies $S(t) \leq C(t), \forall t$, but that the reverse is not true. If the playback delay D is chosen such that the above conditions hold, a scheduling solution can be found. Each valid scheduling strategy generates a sending rate $s(t)$ that satisfies Eqs. (5.2) and (5.3) for all t. Finally, the buffer occupancy of a media client that receives a data rate $S(t)$ and plays out a sending trace $V_D(t)$ is given by:

$$B(t) \quad = \quad S(t) - V_D(t), \forall t. \tag{5.4}$$

The above equation shows that the buffer occupancy is dependent on the choice of the playback delay, so that the joint minimization of both components becomes non-trivial. We first present a method to optimize the choice of the playback delays in scalable streaming systems. Then we show how to define a scheduling strategy that minimizes also the buffer occupancy.

5.3 Playback Delay optimization

5.3.1 Preliminaries

In this section, we discuss the choice of the playback delays for the different sets of clients R^l that simultaneously receive the scalable stream. Small playback delays usually lead to a better quality of service, and we will present algorithms for optimizing the choice of playback delays, under different metrics. We first give a preliminary analysis of the playback delay, and define the minimal playback delay for a client R^l that decodes l layers of the scalable video stream.

We introduce here some general results inspired from [47]. Suppose that we have two increasing non-zero functions $F(t)$ and $G(t)$ such that $\lim_{t\to\infty} F(t) \geq \lim_{t\to\infty} G(t)$. We define the (maximum) horizontal distance between $F(t)$ and $G(t)$ as follows:

$$h(G,F) \quad = \quad \sup_t \left(F^{-1}\left(G(t)\right) - t\right), \tag{5.5}$$

where $F^{-1}(t) = \min\{t : F(t) \geq x\}$ is a pseudo-inverse of $F(t)$. The following relations hold:

$$h(G,F) = 0 \quad \Leftrightarrow \quad F(t) \geq G(t), \forall t \text{ and} \tag{5.6}$$

$$\exists \tau \text{ s.t. } F(\tau) = G(\tau) \tag{5.7}$$

$$h(G,F) < 0 \quad \Leftrightarrow \quad F(t) > G(t), \forall t \tag{5.8}$$

$$h(G,F) > 0 \quad \Leftrightarrow \quad \exists \tau \text{ s.t. } F(\tau) < G(\tau). \tag{5.9}$$

When $F(t)$ and $G(t)$ respectively represent the channel trace and the source trace, the horizontal distance between F and G represents the minimal playback delay that is necessary for a smooth decoding of the source stream. In other words, it represents the minimal shift that as to be applied on G, such that the schedulability condition is verified. Formally, we have the following property.

Property 1. *If $h(G,F) > 0$ and $G'(t) = G(t - h(G,F))$, then $h(G',F) = 0$. In other words, $h(G,F)$ is the minimum shift we need to apply on $G(t)$, so that $F(t) \geq G'(t)$, $\forall t$.*

With multiple traces, we have also:

Property 2. *Let $F(t)$, $G(t)$ and $G'(t)$ be non-decreasing functions such that $G'(t) > G(t)$, $\forall t$. Then: $h(G',F) > h(G,F)$. Indeed by the definition of $h(\cdot)$ and $F^{-1}(\cdot)$, and because $F(t)$ is non-decreasing, the result follows immediately, as $F^{-1}\left(G'(t)\right) > F^{-1}\left(G(t)\right)$, $\forall t$. Similarly, if $G'(t) < G(t)$, $\forall t$ then $h(G',F) < h(G,F)$.*

We can therefore define the minimal playback delay D^l_{min} for smooth playback at the receiver R^l. It is given by

$$D^l_{min} = h\left(V^l(t), C(t)\right), \tag{5.10}$$

where $V^l(t) = \sum_{k=1}^{l} \Lambda^k(t)$ is the cumulative rate of the stream at resolution l. From Property 2, we know that $D^l_{min} \leq D^{l+1}_{min}$, $\forall 1 \leq l \leq L-1$, since the rate traces are positive valued functions, and $V^{l+1}(t) \geq V^l(t), \forall t$. If the layer l is decoded after a minimal playback delay D^l_{min}, the only valid scheduling solutions are strategies where the playback delay for layers $k < l$ is not any larger than D^l_{min}. It is actually not possible to reduce the minimal playback delay for the client R^l, even by changing the scheduling of the lower layer streams.

Let us define $\vec{\delta} = [\delta_1, .., \delta_l]$, with $\delta_1 \geq \delta_2 \geq \ldots \geq \delta_l \geq 0$. We have the following Lemma, which shows that the minimal playback delay required for a smooth decoding of the resolution level l cannot be smaller than D^l_{min}, even if the lower layers are decoded with a smaller delay.

Lemma 2. *Consider a set of L non-decreasing functions $\{H^l(t)\}_{l=1}^{L}$ and a non-decreasing function $F(t)$, defined $\forall t$. We have, $\forall l$, $1 \geq l \geq L$:*

$$h\left(G^l, F\right) \leq h\left(G^l_{\vec{\delta}}, F\right),$$

where $G^l(t) = \sum_{k=1}^{l} H^k(t)$ and $G^l_{\vec{\delta}}(t) = \sum_{k=1}^{l} H^k(t+\delta_k)$.

Proof. As the functions $\{G^l(t)\}_{l=1}^{L}$ are non-decreasing, we have, $\forall l$ and $\forall \delta_l \geq 0$: $H^l(t) \leq H^l(t+\delta_l)$. Thus, $\forall l$,

$$G^l(t) \leq G^l_{\vec{\delta}}(t) \ ,\forall t.$$

From Property 2, it follows that $h\left(G^l, F\right) \leq h\left(G^l_{\vec{\delta}}, F\right)$. □

From the above results we conclude that any playback delay smaller than D^l_{min} results in a buffer underflow at the receiver R^l, while any larger playback delay allows for decoding without experiencing a buffer underflow. This permits to derive a simple bisection search algorithm for computing the minimal delay D^l_{min}, similar to Algorithm 1.

Algorithm 1 $D_{min} = getDmin\left(C(t), V(t)\right)$

```
1: D_low ⇐ 0
2: D_high ⇐some large value
3: while (D_high − D_low) > 1 do
4:    D_test ⇐ ⌊(D_low + D_high)/2⌋
5:    if V(t − D_test) ≤ C(t), ∀t then
6:       D_high ⇐ D_test
7:    else
8:       D_low ⇐ D_test
9:    end if
10: end while
11: D_min = D_test
```

It is important to note here that achieving the minimum playback delay for a given layer l does not necessarily guarantee that a minimum playback delay

is also achieved for any other layer. In general, if the transmission strategy is chosen in order to minimize the delay at layer l, without considering any other layer k, with $k < l$, the clients that only subscribe to the lower layers are penalized by a playback delay that might be larger than necessary. If however the scheduler decides to minimize the playback delay for layer k, it generally increases the playback delay for clients that receive any additional layer l, with $l > k$. The choice of the playback delay results therefore from a typical trade-off between the delays imposed to the different layers, since the delays cannot be minimized simultaneously for all the layers. When the playback penalty is increased for the lower layers, it typically saves channel bits that can be used for decreasing the playback delay penalty for higher layers. In the next section, we formulate an optimization problem for the choice of the playback delays for different policies.

5.3.2 Problem Formulation

Consider a channel given by its cumulative rate trace $C(t)$, and a set $\mathcal{L}$ containing L hierarchically coded layers given by their cumulative source rate traces $\{V^l\}_{l=1}^{L}$. The channel connects a streaming server to L sets of receivers $\{R^l\}_{l=1}^{L}$, that simultaneously subscribe to layers up to l. Let $\mathcal{D} = \{D^l\}_{l=1}^{L}$, with $D^1 \leq D^2 \leq \ldots \leq D^L \leq D_{max}$ denote the set of playback delays imposed to the different sets of clients. The joint minimization of the playback delays for all heterogeneous receivers is generally not achievable in broadcast-like scenarios, as discussed above. We therefore formulate the following optimization problem, which targets a fair selection of the playback delays. Let D^l_{min} represents the minimal playback delay that can be offered to the client R^l, when other clients are not considered. A fair distribution of the playback delay among the different clients can be achieved by controlling the penalties $\overrightarrow{\Delta} = [\Delta^1, ..\Delta^L]$, with $\Delta^l = D^l - D^l_{min}$, $1 \leq l \leq L$, in addition to minimizing the playback delays. The playback delays can therefore by chosen as

$$\mathcal{D}_{opt} = \arg\min_{\mathcal{D}} \varphi\left(\{D^l\}, \{\Delta^l\}\right) \tag{5.11}$$

under the condition that $V^l_{\mathcal{D}}(t) \leq C(t)\ \forall l, 1 \leq l \leq L$, i.e., all the traces are schedulable from Eq. (5.1). The function φ is a generic cost function that combines the average playback delay and the delay penalty imposed to each layer due to the broadcast-like distribution. Finding the best set of playback delays $\mathcal{D}_{opt}$ is actually a combinatorial optimization problem, and its solution generally implies a full search algorithm. We show in the next sections how the search space can be reduced for solving the generic optimization problem of Eq. (5.11). We also present efficient solutions to the problems of fair or weighted distribution of the delay penalty Δ^l between the different layers.

5.3.3 Reduced search space

In order to solve the joint delay optimization problem, we propose to limit the search space of possible delay values. We know already from the above discussion that the minimal playback delay for clients that decode the stream up to layer l, is D^l_{min}. It corresponds to the lowest achievable delay, when clients R^k with $k \neq l$ are not considered in the delay computation. Generally, the playback

delay for layer l is larger than D^l_{min} when the scheduler also tries to reduce the delay for the lower layers $1 \leq k \leq l$. In order to reduce the search space, we are looking for a reasonable upper-limit on the search interval. From Lemma 2, the worst case policy for clients R^l consists in minimizing iteratively the delays for all the clients R^k with $1 \leq k \leq l$. We describe below the greedy delay allocation policy, and we denote the resulting delay D^l_{greedy}.

The greedy delay allocation first minimizes the playback delay of the first layer, which is thus decoded after $D^1_{greedy} = D^1_{min}$. It then iteratively allocates the smallest possible delay to the different layers, given the greedy delay allocation for the lower layers. Formally, we denote the available channel bandwidth for transmitting the layer l as $C^l(t) = C(t) - \sum_{k=1}^{l-1} \Lambda^k(t - D^k_{greedy})$. Therefore, the minimal playback delay for layer l becomes $h(C^l, \Lambda^l)$ under the greedy allocation policy. This scenario results in an upper bound D^L_{greedy} on the playback delay for the highest layer L, when all playback delays are chosen in a greedy manner. In particular, delays that are larger than D^L_{greedy} also provide valid scheduling solutions. However, increasing the playback delay D^L does not reduce the playback delay of the lower layers, and rather contributes to increasing the standard deviation of the penalties given in Eq. (5.11). The delays obtained by the greedy allocation can therefore be safely considered as the upper-limits of the search intervals. The greedy layered scheduling strategy is shown in Algorithm 2.

Algorithm 2 $\left(\left\{D^l_{greedy}\right\}\right) = GreedyD\left(C(t), \{\Lambda^l(t)\}\right)$

1: $C^1(t) \Leftarrow C(t)$
2: **for** $l = 1$ to L **do**
3: $\quad D^l_{greedy} \Leftarrow getDmin\left(C^l(t), \Lambda^l(t)\right)$
4: $\quad C^{l+1}(t) \Leftarrow C^l(t) - \Lambda^l(t - D^l_{greedy})$
5: **end for**

As the greedy delay allocation provides the worst case solution for minimizing the delay for all the receivers, we can limit the search domain for computing the best set of playback delays to the interval $[D^l_{min}, D^l_{greedy}]$, $\forall l$. In addition, due to the hierarchical nature of the scalable video stream, we know the delay can only take non-decreasing values when the number of layers increases (i.e., $D^k \leq D^l$ when $k \leq l$). We can therefore limit the number of potential solutions that need to be tested for optimality by the search algorithm, by setting the condition that $D^L \in [D^L_{min}, D^L_{greedy}]$. Then, for each possible value of D^L, we constrain the search algorithm to test values of D^{L-1} such that $D^{L-1} \in [D^{L-1}_{min}, D^L]$. The search proceeds iteratively and only test values of delay D^l, such that $D^l \in [D^l_{min}, D^{l+1}]$, for $l = L..1$. Using this simple method, the set of playback delays that minimizes Eq. (5.11) can be identified with high probability for most cost functions φ that tends to minimize the average playback delay. The search space of feasible solutions is however drastically reduced compared to a full search algorithm.

5.3.4 Fair penalty distribution

In order to have a fair policy among the different clients R^l, we can distribute the playback delays such that the variance of the penalties $\overrightarrow{\Delta} = [\Delta^1, ..\Delta^L]$ is minimized. The playback delays can thus be chosen such that

$$\mathcal{D}_{opt} = \arg\min_{\mathcal{D}} \left[Var\left(\Delta^1, \ldots, \Delta^l\right)\right], \tag{5.12}$$

under the condition that $V^l_{\mathcal{D}}(t) \leq C(t)\ \forall l, 1 \leq l \leq L$, i.e., all the traces are schedulable from Eq. (5.1).

We propose a low complexity algorithm for computing the optimal playback delay set $\mathcal{D}_{opt}$ in the sense of Eq. (5.12). We can observe that the minimal value of the cost function is reached when all the penalties are equivalent, i.e., $\Delta^k = \Delta^l,\ \forall k, l$. Any set of delays $\mathcal{D} = \left\{D^l | D^l = D^l_{min} + K\right\}, \forall l$ minimizes the cost of Eq. (5.12) by setting the variance between the delay penalties to zero. In other words the source traces of all layers need to be delayed by K units relative to their respective minimal playback delay D^l_{min}. Given the set of minimum playback delays $\mathcal{D}_{min}$, we can construct an aggregate source rate trace $V^L_{\mathcal{D}_{min}}(t)$, defined as:

$$V^L_{\mathcal{D}_{min}}(t) = \sum_{l=1}^{L} \Lambda^l\left(t - D^l_{min}\right). \tag{5.13}$$

If the trace $V^L_{\mathcal{D}_{min}}(t)$ is schedulable, it represents an ideal solution where all layers can be decoded jointly with minimal delay. If it is not the case, the playback delay can be increased in the same manner for all layers, so that the trace becomes schedulable. It corresponds to shifting the aggregate source trace by the smallest delay K, such that $V^L_{\mathcal{D}_{min}}(t-K) \leq C(t), \forall t$. In other words, we can compute K as

$$K = h\left(V^L_{\mathcal{D}_{min}}, C\right), \tag{5.14}$$

and that can be achieved by running the Algorithm 1. Hence the complexity involved in finding the solution is that of the bisection search algorithm used in Algorithm 1, i.e., $O(log(trace\_length))$. The solution is obviously equivalent to the optimal solution of the algorithm in the previous section, when it lies in the reduced search space.

We illustrate the solution with fair distribution of the delay penalties in Figure 5.4. We have encoded a composite video sequence in QCIF format at 30 frames per second, using the MoMuSys MPEG-4 FGS [41] reference codec. The channel is a piecewise CBR channel that provides a mean rate of 128kbps at the beginning, then improves to 256kbps and finally to 384kbps. Using the fair playback delay distribution proposed above, the playout can begin after a playback delay equivalent to 137 frames at receivers of set R^1. The playback delays for layer 2 and 3 are of 199 and 730 frames respectively. The relative playback delay penalty per client set, compared to their respective D^l_{min} value, is equivalent to 135 frames for all clients. Note that the gain in delay for clients in sets R^1 and R^2 is enormous when compared to a strategy that would have the same delay $D^3_{min} = 595$ frames for all clients (dotted line).

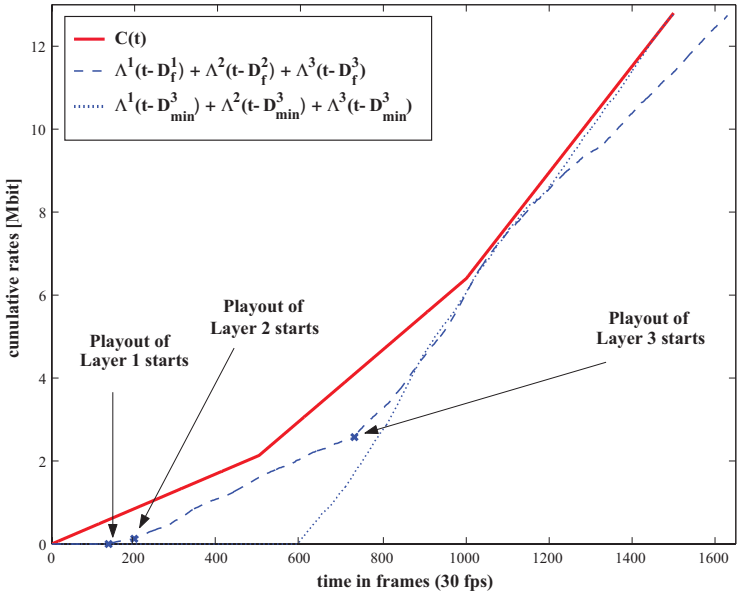

Figure 5.4: The channel can support 3 layers of the encoded stream. The dashed curve shows the aggregate playout curve of the 3 layers with fair values of the playback delays $\mathcal{D}_f$. The aggregate playout curve of the 3 layers using playback delay D^3_{min} is shown for reference (dotted line).

