EFFICIENT AND SECURE DELIVERY OF SCALABLE VIDEO STREAMS

by

Kianoosh Mokhtarian
B.Sc., Sharif University of Technology, Tehran, Iran, 2007

A Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Fall 2009

APPROVAL

Name:	Kianoosh Mokhtarian
Degree:	Master of Science
Title of Thesis:	Efficient and Secure Delivery of Scalable Video Streams

Examining Committee: Dr. Kay Wiese
Chair

Dr. Mohamed Hefeeda, Senior Supervisor

Dr. Robert Cameron, Supervisor

Dr. Ivan Bajic, SFU Examiner

Date Approved: 24 AUG 2009

Abstract

In today's multimedia systems, clients are getting quite heterogeneous in terms of connection bandwidth, processing power, and display resolution. Scalable video coding techniques can support this heterogeneity by enabling us to encode a video stream once and extract/decode it in several ways according to receiver's capabilities. We study secure and efficient delivery of scalable streams. First, we propose an authentication scheme for end-to-end secure delivery of scalable video streams, which supports their full flexibility: it enables verification of any possible substream extracted from the original stream. Then, we consider streaming of scalable videos over peer-to-peer networks, and study efficient management of seed servers resources in these networks. We prove the hardness of optimally allocating these servers and propose two approximation algorithms to solve it. We evaluate both our security and resource allocation solutions and show their efficiency analytically and through simulations.

Keywords: video streaming; multimedia security; peer-to-peer streaming; scalable video coding; H.264/SVC

Acknowledgments

I am deeply beholden to Dr. Mohamed Hefeeda, my senior supervisor, for continuously supporting me over these two years with his patience and knowledge. Anytime I needed help, he made his time available for me and helped me with valuable advice. It was his guidance and encouragement that taught me how to do research, and without his support, completion of this thesis would not have been possible for me.

I would like to express my gratitude to Dr. Rob Cameron, my supervisor, and Dr. Ivan Bajic, my thesis examiner, for being on my committee and reviewing this thesis. I also would like to thank Dr. Kay Wiese for taking the time to chair my thesis defense.

Last but certainly not least, I owe my deepest gratitude to my parents for their love, encouragement, and endless support. They taught me the value of education and thoughtfulness, raised me to always strive for higher levels, and gave me the confidence to make it through life. I will never forget what I owe them, and my eternal gratitude to them cannot be expressed by words.

Contents

Approval ii

Abstract iii

Acknowledgments iv

Contents v

List of Tables viii

List of Figures ix

1 Introduction 1

1.1 Introduction and Background 1

1.2 Problem Statement and Contributions 3

1.2.1 Problem I: Authentication of Scalable Video Streams 3

1.2.2 Problem II: Efficient Allocation of Seed Servers in Peer-to-Peer Streaming Systems 5

1.2.3 Thesis Contributions 5

1.3 Thesis Organization 7

2 Analysis of Media Stream Authentication Schemes 8

2.1 Introduction 8

2.2 Background and Related Work 10

2.2.1 Common Cryptographic Techniques Used for Authentication 10

2.2.2 Related Work 11

2.3 Notations and Performance Metrics 11

2.4 Authentication Schemes for Nonscalable Streams 13
2.4.1 Hash Chaining 14
2.4.2 Augmented Hash Chaining 15
2.4.3 Butterfly Hash Chaining 16
2.4.4 Tree chaining 16
2.4.5 SAIDA and eSAIDA 17
2.4.6 cSAIDA 19
2.4.7 TFDP 19
2.4.8 Other Authentication Schemes 20
2.5 Evaluation of Authentication Schemes for Nonscalable Streams 22
2.5.1 Numerical Analysis 23
2.5.2 Simulation 26
2.5.3 Summary and Discussion 31
2.5.4 iTFDP 31
2.6 Authentication of Scalable Multimedia Streams 33
2.6.1 2-D Hash Chaining 34
2.6.2 2-D Tree Chaining 35
2.6.3 Evaluation of Authentication Schemes for Scalable Streams 36
2.6.4 Limitations of Current Schemes for Modern Scalable Streams 38
2.7 Summary 39