5.3.5 Unequal delay penalties

Some applications may necessitate to devise a scheduling strategy with unequal delay penalties, where some clients are considered as prioritized compared to others. This can be achieved by minimizing the variance of the weighted delay penalties. In this case, the delay distribution has to be chosen according to the following optimization problem:

$$\mathcal{D}_{opt} = \arg\min_{\mathcal{D}} \left[Var\left(w^1 \Delta^1, \ldots, w^L \Delta^L \right) \right], \tag{5.15}$$

under the condition that $V^l_{\mathcal{D}}(t) \leq C(t)\ \forall l, 1 \leq l \leq L$, i.e., all the traces are schedulable from Eq. (5.1). The weights $\overrightarrow{w} = [w^1, ..w^L]$ represent positive weights that permit to control the distribution of the penalties among the L layers. A relatively high value of the weight w^l typically constrains the delay D^l to be close to the minimal delay D^l_{min}.

Depending on the weight distribution, it might be difficult to find the optimal solution to the problem of Eq. (5.15) without using an exhaustive search over the (reduced) space of possible delay values. We however propose a low complexity algorithm that finds the optimal playback delay set $\mathcal{D}_{opt}$. It is based on the a priori information about the structure of the optimal solution that sets the cost function in Eq. (5.15) to 0. It can be reached only when

$$\Delta^l = \frac{w^k}{w^l} \Delta^k, \tag{5.16}$$

$\forall k, l, 1 \leq k \leq L, 1 \leq l \leq L$. This imposes that the delay penalty takes the form $\Delta^l = K/w^l, \forall l$, where K is a constant. Therefore, solving the optimization problem of Eq. (5.15) is equivalent to finding the smallest value of K such that the aggregate trace $V^L_{\mathcal{D}}(t)$ is schedulable. In other words, one has to find the smallest K that verifies

$$\sum_{l=1}^{L} \Lambda^l \left(t - D^l_{min} - \frac{K}{w^l} \right) \leq C(t), \forall t. \tag{5.17}$$

The search algorithm, given in Algorithm 3, simply increases K gradually, until the aggregate trace is schedulable. At each iteration, it updates the playback delays, constructs the aggregate source trace, and checks the schedulability condition. If the resulting trace is schedulable, the algorithm stops. Otherwise, the value of K is augmented by δ, and the process is repeated until the resulting trace is schedulable.

Note that the algorithm may find a sub-optimal solution to the problem of Eq. (5.15) due to granularity of the delay increments. However, the complexity is drastically reduced compared to a full search algorithm. If all the weights are equal, we obviously get back to the *fair* model of the previous section. However, we have seen that in the *fair* case we only need to test $O(log(trace\_length))$ possibilities, where each test can be performed in polynomial time. Algorithm 3 always performs $O(trace\_length)$ such tests.

Algorithm 3 $\mathcal{D}_h = \left(C(t), \{\Lambda^l(t)\}, w, \delta\right)$

$D^l_h \Leftarrow D^l_{min}$, $1 \leq l \leq L$.
Construct the trace using delays D^l_{min}:
$V^L_{\mathcal{D}_h}(t) \Leftarrow \sum_{l=1}^{L} \Lambda^l \left(t - D^l_{min}\right)$
while $V^L_{\mathcal{D}_h}(t) \not\leq C(t)$, $\forall t$ **do**
    increase the delays according to the assigned weights:
    for $l = 1$ to L **do**
        $D^l_h \Leftarrow D^l_h + \frac{\delta}{w^l}$
    end for
    bound delays such that there are non-decreasing:
    for $l = L - 1$ downto 1 **do**
        $D^l_h \Leftarrow \left(min(D^l_h, D^{l+1}_h\right)$
    end for
    construct new trace:
    $V^L_{\mathcal{D}_h}(t) \Leftarrow \sum_{l=1}^{L} \Lambda^l \left(t - D^l_h\right)$
end while
for $l = 1$ to L **do**
    Playback delays are integers (in frame units)
    $D^l_h \Leftarrow \lceil D^l_h \rceil$
end for

We validate the proposed algorithm on a composite sequence, encoded using the MoMuSys MPEG-4 FGS reference codec [81]. We run 100 tests where the channel is a piecewise CBR channel with rates chosen randomly in {128kbps,

256kbps, 384kbps}, and random lengths for each constant rate segment. We set the weights to $w = \{1, 100, 1\}$, which means that the playback delay for layer 2 in the optimal playback delay set should be kept as close as possible to its minimum playback delay. In our results, the optimal playback delay for layer 2 is indeed always within at most 2 frames of D^2_{min}, thus validating our weighted metric function, as expressed in Eq. (5.15). In all the cases, the cost function is minimal. Note that this might not be always the case for the algorithm with reduced search space proposed in Section 5.3.3, since the bounds of the delay interval might not allow to find the optimal solution in the sense of Eq. (5.15) that does not include an explicit minimization of the average playback delay. Finally, the average number of potential solutions tested by the proposed algorithm was $1.59{\cdot}10^3$ for the aforementioned experiment, compared to $1.82{\cdot}10^7$ for the generic full search algorithm in Section 5.3.3. The considerations on the structure of the optimal solution space thus permits a dramatic reduction of the computation time.

5.4 Minimum receiver buffer

5.4.1 β-optimal sending rate

Once playback delays are given, the server still has the flexibility to choose the packet scheduling policy under the constraints given by the channel. The packet scheduling policy typically influences the dynamic behavior of the receiver buffer. In particular, we are interested in defining the sending rate at the server, which minimizes the buffer occupancy at all times t at the receiver in a given streaming scenario represented by $(C(t), V(t), D)$. At the same time, the sending rate shall ensure that the receiver buffer does not experience any starvation in order to guarantee a smooth video playback. This sending rate is called β-optimal and we denote it as $S_\beta(t)$.

If condition of Equation (5.1) is verified, there exists a family of sending rates $\mathcal{S}$ such that each $S(t) \in \mathcal{S}$ satisfies both Equations (5.2) and (5.3). In these cases, the video can be played back at the receiver after D time units without experiencing any buffer underflow. The β-optimal sending rate is the scheduling solution that minimizes the buffer occupancy at the receiver. It can be written as:

$$S_\beta(t) \quad = \quad \arg \min_{S(t)\in\mathcal{S}} (B(t)) \quad \texttt{with} \; B(t) = S(t) - V_D(t), \forall t, \qquad (5.18)$$

which means that, for any sending rate $S(t) \in \mathcal{S} \setminus S_\beta(t)$, we have:

$$S_\beta(t) \leq S(t), \forall t \qquad (5.19)$$

The problem of finding $S_\beta(t)$ has been addressed before under slightly different assumptions using min-plus algebra. We provide a brief overview of this earlier work here for the sake of completeness. Using the formalism from [49] in the case where the channel availability can be reflected by an arrival curve $\sigma(\cdot)$, one can specify $S_\beta(t)$ as being the smallest sending rate that satisfies the following conditions:

- $S(t)$ is a causal flow, i.e., $S(t) = 0$ for $t \leq 0$.

- the flow $S(t)$ is constrained by an arrival curve $\sigma(\cdot)$ that reflects the channel availability constraints. This means that for all $t \geq 0$ and for all $k \in [0, t]$, $S(t) - S(k) \leq \sigma(t - k)$. There is no further constraint imposed by the network.

When the server can prefetch data from any future frame at any time, and when the playback delay is chosen such that there is no buffer underflow at the receiver, there exists a minimal solution to the above set of constraints. It is given by:

$$S_\beta(t) \quad = \quad (V \oslash \sigma)(t - D) \tag{5.20}$$

Here $\oslash$ denotes the *Min-Plus deconvolution* of two wide-sense increasing functions f and g, defined as:

$$(f \oslash g)(t) \quad = \quad \sup_{u:u\geq 0} \{f(t + u) - g(u)\} \tag{5.21}$$

The interested reader is referred to [48] for more details on the network calculus formalism that is used for proving the existence of the minimal sending trace.

In the rest of this Chapter, we will however not consider any service-curve type constraints on the available channel bandwidth. We rather suppose that the complete channel trace $c(t)$ is known for all times t. We propose an algorithm that offers an intuitive and tractable solution for computing the β-optimal sending rate for non-scalable streams. This algorithm is a generalization of the algorithm presented in [44]. We then show that a *jointly* β-optimal sending trace exists also in the case of scalable streams, and we propose a method to compute the sending rate that minimizes the buffer occupancy for a set of heterogeneous receivers.

5.4.2 Single layer streams

We provide an intuitive algorithm for computing the β-optimal sending rate for single layer streams. Let us consider first a limiting case where the channel has to be fully used to transmit the complete bitstream, i.e., $C(D+T) = V_D(D+T)$. In this case illustrated in Figure 5.5 (left), the set of schedulable sending traces only contains one solution. Any sending rate for which there exists some t where $S(t) < C(t)$ implies that $S(D+T) < V_D(D+T)$, where T is the duration of the video sequence. This violates the condition of Eq. (5.2). Hence the only valid sending rate function is also the solution that minimizes the buffer occupancy for all times t. It is given by $S_\beta(t) = C(t)$.

In the general case where $C(T + D) > V_D(T + D)$, several sending traces represent valid scheduling solutions that satisfy the condition of Eq. (5.1), as illustrated in Figure 5.5 (right). In order to compute the β-optimal sending rate, we make the following observations. First, $S_\beta(t)$ obviously shall fulfill the conditions of Eqs (5.2) and (5.3) that define the schedulable solutions. In order to minimize the buffer occupancy $B(t)$, $\forall t$, $S_\beta(t)$ also needs to follow $V_D(t)$ as closely as possible. This is equivalent to keeping the sending rate as small as possible, but still to send enough data to avoid buffer starvation under the constraints imposed by the channel bandwidth. Finally, we know from the limiting case presented above that, whenever there exists a time τ such that $V_D(\tau) = C(\tau)$, the β-optimal sending rate needs to be equal to $C(t)$

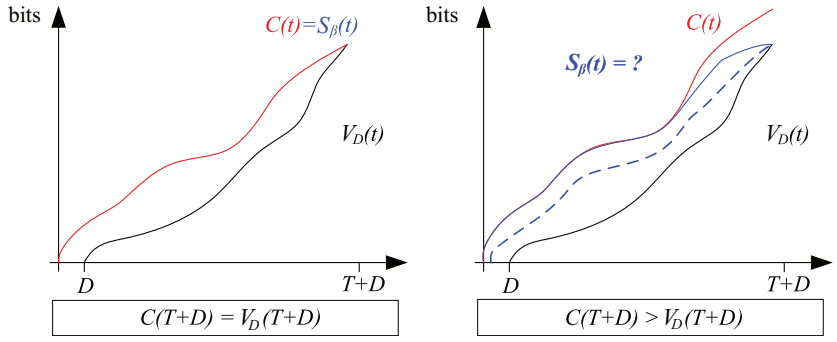

Figure 5.5: *Left:* Limiting case with $S_\beta(t) = C(t)$. *Right:* The set of sending traces $S(t)$ that verifies Eqs. (5.2) and (5.3) generally contains multiple candidates.

up to τ. Therefore, it becomes clear that $B(t)$ can be minimized for all times t if and only if data is sent at the latest possible instant in time such that all data still arrive on time for decoding. We can thus eliminate the early sending opportunities offered by the transmission channel, and *reduce* the channel to the sending opportunities that are *necessary* to transmit all the data before their decoding timestamps.

The β-optimal sending rate can now be computed by the *Variable Rate Smoothing* (VRS) algorithm given in Algorithm 4. The algorithm operates in the cumulative domain, starting at time $t = 0$. It first sets the sending trace to be equal to the channel trace $C(t)$, $\forall t$. Then, it iteratively checks for $t = 0 \dots T + D$ whether there is equality at any future time instant t' between the sending trace and the delayed video trace $V_D(t)$. In this case, the situation is similar to the limiting case presented above, and the sending rate has to be equivalent to the channel rate up to the time instant $t' = t$. However, if the sending trace is strictly larger than the delayed video trace for all time instants $t' > t$, the sending trace at all time instants $t' > t$ is reduced by the difference between the sending trace and the delayed video trace at time t. This operation basically consists in eliminating the early transmission opportunities, which would result in wasting buffer resources. It is equivalent to translating the sending trace curve down by $S_\beta(t) - V_D(t)$ for all $t' > t$. Note that the complexity of Algorithm 4 is $O(T + D)$, where the worst case is achieved if $v(t - D) < c(t)$, $\forall t$.

An illustration of the VRS algorithm is presented in Figure 5.6, where the channel trace $C(t)$ is linear and the delayed source trace $V_D(t)$ is piecewise linear. At time $t = 0$, $S_\beta(t)$ and $V_D(t)$ do not touch. This will remain the same up to t^1. This means that up to t^1, the derivative of $V_D(t)$ is certainly never larger than that of S_β, or equivalently that the instantaneous sending rate is not smaller than the delayed source rate. The algorithm sets the sending trace to $V_D(t)$ up to time t^1. The sending trace computed at time $t = t^1$ touches the source curve at time t^2_{new}. This means that, in the interval $[t^1, t^2_{new}]$, the derivative of $V_D(t)$ is at times larger than the derivative of the sending trace of $S_\beta(t)$. In other words we have reduced the situation in this interval to the limiting case and we have to use all the channel bits in order to transmit the

Algorithm 4 $S_\beta = VRS\left(C(t), V_D(t)\right))$

Require: $V_D(t) \leq C(t), \forall t$

$S_\beta(t) \Leftarrow C(t)$, for all t. The sending trace is computed as a reduced channel trace.
$t \Leftarrow 0$
while $t \leq T + D$ **do**
  if $\nexists t' \geq t$ s.t. $S_\beta(t') = V_D(t')$ **then**
    Reduce the channel down by $S_\beta(t) - V_D(t))$ bits.
    for all τ in $[t, T + D]$ **do**
      $S_\beta(\tau) \Leftarrow S_\beta(\tau) - S_\beta(t) + V_D(t)$
    end for
    $t \Leftarrow t + 1$
  else
    The curves touch, no reduction at this step.
    $t_{new} \Leftarrow \sup_{\tau > t} \{\tau | S_\beta(\tau) = V_D(\tau)\}$
    $t \Leftarrow t_{new}$
  end if
end while

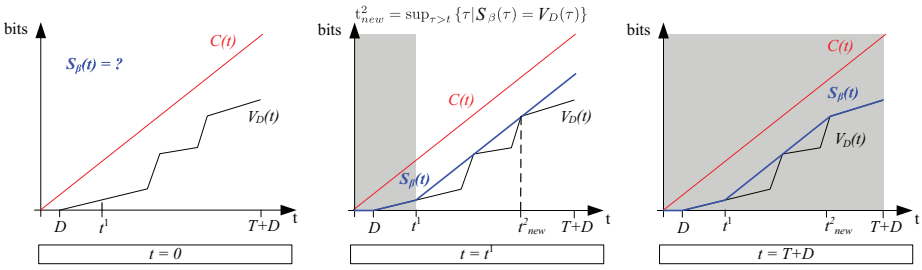

Figure 5.6: Illustration of the VRS Algorithm. In order to minimize the buffer occupancy, the sending trace is reduced to the delayed source trace when the channel rate is superior to the source rate.

needed data. The algorithm does not reduce the sending trace for t in $[t^1, t^2_{new}]$. After $t = t^2_{new}$, the sending trace can again be reduced to the delayed source trace.

Once the optimal sending trace has been computed, the buffer optimal scheduling strategy simply consists in sending data in the increasing order of their decoding deadline while respecting the constraints given by the sending trace. If one does not respect this order, some packets are sent in advance, which can only contribute to the increase of the buffer occupancy. Equivalently, the optimal scheduling of the packets can also be achieved by scheduling packets as late as possible [42], without pre-computing the buffer optimal sending trace. This last opportunity scheduling policy basically consists in reversing time, starting from $t = T + D$ to $t = 0$. Then it schedules at each time instant t as many of the packets with the largest decoding deadlines in $v_D(t)$ as the channel $c(t)$ permits it. This solution jointly computes the buffer optimal scheduling, and the optimal sending trace. It is however based on reversing the

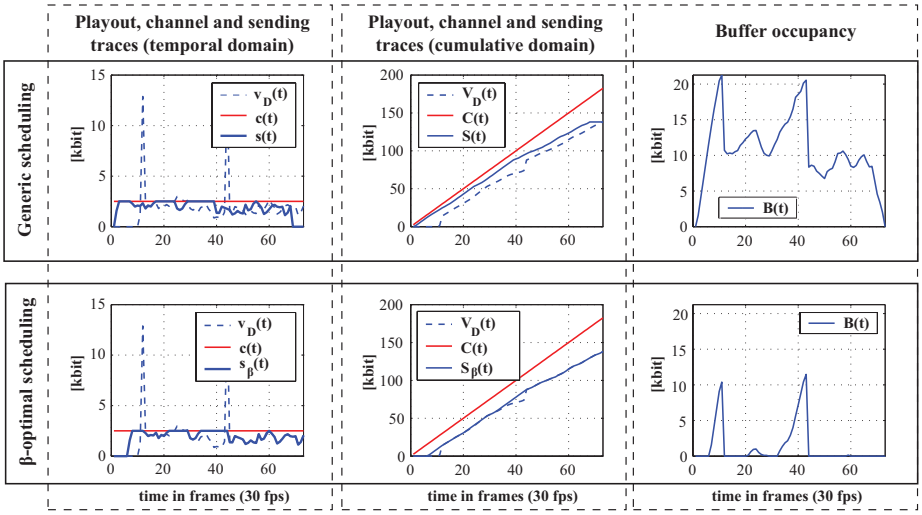

Figure 5.7: β-optimal scheduling outperforms any generic scheduling algorithm with respect to receiver buffer occupancy.

time axis after $t = T + D$, which might present limitations in some practical systems.

Finally, Figure 5.7 illustrates the β-optimal sending rate: the used video trace is formed of 2 GOPs of the MPEG-4 encoded Foreman sequence. The channel is a constant bit rate (CBR) channel. The left column depicts the video source rate $v(t)$, channel rate $c(t)$ and the illustrated sending rate. The middle column shows the same traces in the cumulative domain, and the right column shows the buffer occupancy as a function of time: $B(t) = S(t) - V_D(t)$. The top row shows one valid but sub-optimal sending trace. Note that $\sup_t (S(t) - V_D(t)) = 21264 > 11468$ bits (see *top-right*). The bottom row shows the β-optimal scheduling policy, where the sending rate follows the source rate whenever possible. Whenever $v_D(t) > c(t)$, data is sent at the latest possible opportunity, thus minimizing the buffer occupancy for all t. The maximum amount of buffering needed is 11468 bits (see *bottom-right*).

5.4.3 Scalable streams

In this section, we consider the case of scalable streams and we show that there exists a scheduling strategy that jointly minimizes the buffer occupancy $B^l(t)$ for each receiver group R^l and at all times t. Then we propose a scheduling algorithm that offers a practical solution to build the sending trace $S_\beta(t)$ that is jointly β-optimal for multiple receivers.

We consider that a set of source traces $\mathcal{L} = \left\{\Lambda^l(t)\right\}_{l=1}^{L}$, representing L additive hierarchically encoded layers are sent simultaneously to multiple receivers $R^l, 1 \leq l \leq L$ through a joint bottleneck channel given by the cumulative rate $C(t)$. We further consider a set of non-decreasing playback delays $\mathcal{D} = \left\{D^l\right\}_{l=1}^{L}$ that are used by the different receiver groups. Each receiver in the group R^l starts consuming the media layers 1 to l of the hierarchically encoded stream

after an initial playback delay D^l. The *aggregate source trace* that has to be sent over the channel in order to ensure a smooth playback by all decoders is constructed as:

$$V_{\mathcal{D}}^l(t) \quad = \quad \sum_{i=1}^{l} \Lambda_{D^i}^i(t), \tag{5.22}$$

where $\Lambda_{D^i}^i(t)$ is the cumulative function of $\lambda_{D^i}^i(t)$, the source trace of layer i, delayed by D^i. We assume here that $\mathcal{D}$ is chosen such that the full stream is schedulable, i.e. that $V_{\mathcal{D}}^l(t) \leq C(t)$, $\forall t$, $\forall l$. It is important to note here that the cumulative rate given by Eq. (5.22) is in fact larger than the rate used by the decoder. When a receiver in R^l starts playing the stream at time D^l, all the source traces up to l are drained simultaneously from the receiver buffer. Thus the *playout trace* at a receiver in R^l, which represents the number of bits consumed up to time t, is given as:

$$V_{D^l}^l(t) \quad = \quad \sum_{i=1}^{l} \Lambda_{D^l}^i(t). \tag{5.23}$$

These different traces are illustrated in Figure 5.8. Note that we have:

$$S^l(t) \geq V_{\mathcal{D}}^l(t) \quad \geq \quad V_{D^l}^l(t), \forall t, \tag{5.24}$$

where the first inequality is due to the schedulability condition, and the second inequality results from the construction of the playout trace, with $D^i \leq D^{i+1}$. There are in general several valid sending traces $S^l(t)$ for scheduling the layers 1 to l under a given set of delay and source trace constraints. We denote this set of valid traces as $\mathcal{S}^l = \{S^l(t)\}$. We are interested in finding the trace $S_\beta^l(t) \in \mathcal{S}^l$ that minimizes the buffer occupancy at all the receivers R^l for all times t. It corresponds to the sending trace that minimizes $B^l(t) = S^l(t) - V_{\mathcal{D}}^l(t)$, $\forall t$. The cumulative sending trace is built on l additive layers, and we denote the sending trace of layer l as $Y^l(t)$, with $\sum_{k=1}^{l} Y^k(t) = S^l(t)$. Similarly, we denote the β-optimal sending trace of layer l as $Y_\beta^l(t)$.

From the previous section, we know that if we only consider one resolution level l, $S_\beta^l(t)$ exists. It can be computed by Algorithm 4 for every resolution level l. In a scenario where clients might subscribe only to a subpart of this aggregated stream for a resolution level $k < l$, such a scheduling would however be suboptimal in terms of buffer occupancy for the low resolution clients. In other words, if the sending rate is generated from the stream at resolution l without explicitly considering the lower layers, we can a priori not provide any guarantee on the buffer occupancy at receivers R^k, $k < l$ that consume only the lower layers. We are rather interested in finding the sending trace that minimizes the buffer occupancy at all times t for all receivers simultaneously, if such a solution exists. We prove below an important proposition that says that a joint β-optimal scheduling for multiple receivers actually exists, and that the solution $S_\beta^l(t) \in \mathcal{S}^l$ can actually be constructed on the β-optimal traces for layers $k < l$.

Proposition 1. *If $V_{\mathcal{D}}^L(t) \leq C(t)$, $\forall t$, then there exists a scheduling policy that is jointly β-optimal for all receivers R^l, $1 \leq l \leq L$ that respectively consume the layers 1 to l after an initial playback delay D^l.*

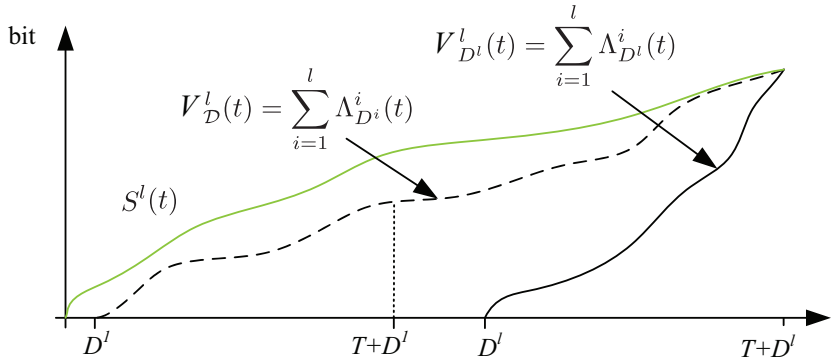

Figure 5.8: Illustration of the aggregate source trace $V^l_{\mathcal{D}}(t)$ and the play-out trace at receiver R^l, denoted as $V^l_{D^l}(t)$, along with one of the possible sending traces $S^l(t)$.

Proof. Given $C(t)$ and $V^l_{\mathcal{D}}(t)$, we know that $S^l_\beta(t)$ exists, for any l taken individually. We want to show that there exists a valid sending trace $Y^l_\beta(t)$ for scheduling layer l, when streams at resolution l and $l-1$ both minimize the buffer occupancy for the respective client sets. Such a trace can be written as:

$$Y^l_\beta(t) = S^l_\beta(t) - S^{l-1}_\beta(t). \tag{5.25}$$

It is a valid sending trace for layer l iff

$$\Lambda^l_{D^l}(t) \leq Y^l_\beta(t) \leq C(t) - S^{l-1}_\beta(t), \forall t. \tag{5.26}$$

In other words, the sending trace for layer l has to be large enough to ensure a smooth playback after a delay D^l. At the same time, it has to be small enough to respect the channel constraints, once the sending trace $S^{l-1}_\beta(t)$ has been allocated already. As $S^l_\beta(t)$ is schedulable by hypothesis, we have $S^l_\beta(t) \leq C(t)$. We can therefore write $S^l_\beta(t) - S^{l-1}_\beta(t) \leq C(t) - S^{l-1}_\beta(t)$ Combined with Eq. (5.25), it leads to proving the second part of Eq. (5.26).

The schedulability of $S^l_\beta(t)$ also induces that the data of layer l are present on time at the decoder. In other words, there exists a set of sending traces $S^{l-1}(t)$ for the data of layers 1 to $l-1$ such that

$$\Lambda^l_{D^l}(t) \leq S^l_\beta(t) - S^{l-1}(t), \forall t.$$

In particular, since by definition $S^{l-1}_\beta(t) \leq S^{l-1}(t)$, we have

$$\Lambda^l_{D^l}(t) \leq S^l_\beta(t) - S^{l-1}_\beta(t), \forall t,$$

or equivalently

$$\Lambda^l_{D^l}(t) \leq Y^l_\beta(t),$$

which proves the first part of Eq. (5.26).

Therefore, there exists a valid sending trace that minimizes jointly the buffer occupancy for receivers sets R^{l-1} and R^l. By recursion, we can construct the β-optimal solutions as $S^l_\beta(t) = \sum_{k=1}^{l} Y^k_\beta(t)$.

Moreover, we prove by contradiction that $Y_\beta^l(t)$ is the minimal valid sending trace for layer l on a channel of bandwidth $C(t) - S_\beta^{l-1}(t)$, when the playback delay is set to D^l.

Assume that there exists a trace $Y_l(t)$ such that $Y_l(t) < Y_\beta^l(t)$ at some time t. In this case, we have $S^l(t) = S_\beta^{l-1}(t) + Y^l(t) \leq S_\beta^l(t)$, which contradicts the assumption on the optimality of $S_\beta^l(t)$. $Y_\beta^l(t)$ is therefore the minimal sending trace for layer l. □

Based on Proposition 1, we can build an iterative algorithm to build the joint β-optimal sending rate for any layer l by greedily building the β-optimal sending rate one layer at a time, starting at the lowest one. In particular, we have

$$S_\beta^l(t) = \sum_{i=1}^{l} Y_\beta^i(t), l > 1 \tag{5.27}$$

and $Y_\beta^1(t) = S_\beta^1(t)$. The sending rate can be computed for each layer iteratively starting from layer 1 by the VRS algorithm. It computes the sending trace that corresponds to $\Lambda_{D^l}^l$, the bits of layer l that are decoded after a playback delay D^l. The bandwidth constraints are updated iteratively, as the bandwidth used by the lower layers is removed from the channel capacity. Therefore, we have

$$C^l(t) = C^{l-1}(t) - Y_\beta^{l-1}(t), \forall 1 < l \leq L, \tag{5.28}$$

where $C^l(t)$ corresponds to the part of the channel that is available to schedule bits from layer l, and $C^1 = C(t)$. Once the optimal sending traces $Y_\beta^i(t)$ have been computed for each layer l, the packet scheduler proceeds by sending the data of each layer in the increasing order of the decoding deadlines, while respecting the different sending traces. For each layer, the scheduler proceeds similarly to the scheduler for single layer streams.

Note that another strategy could be proposed to reach the buffer optimal sending traces. It consists in reverting the time axis, starting from $t = T + D^l$ to $t = 0$. Then the packets of each layer are sent at the latest moment for correct decoding, while respecting the channel bandwidth $c(t)$. As the layer 1 is decoded with the smallest playback delay, it is scheduled first. Other layers are scheduled iteratively under the constraints given by the remaining channel bandwidth $c^l(t)$. Similarly to the case of single layer streams, this solution guarantees the lowest buffer occupancy at the decoder, without the explicit computation of the optimal sending traces. From Proposition 1, it also leads to the jointly optimal policy for all the resolution levels, or all the clients R^l.

We illustrate the performance of this iterative algorithm in Figure 5.9: Figure 5.9-*left* shows a constant bitrate channel and the playout rates of 2 GOPs of the MPEG-4 FGS encoded Formeman sequence, at receiver R^1 (layer 1 only) and R^2 (layers 1 and 2). Playout begins at all the receivers after D=20 frames. In Figure 5.9-*middle*, the same scenario is shown in the cumulative domain. Only the aggregate playout trace at R^2 (i.e. $\Lambda^1(t-D) + \Lambda^2(t-D)$) is shown in blue. The green curve shows the β-optimal sending rate for the aggregate playout curve and the considered channel, as given by the VRS Algorithm. It is thus the β-optimal sending rate for receivers in the set R^2. Figure 5.9-*right*: on the one hand, the solid and dashed blue curves show the sending rates for layer 1

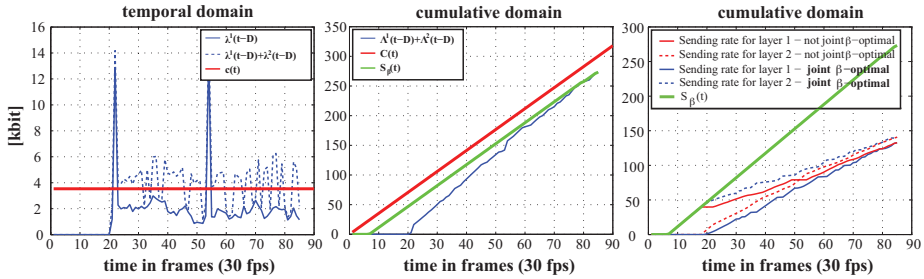

Figure 5.9: Validation of the β-optimal scheduling scheduling algorithm. *Left:* the traces of two layers and a CBR channel in the temporal domain, playout starts after 20 frames. *Middle:* The channel trace and the aggregate playout trace for both layers in the cumulative domain. The green curve shows the sending trace that minimizes the buffer occupancy at R^2, as given by the VRS algorithm. *Right:* We achieve the β-optimal sending trace for layer 1, without sacrificing the β-optimality of the aggregate sending trace for both layers 1 and 2.

data and layer 2 data respectively, in the case where the β-optimal sending trace is computed from the aggregate playout trace at R^2. On the other hand, the dashed red curve shows the β-optimal sending rate for R^1, $S^1_\beta(t)$, obtained from a β-optimal scheduling of layer 1 over the channel $C(t)$. It can be noticed that data for layer 1 is only transmitted shortly before the playout deadline, thus reducing the buffer occupancy at R^1. In this scenario, the solid red curve shows the sending rate of layer 2 over the remaining bitrate $C(t) - S^1_\beta(t)$. Note that in both cases, all data that is sent meets their deadline, and in both cases, the two respective sending rates add up to $S_\beta(t)$ (green curve), which is the β-optimal sending rate for R^2.

In the same scenario, Figure 5.10 finally shows the evolution of the buffer occupancy at receivers R^1 (left) and R^2 (right) if joint β-optimal scheduling is used (top) and if β-optimal scheduling is computed only on the 2 layers stream (bottom). The minimum buffer occupancy at R^2 is achieved in both cases, however the minimum buffer occupancy at R^1 is only achieved in the first case (top-left). The joint β-optimality is achieved through the fact that receivers that subscribe to higher layers buffer less data from lower layers in the first case, and more data from higher layers. However, the buffer contains the same total amount of data in both scheduling choices.