3 The Proposed SVC Stream Authentication Scheme 41
3.1 Introduction 41
3.2 Related Work 42
3.3 Background 43
3.3.1 Overview of H.264/SVC 43
3.3.2 Importance of Protecting All Layers 45
3.4 The Proposed Scheme 46
3.4.1 The Proposed Authentication Scheme 46
3.4.2 Authentication of Quality Layers 49
3.4.3 Authentication of a Video Frame 50
3.4.4 Authentication of a GoP 51
3.4.5 Authentication of a Sequence of GoPs 52

3.5 Analysis and Security of the Proposed Scheme 53
3.5.1 Security Analysis 53
3.5.2 Complexity Analysis 56
3.6 Reducing the Overhead 56
3.6.1 The Cost Function 60
3.6.2 The Impact of Packet Losses on the Overhead Reduction Algorithm . 62
3.7 Performance Evaluation 63
3.7.1 Simulation Setup 63
3.7.2 Simulation Results 65
3.8 The svcAuth Library 69
3.9 Summary 71

4 Seed Server Allocation in P2P Streaming Systems 73
4.1 Introduction 73
4.2 Related Work 74
4.3 System Model and Problem Statement 75
4.3.1 System Overview 76
4.3.2 Problem Statement and Hardness 77
4.4 Problem Solution 79
4.4.1 SRA_DP: Seed Resource Allocation using Dynamic Programming . . . 79
4.4.2 SRA_GREEDY: Seed Resource Allocation using a Greedy Algorithm . 83
4.4.3 The Utility Function 86
4.5 Evaluation 88
4.5.1 Simulation Setup 88
4.5.2 Simulation Results 90
4.6 Summary 94

5 Conclusions and Future Work 95
5.1 Conclusions 95
5.2 Future Work 97

Bibliography 101

List of Tables

2.1 Parameters used in our analysis and their values. 13
2.2 Summary of the analysis of authentication schemes for nonscalable streams. . 23
2.3 Counteracting packet losses by different authentication schemes when the overhead is fixed to 40 bytes per packet (42 bytes for the Augmented chain). 28

4.1 List of notations used in the seed server allocation problem. 77

List of Figures

1.1 Heterogeneous receivers participating in the same streaming session. 3

2.1 Stream authentication using hash chaining. 14

2.2 Stream authentication using SAIDA. 17

2.3 Computation cost (time to verify a block of packets) versus block size. 25

2.4 Communication overhead (bytes per packets) versus block size. 26

2.5 Verification rate versus packet loss ratio. 27

2.6 Verification rate versus communication overhead. 30

2.7 Comparison between the proposed scheme (iTFDP) and cSAIDA. 32

2.8 Verification rate of iTFDP versus cSAIDA for different per-packet overheads. 33

2.9 Authenticating scalable streams by hash chaining. 34

2.10 A Merkle hash tree built over 8 layers. As an example, for authenticating layer 1 only, the hash of layer 2 as well as b and v are needed. 35

2.11 Communication overhead versus bitrate. 37

2.12 A three-dimensionally scalable video and a possible substream of that. 38

3.1 Hierarchical prediction structure of H.264/SVC temporal scalability. Arrows represent prediction. Numbers listed in the bottom row show the displaying order, while numbers inside frames show the coding order. 44

3.2 Results of tampering with one MGS packet. 45

3.3 The proposed authentication scheme. 48

3.4 Authenticating a video frame. 49

3.5 Authenticating frames of a GoP. 52

3.6 Average bitrate extracted from the video versus the size of truncation units. . 64

3.7 Computation cost, delay, and loss resilience of the proposed authentication scheme. 65
3.8 Fraction of received packets that can be verified. 66
3.9 User bandwidth distribution and the number of truncation units per frame at different temporal levels. 67
3.10 The efficiency of the overhead reduction algorithm. 68
3.11 Deciding the best number of groups. 69
3.12 Deployment of svcAuth Authentication and Verification Module. 70
3.13 The svcAuth Authentication Module. 71

4.1 A request from a peer who is demanding the first five layers in total, and is receiving the first two from other peers and the next two from a seed server. . 76
4.2 Tradeoff between approximation factor and computational complexity. 83
4.3 The SRA_GREEDY algorithm for the seed server allocation problem. 84
4.4 Max-min fairness using Eq. (4.11) for serving layers to different demands with limited serving capacity. 87
4.5 Near-optimality of the proposed algorithms. 90
4.6 Satisfaction experienced by at least 90% of peers. 91
4.7 Video quality for peers with different download bandwidths. 92
4.8 Incentive provided for peers to share upload bandwidth. 93

Chapter 1

Introduction

In this chapter, we provide a brief background about secure delivery of multimedia streams, scalable video coding, and peer-to-peer (P2P) streaming systems. Then, we introduce the problems we address in this thesis and summarize our contributions. The organization of this thesis is given at the end of this chapter.