Finally, it is important to note that joint playback delay and buffer optimization can be achieved with the algorithms proposed in the Sections 5.3 and 5.4. The delay optimization does not put assumptions on the actual sending traces, it only considers schedulability conditions. Similarly, when playback delays are selected, the buffer optimization simply consists in finding the smallest sending trace among the set of valid traces. Both problems can be solved sequentially, and the resulting solution jointly optimizes the playback delay, and the buffer occupancy.

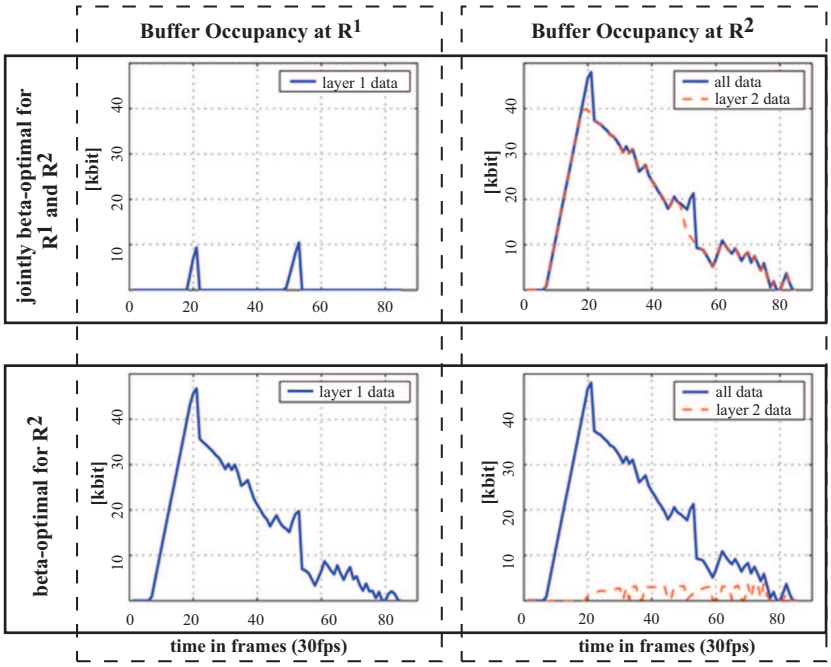

Figure 5.10: Buffer evolution in β-optimal scheduling scenarios.

5.5 Channel-adaptive streaming

5.5.1 Source rate adaptation

In the previous sections, we have provided an analysis of the playback delay and buffer occupancy, as well as joint optimization strategies. These solutions rely on the assumption of perfect knowledge about the bottleneck channel bandwidth. They provide upper-bounds on the performance of common practical systems, where the complete channel trace is usually not known at the server. When the actual channel bandwidth does not exactly correspond to the trace that is used for packet scheduling, the server may not be able to send all packets according to their computed schedules. It has therefore to take actions such as reduction in the source rate, to adapt to temporary bandwidth reduction. Rate adaptation can be performed efficiently on scalable streams by dropping packets from the higher layers. If such mechanisms are used carefully, the quality of service is not significantly affected. Another solution is to devise a conservative scheduling approach that considers lower-bounds on the channel bandwidth. The authors in [46] for example compute the playback delay for a single stream over a *stochastic* channel by deriving a channel trace that lower bounds all possible realizations of the channel. Rate adaptation generally reaches a higher average quality than conservative scheduling methods, at the price of possibly higher quality variations for the clients that subscribe to the highest resolution streams.

We assume that the server knows some channel statistics such as the average bottleneck bandwidth $\bar{c}$. The playback delays and the sending traces are initially

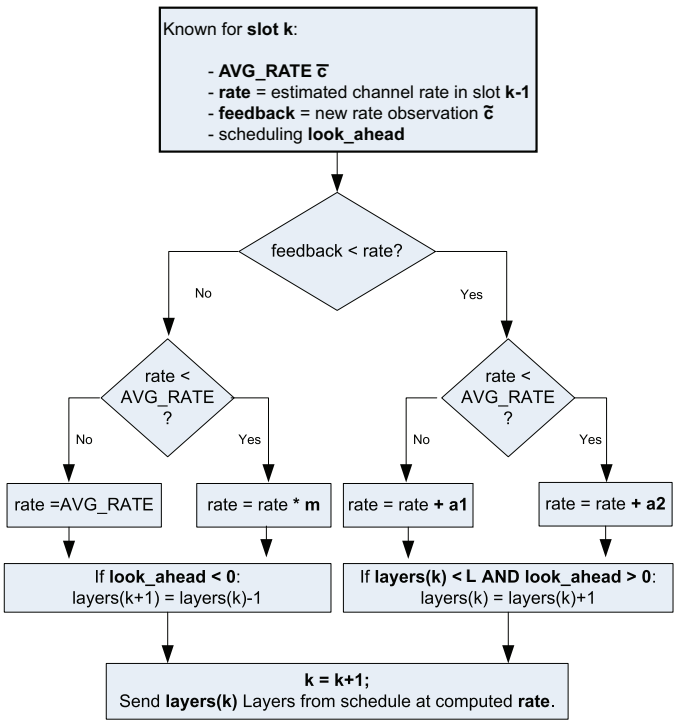

Figure 5.11: Rate adaptation algorithm

computed based on a constant bit rate channel of rate $\bar{c}$. This results in a complete schedule that determines which packet (and which layer) has to be sent at each time instant t, in an ideal scenario. During the broadcasting session, the server monitors the state of the channel, and the sending rate can be adapted in case the available bandwidth becomes insufficient to be able to respect the original packet schedule.

We propose below a sample system based on a simple rate adaptation algorithm, and we show that rate adaptation still permits to keep the buffer occupancy close to minimum, and that playback delays close to the ideal values can be achieved, at the price of only minor and controlled PSNR degradations.

5.5.2 System description

We have tested the rate adaptation scheme on a sample system. The scalable video stream is segmented such that data from different frames and different video layers are fed into different RTP packets. The video stream is sent simultaneously to 3 clients R^1, R^2 and R^3 that decode layers up to 1, 2 and 3 respectively. The average channel rate has been set to 32 kbytes/sec, and we use a NISTNet [82] network emulator to limit the bandwidth on the server-client broadcast link according to a given random bandwidth trace, which is unknown at the server.

The server sends the stored layered stream according to the scheduling strategy computed with a CBR channel of $\bar{c} = 32$ kBps and a given set of target playback delays $\mathcal{D}$. At each discrete time t the server transmits RTP packets according to the ideal scheduling plan, when it is possible. At the same time, the server updates the channel rate estimate as well as the scheduling look-ahead after each second. The approximate channel rate is computed from the round-trip time estimated that are sent in the client RTCP receiver reports, as $\tilde{c} = \frac{packet\_length*2}{RTT}$, where `packet_length` is the average length of packets that have been sent during the previous slot. The scheduling look-ahead represents the difference between the actual scheduling, and the scheduling that has been pre-computed with an ideal channel. In other words, it measures the advance that the scheduler has taken compared to the pre-computed schedules.

Based on these parameters, the rate adaptation algorithm presented in Figure 5.11 adapts the sending rate according to a AIMD (additive increase multiplicative decrease) policy. This policy has been selected for illustrative reasons and for the ease of its implementation. Importantly, it is in no way proven to be optimal, and more elaborate schemes will certainly provide more robust channel estimations. The additive step size is dependent on whether the current rate is above or below the targeted average rate, and we have empirically chosen the following factors, which have yielded good responsiveness of the algorithm in our simulations:

- $a_1 = 1$, if $\tilde{c} > \bar{c}$
- $a_2 = 2$, if $\tilde{c} \leq \bar{c}$.

The choice of having a lower increment if the estimated rate is above average, leads the server to taking advantage carefully of the available rate, while trying to avoid over-estimation. The order of packets is maintained even if the rate has

to be adapted. The sending rate is thus augmented by advancing faster on the pre-defined schedule, and resulting in a positive scheduling look-ahead value.

When the sending rate has to be reduced, we use a multiplicative decrease policy with a factor $m = 0.96$, which has been selected empirically. However, if a decrease occurs when the rate is above the average rate, the rate is immediately clipped to the average rate. The choice of these parameters is based on empirical data and depends on the channel statistics. If the transmission is ahead of schedule, the look-ahead is used to absorb the temporary rate decay and the sending rate is simply reduced, while the order of the packets is maintained. If however, the transmission is running behind the original schedule, packets of the highest layer are not transmitted and simply dropped.

5.5.3 Experimental results

The performance of the above sample system is now analyzed through experiments. We have encoded a composite video sequence (foreman, container, harbour) in QCIF format at 30 frames per second, using the MoMuSys MPEG-4 FGS reference codec. The size of one Group of Pictures (GOP) is of 32 frames. In order to have multiple SNR scalability layers, we have split the FGS enhancement layer along bitplane boundaries, thus coefficients from the same bitplane go into the same SNR layer. The set of target delays was set to $\mathcal{D} = \{D_1 = D_2 = 98, D_3 = 381\}$. The server computes the schedule assuming a CBR channel of 37.5kbps. This corresponds to the effective average rate of the channel that is used for transmissions, although the latter does not provide a constant rate. Based on the feedback it receives, the server then adjusts the schedule accordingly the rate fluctuations that are observed around the average rate.

Figure 5.12 shows the number of layers that are transmitted by the rate-adaptive server, as well as the evolution of the scheduling look-ahead as compared to the pre-computed schedule. We see that at time instant $t = 23$ sec, the channel estimate is low and the scheduling look-ahead is not sufficient to continue sending all layers. So the server stops transmitting layer 3 in order to avoid further congestion until both the channel rate and the scheduling look-ahead increase again. In the illustrated simulation run, this failure is largely due to the bad estimation of the channel at times 0 to 10, where a larger look-ahead could have been built up. Finally Figure 5.13 shows the decodable received source trace at client R^3 that subscribes to the complete stream, and starts decoding after $D_3 = 391$. It can be seen that approximately 3 seconds worth of layer 3 data are missing. This corresponds to the amount of layer 3 data that was not transmitted due to the server's rate adaptation. Dropping the highest layer temporarily from the broadcast leads to a decrease of less than 0.5dB in average PSNR compared to the complete reception of layer 3. However, if a conservative scheduling approach is chosen in such a scenario, the layer 3 is not transmitted at all. The average quality is therefore higher with the rate adaptation solution, at the price of quality variations.

We analyze in Figure 5.14 the influence of the rate adaptation on the playback delays that are necessary to ensure smooth decoding at the receivers. The target playback delays that are pre-computed in an ideal streaming scenario are $\mathcal{D} = \{D_1 = D_2 = 98, D_3 = 381\}$. These delays are obviously conservative, since they can be achieved only when the channel rate corresponds exactly to

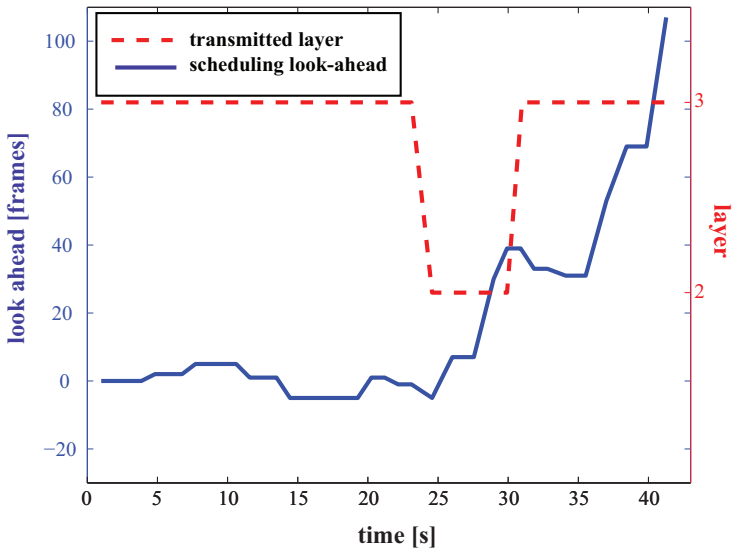

Figure 5.12: Scheduling look-ahead compared to the pre-computed schedule (solid line) and number of transmitted layers (dashed line) according to the rate adaptation algorithm.

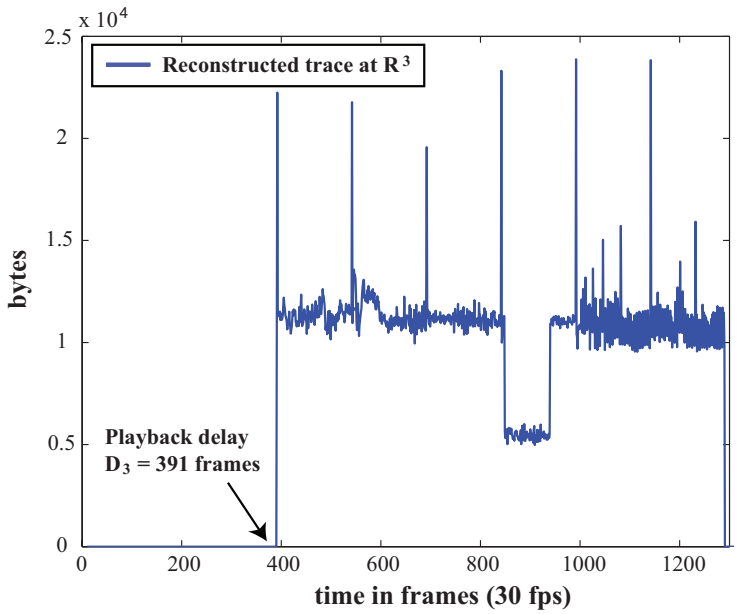

Figure 5.13: Playback trace at R^3, which expects to decode the complete stream (3 layers).

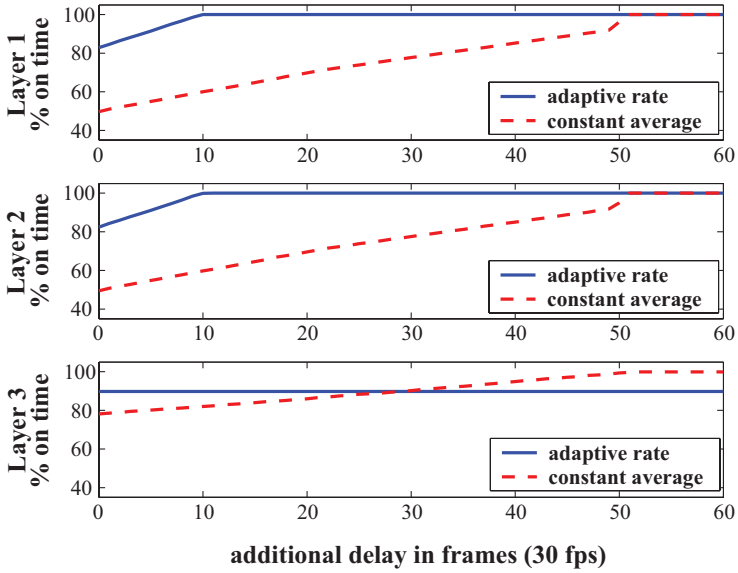

Figure 5.14: Percentage of frames that are received on time as a function of a playback delay margin K.

the previsions. In order to illustrate the influence of the rate adaptation, we represent the number of packets that arrive on time, as a function of an additional delay K used for decoding the streams (i.e., the actual playback delays are $\mathcal{D} + K$). The solid and dashed lines respectively represent the behavior of the rate adaptive scheme, and of an algorithm that does not try to adapt to the actual bandwidth and simply transmits packets according to the pre-computed schedule. It can be seen that the rate adaptive server clearly achieves better performances by keeping the necessary playback delays close to the targeted ones. If an additional delay of only 10 frames is used at the decoder, all layers can be decoded without buffer underflow. Rate adaptation therefore permits to efficiently control the quality of the transmission and to respect the timing constraints of the streaming application. A small conservative margin on the playback delays is sufficient to guarantee a smooth playback.

Finally, we analyze the buffer occupancy at the three receivers. Figure 5.15 illustrates the buffer fullness for playback delays of $\mathcal{D}$ and $\mathcal{D}+10$. In the second case, which ensures a smooth playback delay, we have computed the maximum difference between the actual buffer occupancy and the optimal buffer occupancy in the ideal scenario with a CBR channel rate of 37.5kbps. We can see that the difference with the ideal scenario is always lower than 21kbytes, which is a negligible penalty.

5.5.4 Discussion

The experimental results show that even a simple rate adaption algorithm based on partial channel knowledge can yield results that are close to optimal in terms of both targeted playback delays and buffer occupancy, at the expense of some

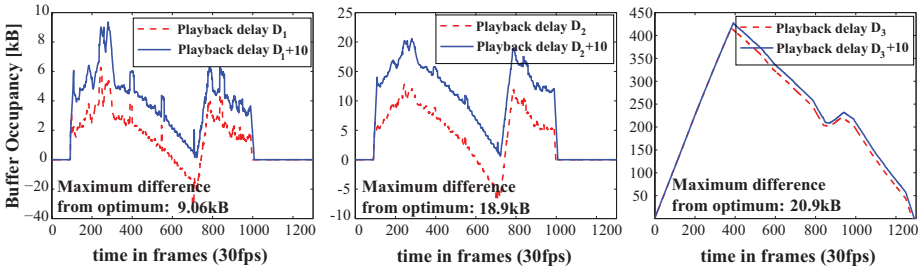

Figure 5.15: Receiver buffer occupancy for R^1 (left), R^2 (middle) and R^3 (right) when using the rate adaptive streaming algorithm.

	D_1	D_2	D_3	$max(B_3)$
non-adaptive	329	329	619	759.55kB
adaptive	108	108	391	430kB
optimal(CBR)	98	98	381	409.5kB

Table 5.1: Performance comparison between an adaptive and non-adaptive scheduling scheme.

minor and controllable PSNR degradations.

In order to highlight this tradeoff we compare the performance of the proposed channel-adaptive scheduling scheme to a baseline system in Table 5.1. In the baseline system, the server computes the same optimal schedule as our system, by assuming a CBR channel of 37.5kbps. The channel that is used in the experiment fluctuates randomly but its average rate is equal to 37.5kbps. The baseline system does not adapt its transmission schedule while streaming: if the effective channel rate is above average, the server can not exploit it. On the other hand, if the channel rate is below average, the transmitted data will be delayed. We show the playback delays at receivers D_1, D_2 and D_3 that need to be respected at clients R_1, R_2 and R_3 respectively, in order to ensure smooth playback. We further indicate the maximum buffer capacity that is needed at client R_3 in both the adaptive and non-adaptive scheduling scheme. As a reference we indicate the optimal playback delays and minimal buffer occupancy that would be achieved if the channel would be CBR with a constant rate of 37.5kbps.

It is worthwhile to be noted that our experimental setup behaves like an overly *nice* network, as any injection of data at a higher rate than the actual channel rate results in a pure delay at the receiver. That is why in the non-adaptive setup, all data is eventually received at each one of the clilents, hence there is no PSNR degradation as compared to the optimal scenario. There are no losses due to buffer overflows (congestions) in the network. If such losses happen, we expect that the rate adaptive system is less affected than the non-adaptive server, since it makes effort to avoid congestions by changing the sending rate according to the available bandwidth.

In the adaptive scheme, part of layer 3 is not transmitted due to a detected temporary bandwidth shortage. This results in an average PSNR degradation of 0.39dB at the receiver R_3. From Table 5.1 it can be seen that this minor

quality degradation allows for an overall Quality of Service that is close to that of the supposed optimal system. An additional delay of only 10 frames (0.33 seconds) allows for smooth playback at each one of the receivers, compared to 238 frames (7.92 seconds) in the non-adaptive scheme. Furthermore, the buffer requirement is nearly twice as large in the non-adaptive scheme, when compared to the proposed channel-adaptive system.

Finally, it can be noted that the additional playback delay that is needed to compensate the discrepancies between estimated rate and actual channel rate can be negotiated between the server and the clients at the beginning of the streaming session. They represent a trade-off between resiliency to channel variations and the waiting time before decoding that is usually kept minimal.

5.6 Conclusions

This chapter has described the problem of scalable media scheduling in broadcast scenarios. In particular, we have shown the playback delay can generally not be jointly minimized for all the receivers. It typically represents the price to pay for applications where different users simultaneously subscribe to different quality levels of the same stream. We have presented a reduced complexity solution for optimizing the delay in a set of receivers. When the optimal strategy consists in minimizing the variance of the delay penalties, we have proposed low complexity algorithms that compute the optimal delay set. When delays are fixed, we have shown that there is a unique scheduling solution that minimizes the buffer occupancy at all the receivers simultaneously. If both problems are solved sequentially, one can achieve jointly an optimal delay selection and a minimal buffer occupancy. Finally, we have proposed a rate adaptation algorithm, which deals with unpredictable channel bandwidth variations. This simple scheme permits to achieve close to optimal results, even when the knowledge about the channel status is limited. It provides a viable alternative to conservative packet scheduling in practical streaming scenarios.

Chapter 6

Distributed Streaming using Rateless Codes

6.1 Background

While we have considered media-friendly rate allocation in one-to-one streaming scenarios in Chapters 3 and 4, in Chapter 5, we have proposed adaptive scheduling algorithms for layered media streams in one-to-many delivery schemes. In the current chapter, we will consider a distributed streaming architecture in which multiple senders are available to stream scalable video content to a single client.

Today's networking infrastructure presents the property of diversity. Whenever the content that a client intends to consume is available at multiple servers in the network, it can be beneficial to use all or some of these available sources for content delivery. Each single channel that connects a particular source to the client may be driven by either a media-unfriendly, or media-friendly rate and congestion control algorithm. But due to the network dynamics, the load on any of these channels is prone to reach levels that are high enough to provoke temporary outages. By exploiting the network diversity, the delivery mechanism can be made more robust. The aggregation of the channels that connect each available source to the client, provides a more robust transmission resource. This aggregate channel eliminates the single point of failure that is represented by a single transmission channel.

Related work in the area of distributed streaming [18] has shown that the usage of multiple streaming servers in different network locations provides better robustness in case one of the channels becomes congested. As the data packets most likely take different paths from their respective source to the client, the overall network load can be balanced, and the most reliable paths can be exploited more efficiently. Similarly, sources in modern peer-to-peer (P2P) systems may not be able or willing to commit to send the full video bitstream to a single client down-stream, especially if the rate of the stream is high. In such a scenario, aggregation from multiple peers is the only way to effectively deliver the requested stream at the desired quality. Note that in the remainder of this chapter we will use both terms servers and peers interchangeably.

Due to the varying nature of each of the available channels, the coordination

of the available senders is critical for the success of such a delivery scheme. In classical server-client architectures, the scheduler needs to decide which video packet of which frame to transmit at a given time, based on both the channel state and the relative importance of each video packet. When multiple senders are considered, each of the schedulers needs to incorporate the decisions of all the other available senders into its own scheduling process. In an effort to use the network resources efficiently, servers need to carefully coordinate their packet scheduling strategies [13], and avoid wasting bandwidth by duplicate packets. This renders the deployment of optimal schedulers very complex. In order to alleviate this problem it has been proposed to partition the content beforehand among available sources [83], so that no duplicate packet can be delivered to the client.

In this chapter, we will use a different approach by considering rateless codes, or *Fountain codes*, in order to completely avoid the problem of coordinating the different transmission schedules at each server. We show that using rateless codes, it is feasible to efficiently stream layered media from multiple sources to a client with no need of coordination among the sending servers. At the same time, we make sure that each packet that is sent by any of the servers is not redundant for the client that receives it. This is in spirit similar to [84]. We however consider more realistic channel models that typically exhibit correlated loss patterns in the form of error bursts, as it is the case in most transmission scenarios. Indeed, whenever channel losses are a result of a network congestion, bursts of consecutive packets are typically discarded by the congested router.

Furthermore, we assign a cost to the transmission of each packet, which is charged to the client of the media stream. This packet cost will depend on the used channel.

Given this setup, we propose optimized sending schemes for a set of servers delivering a given media stream, and devise a heuristic-based algorithm that can provide close to optimum performance in realistic streaming scenarios. The proposed framework is generic and provides a low complexity distributed streaming solution. Building on the universal channel code properties of rateless codes, the system is able to adapt to channel losses, without adaptively transcoding the data at each sender, contrarily to [6].

The remainder of this chapter is organized as follows. In Section 6.2 we outline the considered framework and provide a brief introduction to rateless codes by the example of Raptor codes. In Section 6.3 we devise an optimization problem whose solution drives the optimal performance of the considered system. In Section 6.4, we finally provide and validate a distributed heuristic-based algorithm to solve the optimization problem under complexity constraints.

6.2 Framework

6.2.1 Network model

In an effort to make our notation transparent, we give an overview in Table (6.1). We will use bold-face letters to denote vectors, e.g. $\mathbf{a} = (a_1, \ldots, a_N)$. All vectors will be of dimension N, which is the number of peers/channels that are available to serve the client. For the sake of simplicity we will use the term *rate* to denote a number of packets per time unit. Hence we suppose packets of fixed

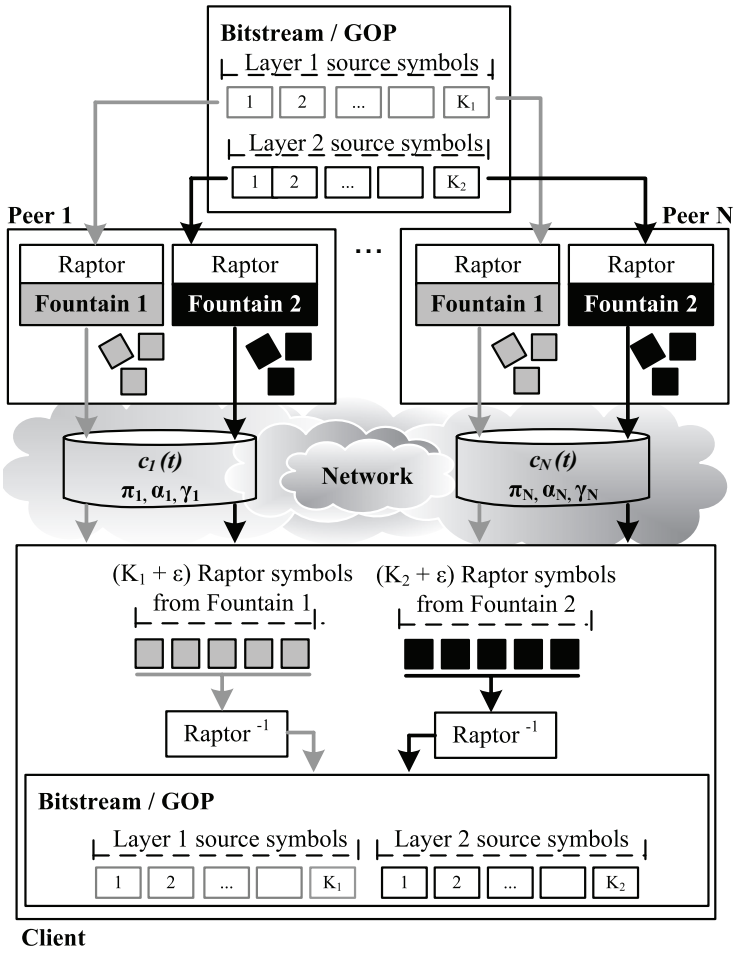

Figure 6.1: Streaming from multiple sources using Fountain Codes.

size, and rates (in bit/s) that are multiples of the packets size. In practice this assumption can be met by flexible packetization standards or by zero-padding.

v	video target rate
λ^j	rate of layer j
A	total allocated rate
a_n	rate allocated on channel n
c_n	maximum available rate on channel n
γ_n	cost of sending a packet on channel n
p_n, q_n	parameters of the loss process on channel n
π_n, α_n	packet loss rate (PLR) and average burst length (ABL) of channel n

Table 6.1: Notation.

Figure 6.1 gives an illustration of the framework we are describing in this

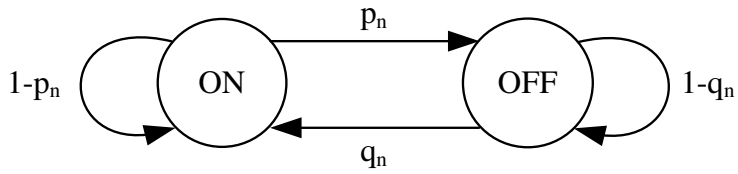

Figure 6.2: Two-state Markov model for channel n. A packet is correctly delivered if the chain is in state *ON*, and lost in state *OFF*.

section. A client wants to retrieve a layered media stream from the network. To do so, it has access to a set of N servers/peers, which all hold the layered video stream that needs to be transmitted. We suppose that the client can connect to each of the serving peers through a distinct channel. Our model also extends to partially disjoint paths, where the bandwidth on shared network segments is adequately distributed between the different packet flows. Each channel is characterized at each time instant t by a rate $c_n(t)$ at which it is able to transmit data. A pair of parameters (p_n, q_n) specifies both the packet loss rate π_n (PLR) and the average packet loss burst length α_n (ABL) observed when channel n is used. The cost that is charged to the client when a packet is transmitted over channel n reads as γ_n. As observed for example in the simulations for the media friendly rate allocation scheme proposed in Chapter 3, bursts happen regularly in the case of dynamic changes in the network topology, mostly due to temporarily congested routing nodes. The model that we use is able to catch these bursty loss patterns in a simple way. Note that we assume that the parameters $(\pi_n, \alpha_n, \gamma_n)$ do not depend on the transmission rate.

To summarize, we consider that each channel is governed by a Gilbert-Elliot model, explicited by a two-state Markov chain: in the *ON* state a packet is delivered to the client, whereas in the *OFF* state the packet is lost, see Figure(6.2). The transition probability matrix for channel n reads as:

$$P_n = \begin{bmatrix} 1-p_n & p_n \\ q_n & 1-q_n \end{bmatrix} \tag{6.1}$$

and a transition is triggered at each time that a packet is sent over the channel. Finally, referring to the outlined model, the PLR and ABL for channel n are given by the stationnary probability of being in the *OFF* state and by the average residence time in the *OFF* state respectively:

$$\pi_n = \frac{p_n}{p_n + q_n}, \tag{6.2}$$

$$\alpha_n = \frac{1}{q_n}. \tag{6.3}$$

6.2.2 Rateless Codes

With rateless codes, such as LT [85] and Raptor [86] codes, one can generate a potentially unlimited number of symbols from K original symbols. Ideal Raptor codes have the property of generating unique symbols with high probability, such

that any $(K + \epsilon)$ encoded symbols can be used to decode the original K source symbols. The notion of *Fountain code* comes from the analogy of a rateless code with a water Fountain (the unlimited number of symbols) from which any cup of volume $(K + \epsilon)$ satisfies the needs of the client. It does not matter which drops (symbols) of water it has obtained. Similarly, it does not matter if the received symbols from a digital fountain come from the same sender, as long as different senders have encoded the same input symbols. This property is key in order to use multiple uncoordinated senders to provide the same stream to a client. As long as the set of symbols they provide has been generated from the same input symbols, the encoded symbols will be different at each sender with high probability. This means that every delivered symbol in the system gives the same amount of novel information to the client.

In practice each encoded symbol is identified by an *Encoding Symbol ID*, or ESI. This ESI is typically transmitted along with the encoded Raptor symbol itself, and is used to synchronize the state of the Raptor decoder with the state of the Raptor encoder which has produced the symbol. As ESIs are coded on 2 bytes, the number of symbols that can be generated from a set of source symbols is limited to the number of available ESIs, thus providing a maximum of 2^{16} distinct encoded symbols. The symbol size T can range from 1 byte to several hundred bytes. If a *block* of K symbols of size T is encoded into a large number of encoded symbols of size T and if $1000 \leq K \leq 8192$, then the decoding overhead ϵ is typically of about 2 symbols. It is worth noting that Raptor codes induce linear complexity for both encoding and decoding, and therefore also allow for on-the-fly encoding if needed. For further details on Raptor codes and their implementation, we refer the interested readers to [86, 87, 88].

6.2.3 Coding scheme for layered media

A rateless code, applied blindly on a media bitstream, would mix the time-dependencies and the intra-layer dependencies that are present in the original scalable media stream. This would result in an encoded version of the media bitstream which can only be downloaded completely before consumption, but which can no longer be streamed. Indeed only if an amount of encoded symbols that equals at least the number of all the source symbols for all frames and layers is received, then the stream can be decoded.