1.1 Introduction and Background

The demand for multimedia services has been steadily increasing in the past few years and is expected to grow even faster in near future, as confirmed by many market research reports [1–3]. With such a strong demand, multimedia services are becoming prevalent and many people rely on them in different aspects of their daily lives, including work, education, and entertainment. In these services, however, multimedia content is often distributed over open and generally insecure networks, such as the Internet, which makes the content vulnerable to malicious manipulations and alterations. Accordingly, secure delivery of multimedia streams has become an important and critical concern in today's multimedia systems. By secure delivery we mean ensuring the authenticity of the stream such that any tampering with the content by an attacker can be detected by the receiver. Attackers may tamper with the multimedia data by removing, inserting, or modifying portions of the data. These attacks may be performed for commercial, political, or even personal purposes. Securing the delivery of multimedia streams is the first problem we study in this thesis.

On the other hand, distribution of multimedia streams in large scales over the Internet has been a topic of interest to academia and industry for several years. Due to the lack of IP

Multicast deployment in today's Internet and the high cost of purely server-based solutions, the use of peer-to-peer (P2P) technology for Internet streaming has attracted significant attention in recent years [4]. P2P streaming systems do not require any particular support from the Internet infrastructure, and are easy to deploy. In addition, as more peers join the system and demand the data, they also increase the streaming capacity by uploading the streams they receive. The second problem we study in this thesis is efficient management of resources for serving scalable video streams in P2P streaming systems.

In today's multimedia streaming systems, clients are getting more and more heterogeneous in terms of connection bandwidth, processing power, and screen resolution. For example, a user with a limited-capability cellular phone with a small screen and wireless connection, and one with a high-end powerful workstation behind cable connection can both be requesting the same video stream. An example of this is illustrated in Figure 1.1. To support a wider range of such receivers, it is preferred to encode and distribute a lower-bitrate video stream, but this will provide a low quality for everyone. By encoding a higher-bitrate stream, on the other hand, we cannot support many of the receivers. This problem may be solved by encoding and distributing multiple versions of the video, which is called *simulcasting*. However, a video has to be encoded many times for different combinations of decoding capabilities, connection bandwidths, and viewing resolutions. Moreover, switching among versions is not easy, because (i) for every switching, a client has to wait, possibly for a few seconds, for the next Intra-coded frame (I-frame) of the new version, and (ii) the streams of different versions could be asynchronous [5]. As an alternative, Multiple Description Coding (MDC) can encode a video into multiple descriptions, where the quality of the video will be proportional to the number of descriptions received. However, MDC techniques are well known for having considerable bitrate overhead and being computationally complex [5].

In contrast, a *scalable* video stream has the advantage that it can be encoded once, and a variety of substreams can be extracted and decoded from it according to receiver's capabilities. In other words, a wide range of heterogenous clients can benefit from the same stream. Scalable coding has a lower overhead and is simpler than MDC coding [5]. Moreover, recent scalable video coding techniques have further improved this coding efficiency and significantly outperformed previous scalable videos [6]. For example, the Scalable Video Coding (SVC) extension of the state-of-the-art H.264/AVC video coding technique [7], known as the H.264/SVC [8] standard, can generate highly flexible video streams at a bitrate close to the bitrate of the corresponding nonscalable video stream. These scalable streams support