In our proposed coding scheme, our goal is to keep each layer independently available for delivery. By doing so, a client is able to select which layers it wants to consume. In an effort to keep the content streamable, we propose to create one Fountain per layer and per Group of Pictures (GOP), which form independently decodable parts of the bitstream. By doing so, the client does not have to wait for all of the bitstream to be received before decoding the stream, but can start doing so after the reception of the first GOP. Moreover, the amount of data represented by a GOP is in practice sufficiently large for the use of Raptor codes to be efficient, as discussed in Section 6.2.2. The proposed coding scheme is illustrated in Figure 6.3. It allows to keep the hierarchical and temporal dependencies present in the original bitstream, which are essential for the scalable delivery of the stream. As long as the client receives $K + \epsilon$ distinct symbols on aggregate from all of the available sources, it will be able to decode the corresponding video data. Even in the case of practical Raptor codes, there are several ways to guarantee that each server sends different symbols from

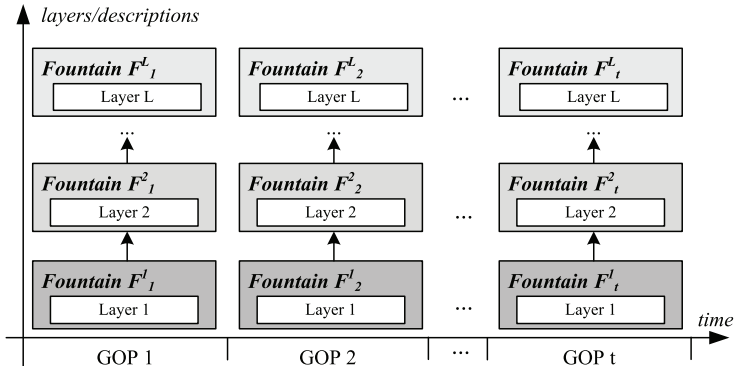

Figure 6.3: We encode each GOP of each layer into one Digital Fountain.

the same Fountain. For example, the requesting client can provide a different random seed to each of the sources, determining a subset of ESIs (and thus encoded symbols), which the server has to transmit. Another option could be to centrally encode a large number of symbols for each Fountain, and to put disjoint subsets on different servers that form a Content Distribution Network, or CDN.

In our framework, each network packet will carry symbols from a single Fountain, i.e., from a single layer. This is in contrast to the MD-FEC coding scheme proposed for example in [37]. In MD-FEC schemes, layers are unequally protected and each network packet carries information related to each one of the available layers. A similar packetization scheme using Raptor encoded layers has been proposed in [89]. These schemes inherently tend to over-protect lower layers of the media stream: even if enough packets have already been received to permit the decoding of a given layer, each additional packet that is received still carries parity information for each one of the available layers, hence reducing the overall goodput of the streaming application. The amount of data that is delivered for layers that are already decodable, is essentially of no use to the application. We will provide further details on how the Raptor symbols that are generated from one layer of a GOP are put into network packets when we present our simulation setup in Section 6.5.

Given the above considerations it is clear that the performance of the framework we propose will rely on the solution to two distinct problems:

- the rate a_n assigned to each peer needs to be split in an efficient manner among the available layers,
- we need to find a rate allocation $\mathbf{a}$ among the N available channels that maximizes the probability of receiving the number of symbols that is required for correct decoding.

6.2.4 Peer rate distribution

In this section, we show how a server should partition the streaming rate a_n it has been allocated, between the different layers of the video stream that is to be transmitted. Our objective is to keep servers synchronized while we do

not allow communication among them. Moreover we want to provide a robust distributed streaming solution that benefits from server diversity.

We suppose that L video layers are available for transmission, and each layer is characterized by its rate λ^l, $1 \leq l \leq L$ in symbols per GOP. In order to decode layer l, a Raptor decoder will need $\tilde{\lambda}^l = \lambda^l + \epsilon^l$ Raptor symbols, where ϵ^l is the *decoding overhead*. In order to keep the notation clear, we will use a slight abuse of notation in what follows, by supposing that the rate of each layer includes the decoding overhead, i.e., $\lambda^l = \tilde{\lambda}^l$. In the scenario we consider, a client selects the number of layers l^* out of the total L layers that are available. By doing so it is able to select the version of the video stream that best suits its needs. We suppose that the client has knowledge of the rates of each layer, λ^l, $1 \leq l \leq L$. In practice this meta information can be conveyed to the client by any server during a session initialization handshake for example.

The target video rate to v which has to be received by the client is given by:

$$v = \sum_{l=1}^{l^*} \lambda^l. \tag{6.4}$$

where λ^l denotes the rate that needs to be received in order to decode layer l. In order to receive this target rate v from the N available servers, the client allocates a rate a_n, $1 \leq n \leq N$ to each server, the sum of which we denote by A, the aggregate allocated rate. A is clearly also the sum of the rates allocated to each layer, A^l, $1 \leq l \leq l^*$:

$$A = \sum_{n=1}^{N} a_n = \sum_{l=1}^{l^*} A^l. \tag{6.5}$$

In the following section we will focus on how exactly $\mathbf{a}$ is computed. However, as the n channels that are available are lossy, it becomes clear that A includes the target rate v, plus some amount of redundant Raptor symbols, which are allocated to cope with the loss processes on the different channels. Out of this redundant rate $(A - v)$, we call ξ^l the redundancy that is allocated for layer l. We have:

$$A - v = \sum_{l=1}^{l^*} \xi^l. \tag{6.6}$$

Moreover, the total rate allocated to layer l can be expressed as:

$$A^l = \lambda^l + \xi^l, \;\; 1 \leq l \leq l^*. \tag{6.7}$$

In a distributed peer allocation scheme, each peer should fill its rate a_n in a similar way. By splitting the total rate allocated for video data and redundancy respectively among the allocated peer rates a_n, we can write:

$$a_n = a_n^v + a_n^\xi, \;\; 1 \leq n \leq N. \tag{6.8}$$

Here a_n^v and a_n^ξ denote both the video- and redundant rate that are allocated by peer n. By taking all of these considerations into account, we can finally write A as:

$$A = \sum_{n=1}^{N} \left(\sum_{l=1}^{l^*} \frac{\lambda^l}{v} a_n^v + \frac{\xi^l}{A - v} a_n^\xi \right), \tag{6.9}$$

which indicates that the distribution among layers of the allocated rate a_n at each peer n should respect the proportions of the source rates of each layer in the total video rate, as well as the proportions of the redundant rates for each layer in the total redundant rate.

6.3 Optimal Server Rate Allocation Problem

Let v be the target video rate that needs to be delivered to the client, including the Raptor overhead. Further let $P(v)$ be the probability of receiving at least the target rate. We want to find the rate allocation $\mathbf{a}^* = (a_1^*, \ldots, a_N^*)$ which achieves the optimal tradeoff between maximizing $P(v)$ and minimizing the resulting cost, i.e.:

$$\mathbf{a}^* = \underset{a_n \leq c_n, \forall n}{\arg\max} \left(P(v) - \theta \frac{\mathbf{a} \cdot \gamma^{\mathbf{T}}}{\mathbf{c} \cdot \gamma^{\mathbf{T}}} \right), \tag{6.10}$$

where $\gamma^{\mathbf{T}}$ denotes the transpose of the cost vector $\gamma = (\gamma_1, \ldots, \gamma_N)$, and θ is a Lagrangian factor.

As stated, the problem exhibits combinatorial complexity. It is not trivial to compute $P(v)$, which is the probability of receiving at least v packets on aggregate over the N channels, each of which is defined by a Gilbert-Elliot loss process. We can express this probability in terms of the corresponding probability density function (pdf):

$$P(v) = 1 - \sum_{j=0}^{v-1} p_A(j), \tag{6.11}$$

where $p_A(j)$ is the probability of correctly receiving at least j out of the A packets that are transmitted on aggregate over the N independent channels. This can in turn be computed by the convolution of the N probability density functions that give, for each channel n, the probability of correctly receiving i out of the a_n packets:

$$p_A = \bigotimes_{n=1}^{N} p_{a_n}. \tag{6.12}$$

Note that, in contrast to the i.i.d. and uniform case [90], there is no analytical form to express p_{a_n} in the case of a bursty loss channel, especially if the number of packets that are transmitted is relatively small and the pdf is thus poorly approximated by a Normal density. These probability density functions can however be computed using the exact but iterative solution proposed by [91], at the price of increased computational complexity. In the next section, we however propose a suboptimal rate allocation solution, which achieves close to optimal performance with a reduced complexity.

6.4 Heuristics based Algorithm

We will now introduce several heuristic choices, which allow for the design of a low complexity distributed allocation scheme.

6.4.1 Heuristic Peer Rate Distribution

We choose to introduce the following approximation in the peer rate allocation that we have introduced in Equation (6.9):

$$\frac{\xi^l}{A-v} = \frac{\lambda^l}{v}, \;\; 1 \leq l \leq l^*. \tag{6.13}$$

This allows to rewrite Equation (6.9) as:

$$A = \sum_{n=1}^{N} \left(\sum_{l=1}^{l^*} \frac{\lambda^l}{v} a_n \right), \tag{6.14}$$

which indicates that the distribution among layers of the allocated rate a_n at each peer n should be proportional to the relative size of every layer in the target rate v. Our results indicate that this heuristic choice provides robust delivery of the l^* transmitted layers. It is important to note that each peer knows the sizes of each layer λ^l. As long as each peer knows both the rate a_n that has been allocated to it and the target rate v, it is able to distribute a_n among the l^* layers in a completely distributed way. We refer to Figure 6.4 for an illustration.

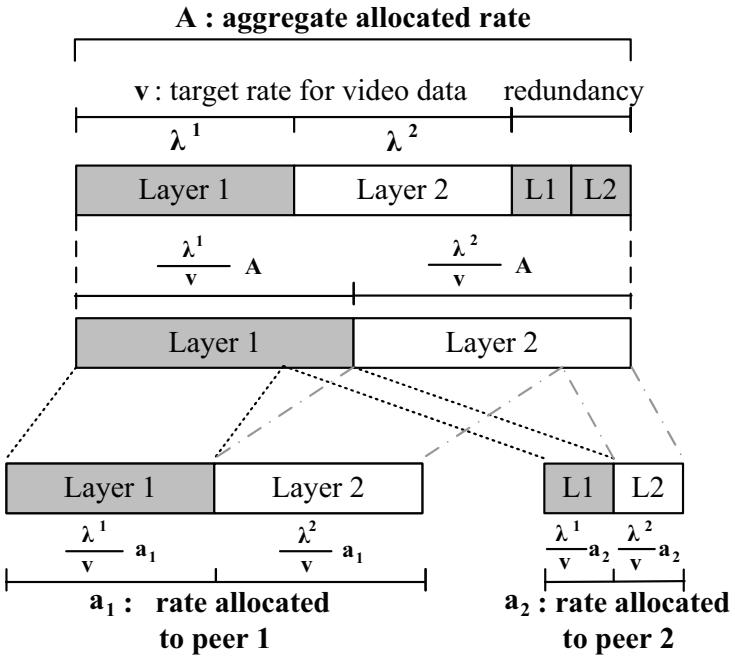

Figure 6.4: The total allocated rate is split between layers in a distributed way.

6.4.2 Heuristic Server Rate Allocation

As there is no simple constructive algorithm to find the optimal solution to the optimal rate allocation problem, we propose a low complexity rate allocation

algorithm based on heuristics. This algorithm is run at the client, which is offered a set of N channels to the servers, along with their loss parameters and maximum transmission rates. The client has to determine the rates that need to be sent from each of the streaming peers, and communicate the results to the servers. The allocation problem can be decomposed as follows :

- the client should first classify channels, in order to select which channel is *good*, in the sense that using it minimizes the resulting overall cost, while it maximizes the likelihood of receiving potentially allocated packets.
- it then determines how these channels should be used. It uses in priority the *good* channels, and fills them up gradually by updating a_n, until the joint likelihood of receiving the allocated packets satisfies

$$\sum_{n=1}^{N} a_n \overline{P}(a_n) \geq v P^T, \tag{6.15}$$

  where P^T is the target probability of success, and $\overline{P}(a_n)$ is the probability of receiving the a_n allocated packets on channel n.

In the remainder of this section, we present two methods for classifying *good* channels, and we validate the respective heuristic-based algorithms in different streaming scenarios. Throughout this section we will assume that $\theta = 1$ for the sake of clarity.

6.4.3 Baseline

We first consider a *baseline* algorithm where channels are simply classified according to the average loss probability [90], and the transmission cost. In other words, the channels are sorted according to $\frac{1-\pi_n}{\gamma_n}$.

The average burst length is not considered in the baseline scheme, and the probability of receiving a_n packets is approximated as :

$$\overline{P}(a_n) = (1-\pi_n)^{a_n}. \tag{6.16}$$

Since the probability $\overline{P}(a_n)$ is decreasing with a_n, a selected channel is fully used, unless the termination constraint is met earlier. If the network model in use would assume an i.i.d. uniform loss process instead of correlated losses, this baseline algorithm would represent a good heuristic choice.

6.4.4 ABL based Algorithm

The second algorithm is based on a more accurate estimate of the probability of receiving all of the allocated packets over channel n. It considers the burstiness of the loss process. Indeed, based on the Gilbert-Elliot model, the probability of receiving *all* the a_n packets is given by:

$$\overline{P}(a_n) = (1-\pi_n)(1-p_n)^{a_n-1}. \tag{6.17}$$

It becomes clear in this case that a simple water-filling algorithm that considers the loss probability π_n, but not the average length of bursts of errors, will fail

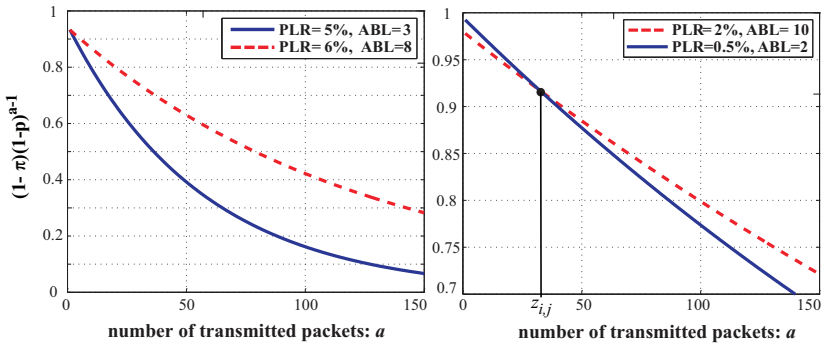

Figure 6.5: Two cases that show the importance of the ABL. The plots show $(1-\pi)(1-p)^{a-1}$ on the y-axis, and the number of sent packets c on the x-axis. *Left:* the probability of receiving all of the sent packets is consistently higher for the channel with *higher* PLR. *Right:* The probability curves for 2 channels may intersect.

in selecting the *good* channels first. For example, a channel with high PLR can provide consistently high success probabilities (see Figure 6.5-left) depending on the respective ABL values. There are even cases where the choice of a particular channel depends on the number of packets to be sent (see Figure 6.5-right). We propose to take this phenomenon into account in the selection of the *good* channels. We first classify the channels according to $\frac{\overline{P}(1)}{\gamma_n}$, with $\overline{P}(1)$ as given by Equation (6.17), in a similar way as the baseline algorithm. However, this hierarchy is now allowed to vary with the number of packets assigned to the channels. Using Equation (6.17) and considering a pair of channels (i, j) we compute the number of packets $z_{i,j}$ at which the hierarchical order of channels i and j in the channel ordering changes. The intersection points are computed as:

$$\frac{(1-\pi_i)(1-p_i)^{z_{i,j}-1}}{\gamma_i} = \frac{(1-\pi_j)(1-p_j)^{z_{i,j}-1}}{\gamma_j} \tag{6.18}$$

$$z_{i,j} = 1 + \frac{\log\left(\frac{1-\pi_j}{1-\pi_i}\right) + \log\left(\frac{\gamma_i}{\gamma_j}\right)}{\log\left(\frac{1-p_i}{1-p_j}\right)} \tag{6.19}$$

It should be noted that there is at most one such switching point for each channel pair in the rate interval of relevance $[0, max(c_i, c_j)]$. Also, as $z_{i,j} = z_{j,i}$, we only need to compute at total of $\frac{N(N-1)}{2}$ switching points. The rest of the algorithm is a straightforward generalization of the water-filling method, with the additional step however that, while filling channel i, we switch the ongoing filling operation entirely to channel j if we need to allocate more than $z_{i,j}$ packets. The algorithm proceeds until the stopping criterion of Eq. (6.15) is satisfied. A pseudo-code version of the algorithm is given in Algorithm 5.

Algorithm 5 ABL-based algorithm

Require: Compute z_{ij} for each channel pair $i \neq j$.
Require: Set $z_{ij} = 0$ if it falls outside the allowable rate range.
Require: $a_n \Leftarrow 0$, initialize rate allocation.
Require: TERMINATION: Equation (6.15).
Require: $flag \Leftarrow 0$, 1 if we switched to the current channel.

while TERMINATION == FALSE AND $SUM(a_n) < SUM(c_n)$ **do**
  if $flag == 0$ **then**
    $s \Leftarrow \arg\ \max_{1 \leq n \leq N} \frac{\overline{P}(1)}{\gamma_n}$
  end if
  $s\_switch \Leftarrow \min_j z_{s,j}$
  $flag \Leftarrow 0$
  if $z_{s,s\_switch} > 0$ **then**
    $a_s \Leftarrow z_{s,s\_switch}$
    if TERMINATION == FALSE **then**
      $a_s \Leftarrow 0$
      $s \Leftarrow s\_switch$
      $flag \Leftarrow 1$
    else
      $a_s \Leftarrow 0$
      while TERMINATION == FALSE AND $a_s < c_s$ **do**
        $a_s \Leftarrow a_s + 1$
      end while
    end if
  else
    while TERMINATION == FALSE AND $a_s < c_s$ **do**
      $a_s \Leftarrow a_s + 1$
    end while
Require: Exclude s from the set of availalble channels.
  end if
end while

Algo	prob	cost	metric
Baseline (eq)	$2.1 \cdot 10^{-2}$	n/a	n/a
ABL-based (eq)	$6.8 \cdot 10^{-4}$	n/a	n/a
Baseline (rand)	$6.5 \cdot 10^{-4}$	$2.1 \cdot 10^{-4}$	0.023
ABL-based (rand)	$5.0 \cdot 10^{-4}$	$4.48 \cdot 10^{-4}$	0.041

Table 6.2: Difference between heuristics-based algorithms and optimal rate allocation [MSE].

6.4.5 Validation

In order to validate both algorithms, we have considered a scenario in which a client has access to three serving peers, each offering to provide a maximum rate of $c_n = 40$ packets. The target rate to be received has been set to be equivalent to $v = 30$ packets. We have computed the optimal rate allocations for 500 realizations on the three channels, with respect to Equation (6.10).

In each realization, the PLR for the three channels has been randomly selected in the interval from 1% to 10% with uniform probability. Similarly the ABL for each realization has been randomly select for each channel in the interval from 2 to 20 with uniform probability. We have also computed the rate allocation given by both the baseline algorithm and the ABL-based algorithm, in each network realization. In order to verify whether the optimal channels are chosen by the algorithms, we constrain the heuristics-based solutions to allocate a total number of packets that is equal to the total number of packets used under the optimal rate allocation. The distribution of the rate among the different channels may however differ significantly. The resulting rate allocations have been used to compute the actual success probability, the induced cost and the value of the optimization metric according to Equation (6.10). In Table 6.2 we report the average difference (in MSE) for the values of these metrics with respect to the performance of the optimal rate allocation, over 500 realizations. We propose two different sets of simulations: *i)* in the first set (eq), all channel costs are always equal, *ii)* in the second set (rand), the costs for each channel are randomly select between 0.01 and 1 with uniform probability for each realization (with $\theta = 1$). Based on these experiments, we conclude that the algorithm we propose provides a robust channel selection algorithm when the channels are characterized by both ABL and PLR, and that consideration of the ABL in the channel selection is quite important. In our simulations, different transmission costs for the available channels have tended to show a higher impact on the channel selection than the respective loss processes.

However, in a realistic scenario none of the two algorithms is aware of how many packets need to be allocated on either of the chosen channels. Both algorithms gradually fill the chosen channels until the termination condition given by Equation (6.15) is satisfied. As the baseline algorithm does not take into account the ABL, it makes a consistent error in estimating the probability of receiving the allocated packets: hence it does not know when to terminate. This behavior is illustrated using a simple yet representative example in Figure 6.6. The considered scenario is as follows. There are 3 available peers, each willing to transmit a maximum of 60 packets. The loss parameters for the 3 channels are ($\pi_1 = 5\%,\ \alpha_1 = 3$), ($\pi_2 = 6\%,\ \alpha_2 = 8$) and ($\pi_3 = 7\%,\ \alpha_1 = 5$)

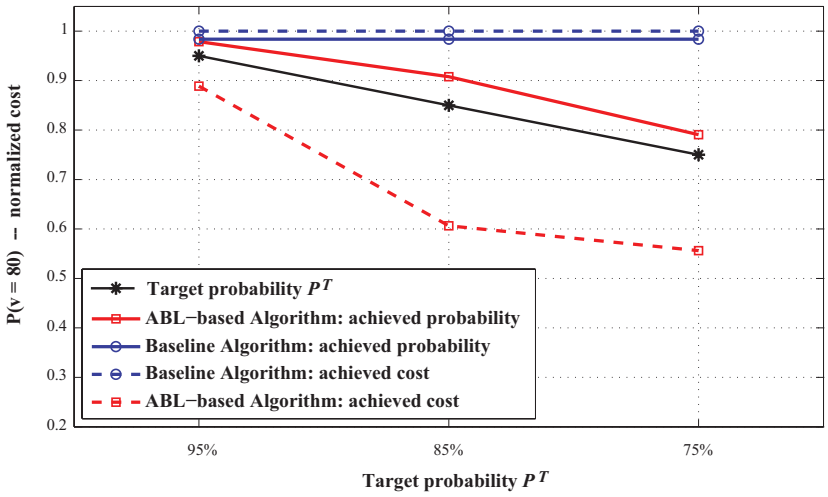

Figure 6.6: As opposed to the baseline algorithm, the ABL-based algorithm is capable of terminating correctly.

respectively. We vary the target probability of success P^T in the right-hand side of Equation (6.15) from 95% down to 75%. It can be observed that the baseline algorithm always ends up using all the available channels at full rate and implying maximal cost. The ABL-based algorithm, through its more accurate estimation of the success probability, is able to tune the amount of packets that is injected in the network so that the actual success probability induced by the resulting rate allocation follows the target probability P^T.

6.5 Simulation Results

We have encoded the SOCCER test sequence (CIF, 30Hz) using the H.264-SVC reference codec into a base layer ($\simeq 300kbps$) and one SNR-enhancement layer ($\simeq 700kbps$). The GOP size is set to 32 frames. Hence the average size in bytes is equal to 40 000 for a baselayer GOP, while it is 93 334 for an enhancement layer GOP. Each GOP and layer of the bitstream has then been encoded into a digital fountain using a Raptor code, where the Raptor symbol size T has been set to 16 bytes. This corresponds thus to source block sizes K of 2 500 and 5 834 for both layers respectively.

We have chosen the network packet size to a be a multiple of 18, so that each packet can hold an integer number of Raptor symbols, including the ESIs for each symbol. Specifically we use packets of size 1440 bytes, containing a maximum of 80 Raptor symbols each. Given the above considerations, the average number of such packets that need to be received for both layers in order to allow for correct decoding with high probability is thus given by:

$$\lambda^1 = \left\lceil \frac{2\ 500}{80} \right\rceil = 32 \tag{6.20}$$

Algo	Y-PSNR [dB]	cost	redundancy [%]
Baseline	38.75	141	34.28
ABL-based	38.75	120	14.28
Optimal	38.75	117	9.52

Table 6.3: PSNR, inferred cost and received redundancy for the two proposed algorithms and the optimal rate allocation.

for the baselayer and

$$\lambda^2 = \left\lceil \frac{5\ 834}{80} \right\rceil = 73 \tag{6.21}$$

for the enhancement layer.

The client retrieves the stream by aggregating 3 available channels that all have a capacity of 512 kbps. Clearly, only the base layer could possibly be transmitted in the absence of channel aggregation, resulting in a maximum average (Y)-PSNR of 33.03dB.

Two of the available channels have similar loss characteristics given by the tuple ($\pi_{1,2} = 3\%,\ \alpha_{1,2} = 5$), whereas the third channel is given by ($\pi_3 = 1\%,\ \alpha_3 = 6$). The cost for sending a packet on either channel is equal to $\gamma_{1,2,3} = 1$.

Following the above considerations, both layers can be correctly decoded whenever 105 packets have been received, 32 of which originate from the base layer fountain, the remaining 73 from the enhancement layer fountain. We have run both rate allocation algorithms on the above scenario, using $v = 105$ and $P^T = 0.94$, and all three senders have split their respective allocated rate among the 2 layers as given by Equation (6.14).

As both algorithms tend to over-provision the system, it is not surprising that the client is always able to retrieve the 2 available layers over the 3 channels with high probability, thus receiving the best available quality. However, as the baseline algorithm underestimates the reception probability of the allocated packets, it does not evaluate the termination condition of the greedy algorithm correctly. It hence allocates all the available resources to the streaming process, resulting in a waste of bandwidth.

The results are given in Table (6.3) where the optimal allocation given by Equation (6.10) is provided as a reference. The *cost* column indicates the number of packets injected into the network. The *redundancy* column shows the percentage of redundant rate that is transmitted for either rate allocation, as compared to the 105 packets that are necessary for correct decoding. Note that in this case, the redundancy ratio for the baseline algorithm is only bounded through the available channel capacity, as the algorithm does not terminate correctly. These results confirm that, using the ABL-based algorithm, the proposed heuristics based greedy algorithm is able to provide close to optimal performance for delivering scalable encoded media streams.

We have simulated the above transmission scenario 100 times using each time different realizations of each of the 3 used channel processes. Using the proposed rate allocation, the client was successful in decoding all of the transmitted GOPs in 97 of the runs.

6.6 Conclusions

We have presented a low-complexity framework for the delivery of a given layered media stream from multiple senders to a single receiver, over channels that present correlated packet loss patterns. As our distributed streaming system relies on the aggregate reception of a sufficient number of packets for correct decoding, we have devised an optimization problem whose solution gives the optimal rate allocation among channels that maximizes the probability of lossless decoding. We have provided and validated a heuristics-based algorithm, which is able to quickly provide a sub-optimal solution to the combinatorial optimization problem. Our findings finally indicate that it is important to consider both the Packet Loss Ratio (PLR) and Average Burst Length (ABL) when addressing channel selection and rate allocation problems in multipath routing, or rate aggregation over bursty channels.

Chapter 7

Conclusions

7.1 Contributions

In this thesis we have addressed several problems in order to improve the delivery of scalable video streams over the network architecture that is provided by today's Internet.

First, we have proposed a distributed and completely scalable media-friendly rate and congestion control algorithm. By explicitly considering the rate needs of the media streams during the rate allocation process, our algorithm achieves rate allocations that are more meaningful for the streaming applications. We have introduced our rate and congestion control algorithm by considering the case of delivering video streams that have the FGS property, which show all the characteristics that are needed to adapt the streams to the allocated rates. Through extensive simulations we have shown that our algorithm maximizes the average received quality for all the streams in the network, and is able to quickly adapt to network dynamics.

We have then considered the practically important case where streams that do not have the FGS property have to be delivered. This analysis has resulted in important conditions on the encoding rates of layered scalable video streams. An in-depth analysis of practical considerations has led us to derive an efficient scheduling algorithm, which, in conjunction with our rate allocation algorithm, enables the efficient distribution of layered video streams.

Second, we have considered broadcast scenarios where a layered video stream is transmitted to a heterogeneous client population. In this population, different clients potentially consume different layers of the offered stream. We have shown that minimal buffer occupancy can be jointly achieved at all the receivers and have further considered the problem of joint playback delay minimization for all the receivers. We have proposed an optimal scheduling algorithm and a simple adaptive scheduling scheme which proves that close to optimal performance can be achieved in practice.

Finally, we have exploited the inherent network diversity that is provided by the Internet infrastructure for the robust delivery of scalable media streams. Our framework is based on the use of rateless codes, which allow us to eliminate the need of coordination between the different servers. Furthermore, our coding scheme allows to adapt the version of the stream that is delivered to both the

client's needs and the network availability.

7.2 Future directions

While we did not make strong assumptions on the physical communications layer, we have not considered the specificities of wireless networks. It should be interesting to examine under which conditions the rate and congestion control algorithm that we have proposed can be implemented in wireless scenarios, where packet losses can no longer be used to explicitly compute congestion signals.

The possibility to use the media-friendly rate allocation algorithm we have presented in conjunction with coarse layered video stream marks a first and important result in this direction. However it can be noted that we rely on a particular traffic model when the RCC algorithm is designed, as given by FGS encoded streams for example. We then propose conditions under which traffic from coarse layered video streams can be adapted to fit the characteristics of this traffic model. It may be interesting to inherently use a coarsely layered traffic model in order to derive a media-friendly RCC algorithm. This would in fact imply to recast the Network Utility Maximization problem as posed by Kelly [3] as an integer optimization problem, leading potentially to different results.

Considering the distributed streaming framework presented in Chapter 6, future directions should include extensions to apply the presented work to Peer-to-Peer streaming architectures. To that aim, optimal content replication and caching of received Raptor symbols should be analyzed in dynamic network settings, as well as the possibility to apply Network Coding to the video information using rateless codes.

Bibliography

[1] V. Jacobson, "Congestion avoidance protocol," *ACM SIGCOMM*, pp. 314–329, 1988.

[2] M. Handley, S. Floyd, J. Padhye, and J. Widmer, "Rfc3448: Tcp friendly rate control (tfrc): Protocol specification," *Internet RFCs*, Jan 2003.

[3] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, pp. 237–252(16), Mar 1998.

[4] T. Nguyen and A. Zakhor, "Multiple sender distributed video streaming," *IEEE Transactions on Multimedia*, Jan 2004.

[5] T. Nguyen and A. Zakhor, "Distributed video streaming over internet," *IEEE International Conference on Multimedia Computing and Networking (MMCN)*, 2002.

[6] A. Majumdar, R. Puri, and K. Ramchandran, "Distributed multimedia transmission from multiple servers," *IEEE International Conference on Image Processing (ICIP)*, vol. 3, pp. 177–180, 2002.

[7] C. Gkantsidis and P. Rodriguez. . . , "Network coding for large scale content distribution," *IEEE INFOCOM*, Jan 2005.

[8] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The bittorrent p2p file-sharing system: Measurements and analysis," *International Workshop on Peer-to-Peer Systems (IPTPS)*, Jan 2005.

[9] J. Chakareski, E. Setton, Y. Liang, and B. Girod, "Video streaming with diversity," *IEEE International Conference on Multimedia and Expo (ICME)*, Jan 2003.

[10] M. Hefeeda, A. Habib, D. Xu, B. Bhargava, and B. Botev, "Promise: peer-to-peer media streaming using collectcast," *ACM International Multimedia Conference*, pp. 45–54, Nov 2003.

[11] V. N. Padmanabhan, H. J. Wang, and P. A. Chou, "Supporting heterogeneity and congestion control in peer-to-peer multicast streaming," *International Workshop on Peer-to-Peer Systems (IPTPS)*, vol. 3279, pp. 54–63, Feb 2004.

[12] Z. Xiang, Q. Zhang, Q. Zhu, Z. Zhang, and Y.-A. Zhang, "Peer-to-peer based multimedia distribution service," *IEEE Transactions on Multimedia*, vol. 6, pp. 343–355, Apr 2004.

[13] V. Agarwal and R. Rejaie, "Adaptive multi-source streaming in heterogeneous peer-to-peer networks," *Conference on Multimedia Computing and Networking (MMCN)*, pp. 13–25, Jan 2005.

[14] D. Jurca, J. Chakareski, J. Wagner, and P. Frossard, "Enabling adaptive video streaming in p2p systems," *IEEE Communications Magazine*, vol. 45, pp. 108–114, Jan 2007.

[15] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," *ACM International Worshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pp. 177–186, May 2002.

[16] K. L. Calvert, S. Bhattacharjee, E. W. Zegura, and J. Sterbenz, "Directions in active networks," *IEEE Communications Magazine*, vol. 35, pp. 72–78, Oct 1998.

[17] J. Apostolopoulos and M. Trott, "Path diversity for enhanced media streaming," *IEEE Communications Magazine*, Jan 2004.

[18] J. Apostolopoulos, T. Wong, W. Tan, and S. Wee, "On multiple description streaming with content delivery networks," *IEEE INFOCOM*, Jun 2002.

[19] J. Chakareski and B. Girod, "Server diversity in rate-distortion optimized media streaming," *IEEE International Conference on Image Processing (ICIP)*, Jan 2003.

[20] J. Moy, "Rfc2328: Ospf version 2," *Internet RFCs*, Jan 1998.

[21] L. Massoulie and J. Roberts, "Bandwidth sharing: Objectives and algorithms," *IEEE INFOCOM*, vol. 3, pp. 1395–1403, Jun 1999.

[22] D. Katabi, M. Handley, and C. Rohrs, "Internet congestion control for future high bandwidth-delay product environments," *ACM SIGCOMM*, Jan 2002.

[23] S. Low, L. Andrew, and B. Wydrowski, "Understanding xcp: Equilibrium and fairness," *IEEE INFOCOM*, Jan 2005.

[24] S. Kunniyur and R. Srikant, "End-to-end congestion control schemes: Utility functions, random losses and ecn marks," *IEEE INFOCOM*, vol. 3, pp. 1323–1332, 2000.

[25] S. Futemma, K. Yamane, and E. Itakura, "Tfrc-based rate control scheme for real-time jpeg 2000 video transmission," *IEEE Consumer Communications and Networking Conference (CCNC)*, Jan 2005.

[26] R. Johari and D. Tan, "End-to-end congestion control for the internet: Delays and stability," *IEEE/ACM Transactions on Networking*, vol. 9, no. 6, pp. 818–832, 2001.

[27] Z. Cao and E. W. Zegura, "Utility max-min: an application-oriented bandwidth allocation scheme," *IEEE INFOCOM*, vol. 2, pp. 793–901, Mar 1999.

[28] W. H. Wang and S. H. Low, "Application-oriented flow control: Fundamentals, algorithms and fairness," *IEEE/ACM Transactions on Networking*, vol. 14, pp. 1282–1292, Dec 2006.

[29] M. Chiang, S. Zhang, and P. Hande, "Distributed rate allocation for inelastic flows: Optimization frameworks, optimality conditions, and optimal algorithms," *IEEE INFOCOM*, vol. 4, pp. 2679–2690, Mar 2005.

[30] J. He, M. Bessler, M. Chiang, and J. Rexford, "Towards robust multi-layer traffic engineering: Optimization of congestion control and routing," *IEEE Journal on Selected Areas in Communications*, Jan 2007.

[31] Y. Zhang, D. Leonard, and D. Loguinov, "Jetmax: Scalable max-min congestion control for high-speed heterogeneous networks," *IEEE INFOCOM*, pp. 1–13, Apr 2006.

[32] M. Dai, D. Loguinov, and H. Radha, "Rate-distortion analysis and quality control in scalable internet streaming," *ACM International Worshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Jan 2003.

[33] M. Dai and D. Loguinov, "Analysis of rate-distortion functions and congestion control in scalable internet video streaming," *ACM International Worshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pp. 60–69, Jun 2003.

[34] M. Dai, D. Loguinov, and H. M. Radha, "Rate-distortion analysis and quality control in scalable internet streaming," *IEEE Transactions on Multimedia*, vol. 8, pp. 1135–1146, Dec 2006.

[35] S. Kwon, A. Tamhankar, and K. Rao, "Overview of h. 264/mpeg-4 part 10," *Journal of Visual Communication and Image Representation*, Jan 2006.

[36] Y. Wang, S. Wenger, J. Wen, and A. Katsaggelos, "Error resilient video coding techniques," *IEEE Signal Processing Magazine*, Jan 2000.

[37] R. Puri and K. Ramchandran, "Multiple description source coding using forward error correction codes," *Asilomar Conference on Signals, Systems and Computers*, Jan 1999.

[38] M. Wien, H. Schwarz, and T. Oelbaum, "Performance analysis of svc," *IEEE Transactions on Circuits and Systems for Video Technology*, Jan 2007.

[39] T. Stockhammer and M. Hannuksela, "H. 264/avc video for wireless transmission," *IEEE Wireless Communications*, Jan 2005.

[40] T. Stockhammer, G. Liebl, and M. Walter, "Optimized h. 264/avc-based bit stream switching for mobile video streaming," *EURASIP Journal on Applied Signal Processing*, Jan 2006.

[41] W. Li, "Overview of fine granularity scalability in mpeg-4 video standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, pp. 301–317, Mar 2001.

[42] W. Feng, "Rate-constrained bandwidth smoothing for the delivery of stored video," *IS&T/SPIE Conference on Multimedia Networking and Computing*, pp. 316–327, Feb 1997.

[43] J. Zhang and J. Hui, "Applying traffic smoothing techniques for quality of service control in vbr video transmissions," *Computer Communications*, vol. 21, pp. 375–389, Apr 1998.

[44] J. Rexford and D. Towsley, "Smoothing variable-bit-rate video in an internetwork," *IEEE/ACM Transactions on Networking*, vol. 7, pp. 202–215, Apr 1999.

[45] J. M. McManus and K. W. Ross, "Video-on-demand over atm: constant-rate transmission and transport," *IEEE Journal on Selected Areas in Communications*, vol. 14, pp. 1087–1098, Aug 1996.

[46] T. Stockhammer, H. Jenkac, and G. Kuhn, "Streaming video over variable bit-rate wireless channels," *IEEE Transactions on Multimedia*, vol. 6, pp. 268–277, Apr 2004.

[47] P. Thiran and J.-Y. L. Boudec, *Network Calculus*, vol. 2050. 2001.

[48] J. L. Boudec and O. Verscheure, "Optimal smoothing for guaranteed service," *IEEE/ACM Transactions on Networking (TON)*, Jan 2000.

[49] P. Thiran, J.-Y. L. Boudec, and F. Worm, "Network calculus applied to optimal multimedia smoothing," *IEEE INFOCOM*, vol. 3, pp. 1474–1483, Apr 2001.

[50] O. Verscheure, P. Frossard, and J. L. Boudec, "Joint smoothing and source rate selection for guaranteed service networks," *IEEE INFOCOM*, Jan 2001.

[51] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven layered multicast," *ACM SIGCOMM*, vol. 26, pp. 117–130, Aug 1996.

[52] Q. Zhang, Q. Guo, W. Zhu, and Y.-Q. Zhang, "Sender-adaptive and receiver-driven layered multicast for scalable video over the internet," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, pp. 482–495, Apr 2005.

[53] D. Saparilla and K. W. Ross, "Optimal streaming of layered video," *IEEE INFOCOM*, vol. 2, pp. 737–746, Mar 2000.

[54] R. Rejaie, M. Handley, and D. Estrin, "Layered quality adaptation for internet video streaming," *IEEE Journal on Selected Areas in Communications*, vol. 18, pp. 2530–2543, Jan 2000.

[55] M. Zink, J. Schmitt, and R. Steinmetz, "Layer-encoded video in scalable adaptive streaming," *IEEE Transactions on Multimedia*, vol. 7, pp. 75–84, Jan 2005.

[56] M. Zink, O. Kuenzel, J. Schmitt, and R. Steinmetz, "Subjective impression of variations in layer encoded videos," *IEEE/IFIP International Workshop on Quality of Service*, pp. 134–154, 2003.

[57] M. Zink, J. Schmitt, and C. Griwodz, "Layer-encoded video streaming a proxy's perspective," *IEEE Communications Magazine*, Jan 2004.

[58] P. de Cuetos and K. W. Ross, "Optimal streaming of layered video: joint scheduling and error concealment," *ACM International Multimedia Conference*, pp. 55–64, 2003.

[59] P. de Cuetos and K. W. Ross, "Unified framework for optimal video streaming," *IEEE INFOCOM*, vol. 3, pp. 1479–1489, Mar 2004.

[60] Z. Miao and A. Ortega, "Optimal scheduling for streaming of scalable media," *Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 1357–1362, Oct 2000.

[61] P. A. Chou and Z. Miao, "Rate-distortion optimized streaming of packetized media," *IEEE Transactions on Multimedia*, vol. 8, pp. 390 –404, Apr 2006.

[62] Y. Lee, Y. Altunbasak, and R. Mersereau, "Optimal packet scheduling for multiple description coded video transmissions over lossy networks," *IEEE Global Telecommunications Conference (GLOBECOM)*, Jan 2003.

[63] F. P. Kelly, "Charging and rate control for elastic traffic," *European Transactions on Telecommunications*, vol. 8, pp. 33–27, 1997.

[64] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 556–567, 2000.

[65] Y. Zhang, S. R. Kang, and D. Loguinov, "Delayed stability and performance of distributed congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 34, pp. 307–318, Aug 2004.

[66] P. Ranjan, R. J. La, and E. H. Abed, "Global stability conditions for rate control with arbitrary communication delays," *IEEE/ACM Transactions on Networking*, vol. 14, pp. 94–107, Feb 2006.

[67] R. J. La and V. Anantharam, "Utility-based rate control in the internet for elastic traffic," *IEEE/ACM Transactions on Networking*, vol. 10, pp. 272–286, Apr 2002.

[68] Y. Zhang and D. Loguinov, "Local and global stability of symmetric heterogeneously-delayed control systems," *IEEE Conference on Decision and Control (CDC)*, 2004.

[69] T. Alpcan and T. Basar, "A globally stable adaptive congestion control scheme for internet-style networks with delay," *IEEE/ACM Transactions on Networking*, vol. 13, pp. 1261–1274, Dec 2005.

[70] J.-W. Lee, R. R. Mazumdar, and N. B. Shroff, "Non-convex optimization and rate control for multi-class services in the internet," *IEEE/ACM Transactions on Networking*, vol. 13, pp. 827–840, Aug 2005.

[71] L. Ying, E. Dullerud, and R. Srikant, "Global stability of internet congestion controllers with heterogeneous delays," *American Control Conference*, vol. 4, pp. 2948–2953, Jun 2004.

[72] L. Massoulie, "Stability of distributed congestion control with heterogeneous feedback delays," *IEEE Transactions on Automatic Control*, vol. 47, pp. 895–902, Jun 2002.

[73] Y. Zhang, S. R. Kang, and D. Loguinov, "Delay-independent stability and performance of distributed congestion control," *IEEE/ACM Transactions on Networking*, vol. 15, Dec 2007.

[74] J. Yan, K. Katrinis, M. May, and B. Plattner, "Media- and tcp-friendly congestion control for scalable video streams," *IEEE Transactions on Multimedia*, vol. 8, pp. 196–206, Apr 2006.

[75] J. Chakareski and P. Frossard, "Rate-distortion optimized bandwidth adaptation for distributed media delivery," *IEEE International Conference on Multimedia and Expo (ICME)*, pp. 763–766, 2005.

[76] J.-P. Wagner and P. Frossard, "Playback delay optimization in scalable video streaming," *IEEE International Conference on Multimedia and Expo (ICME)*, pp. 860–863, Jul 2005.

[77] A. Gersho and R. M. Gray, "Vector quantization and signal compression," *Springer International Series in Engineering and Computer Science*, Jan 1992.

[78] S. Bakthavachalu and M. Labrador, "Tfrc friendliness and the case of ecn," *International Journal of Communication Systems*, Jan 2004.

[79] H. S. Cho and J. Lee, "Atfrc: Adaptive tcp friendly rate control protocol," *Information Networking: Networking Technologies for Enhanced ...*, Jan 2003.

[80] W. Zhao and S. K. Tripathi, "Bandwidth-efficient continuous media streaming through optimal multiplexing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 27, no. 1, pp. 13–22, 1999.

[81] "Momusys code, mpeg-4 verification model version 18.0," Jan 2001.

[82] "Nistnet network emulator."

[83] J. Chakareski and P. Frossard, "Distributed streaming via packet partitioning," *IEEE International Conference on Multimedia and Expo (ICME)*, Jan 2006.

[84] C. Wu and B. Li, "rstream: Resilient and optimal peer-to-peer streaming with rateless codes," *ACM International Multimedia Conference*, 2005.

[85] M. Luby, "Lt codes," *IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 271–282, 2002.

[86] A. Shokrollahi, "Raptor codes," *IEEE/ACM Transactions on Networking*, vol. 52, no. 6, pp. 2551–2567, 2006.

[87] J. Afzal, T. Stockhammer, T. Gasiba, and W. Xu, "System design options for video broadcasting over wireless networks," *IEEE Consumer Communications and Networking Conference (CCNC)*, Jan 2006.

[88] M. Luby, M. Watson, T. Gasiba, T. Stockhammer, and W. Xu, "Raptor codes for reliable download delivery in wireless broadcast systems," *IEEE Consumer Communications and Networking Conference (CCNC)*, Jan 2006.

[89] T. Schierl, K. Gänger, T. Stockhammer, and T. Wiegand, "Svc-based multi source streaming for robust video transmission in mobile ad-hoc networks," *IEEE Wireless Communications*, Jan 2006.

[90] J. Wagner, J. Chakareski, and P. Frossard, "Streaming of scalable video from multiple servers using rateless codes," *IEEE International Conference on Multimedia and Expo (ICME)*, Jan 2006.

[91] P. Frossard, "Fec performance in multimedia streaming," *IEEE Communications Letters*, Jan 2001.

Curriculum Vitae

Personal Information

Name:	Jean-Paul Wagner
Address:	Ecole Polytechnique Fédérale de Lausanne EPFL - STI - IEL - LTS4 Bâtiment ELD, ELD 239 Station 11 CH - 1015 Lausanne Switzerland
Phone:	+41 - (0)21 - 693 47 09
Fax:	+41 - (0)21 - 693 76 00
E-Mail:	Jean-Paul.Wagner@epfl.ch
Date/Place of Birth:	April 21st 1979, Luxembourg
Nationality:	Luxembourgish
Marital Status:	Engaged

Professional Experience

2004 - 2008	Research Assistant at the Signal Processing Laboratory, Ecole Polytechnique Fédérale de Lausanne (EPFL).
Jul 2007	Invited scientist at the Real-Time and Control Laboratories, General Electrics (GE) Global Research Center, Shanghai, China.
May - Oct 2002	Research Intern, Sun Microsystems Laboratories, Mountain View, CA.
Feb - Jul 2001	Assistant, Audio-Visual Communications Laboratory, Ecole Polytechnique Fédérale de Lausanne (EPFL).
Aug - Sep 2000	Trainee at Société Européenne des Satellites, SES-Astra, Luxembourg.

Education

2004 - 2008	Ph.D. at the Signal Processing Laboratory - LTS4, Ecole Polytechnique Fédérale de Lausanne (EPFL).
1998 - 2004	M.Sc. in Communication Systems, Ecole Polytechnique Fédérale de Lausanne (EPFL).
1992 - 1998	High School, Lycée de Garçons de Luxembourg.

Linguistic Skills

French	Fluent, oral and written.
English	Fluent, oral and written.
German	Fluent, oral and written.
Luxembourgish	Mother Tongue.

Technical Activities

Local Chair of the 16th International Packet Video Workshop,
Lausanne, Switzerland, Nov. 11-13th, 2007.

Publications

Patents

- R. B. Smith, K. McIntyre, A. Goyal, J.-P. Wagner
  Method and Apparatus for communicating with a computing device that is physically tagged
  United States Patent 20040205191, October 2004,
  Sun Microsystems Laboratories, CTO.

Journal Papers

- J.-P. Wagner and P. Frossard
  Distributed Media-Friendly Rate Allocation,
  submitted to IEEE Transactions on Multimedia, March 2008.

- J.-P. Wagner and P. Frossard
  Joint Playback Delay and Buffer Optimization in Scalable Video Streaming,
  IEEE Transactions on Circuits and Systems for Video Technology, under review.

- D. Jurca, J. Chakareski, J.-P. Wagner and P. Frossard
  Enabling Adaptive Video Streaming in P2P systems,
  IEEE Communications Magazine, June 2007.

Conference Papers

- J.-P. Wagner and P. Frossard
  Adaptive and Robust Media Streaming over Multiple Channels with Bursty Losses,
  Invited Paper, in Proc. of EUSIPCO 2007, September 2007, Poznan, Poland.

- J.-P. Wagner, J. Chakareski and P. Frossard
  Streaming of Scalable Video from Multiple Sources using Rateless Codes,
  in Proc. of IEEE ICME 2006, July 2006, Toronto, Canada.

- J.-P. Wagner and P. Frossard
  Playback Delay and Buffering Optimization in Scalable Video Broadcasting,
  Invited Paper, in Proc. of the First International Conference on Multimedia Services Access Networks (MSAN), Orlando, FL, June 2005.

- J.-P. Wagner and P. Frossard
  Playback Delay Optimization in Scalable Video Streaming,
  in Proc. of IEEE ICME 2005, Amsterdam, July 2005.

- J.-P. Wagner and R. Cristescu
  Power Control for Target Tracking in Sensor Networks,
  in Proc. of the 39th Conference on Information Sciences and Systems (CISS 2005), Baltimore, MD, March 2005.

- R. Puri, S. Servetto, J.-P. Wagner, P. Scholtes and M. Vetterli
  Video Multicast in (Large) Local Area Networks,
  in Proc. of IEEE Infocom '02, New York City, NY, June 2002.

Technical Reports

- J.-P. Wagner and P. Frossard
  Distributed Media-Friendly Rate Allocation,
  EPFL, Signal Processing Laboratory, Technical Report LTS-2008-003, March 2008.

- J.-P. Wagner and P. Frossard
  Joint Playback Delay and Buffer Optimization in Scalable Video Streaming,
  EPFL, Signal Processing Institute, Technical Report, TR-ITS-2007.13, December 2007.