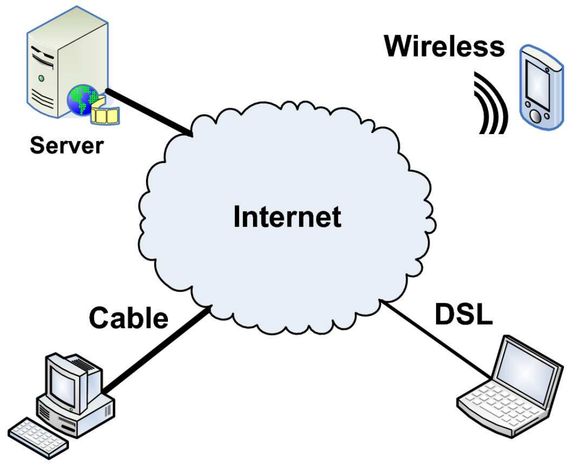


Figure 1.1: Heterogeneous receivers participating in the same streaming session.

three scalability types at the same time: temporal, spatial, and quality, which provide different frame rates, different spatial resolutions, and different visual qualities, respectively. These features have attracted significant attention and these streams are being increasingly adopted in many applications, e.g., [9–12].

1.2 Problem Statement and Contributions

Our goal is to adopt and take best advantage of scalable video streams in today's multimedia streaming systems. This thesis studies two problems related to this goal: securely delivering scalable video stream and efficiently allocating resource for serving scalable streams in P2P systems.

1.2.1 Problem I: Authentication of Scalable Video Streams

Although the problem of data authentication is well digested and practical solutions exist, secure delivery of multimedia streams is challenging due to several reasons. First, the authentication mechanism, which can be computationally expensive, has to keep up with the online nature of streams. Second, multimedia content is often distributed over unreliable channels, where packet losses are not uncommon. The authentication scheme needs to function properly even in presence of these losses. Third, the information added to the streams by the authentication scheme should be minimized in order to avoid increasing the already-high storage and network bandwidth requirements for multimedia content. Finally and most importantly, the authentication scheme must support the flexibility of scalable streams and successfully verify *any* substream extracted from the original stream. More precisely, the problem of authenticating scalable video streams can be stated as follows.

Problem 1 *Given a video stream encoded in a scalable manner that enables extraction of substreams along three types of scalability, temporal, spatial, and quality, design an efficient authentication scheme that can sign the video stream once and ensure successful verification of all possible substreams.*

Due to its importance, authentication of multimedia streams, nonscalable and scalable, has been extensively studied by academia and industry. Several authentication schemes have been proposed to address this problem in different settings. First, merits and shortcomings of these schemes over each other are not clear such that we can choose the most suitable scheme for a given multimedia streaming application. Moreover, authentication of recent scalable video streams is not supported by these schemes, even though authentication of traditional scalable video streams has been studied by several previous works, e.g., [13–15]. In a traditional scalable stream, the video is encoded as a base layer and a number of enhancement layers that progressively improve the video in terms of spatial resolution or visual quality. Recent scalable video streams, in contrast, can provide different types of scalability at the same time and with higher flexibility. They are more general than the traditional, and much simpler, linear layered videos in which scalability is typically provided along one dimension and the layers in that dimension are cumulative. Recent scalable streams, most notably H.264/SVC, can be adapted along different scalability dimensions, and enable many possible combinations of layers. In addition, they employ new useful coding tools designed for scalable coding, such as hierarchical prediction of frames and medium-grained scalability. Consequently, previous authentication schemes, which are designed for simple layered videos, are not applicable to these scalable streams, as we discuss in more details in Section 2.6. To the best of our knowledge, there are no authentication schemes in the literature that can efficiently support the full flexibility of recent scalable streams.

1.2.2 Problem II: Efficient Allocation of Seed Servers in Peer-to-Peer Streaming Systems

In P2P streaming systems, the upload bandwidths of peers are often far less than their demanded download rates. For example, an average-to-good quality video stream requires about 1–2 Mbps bandwidth, whereas the average upload capacity of home users with DSL and cable connections is often less than a few hundred kbps. To make up for this asymmetry, a number of seed servers need to be deployed in the P2P network in order to deliver high-quality video streams to users. These servers have finite capacity and are often loaded with a volume of requests larger than their serving capacity. Accordingly, arbitrary allocation of these servers for serving peers will result in poor management of resources and inefficient utilization of data, especially when these servers serve scalable video streams. We study efficient allocation of seeding resources in P2P streaming systems with scalable videos. This problem can be stated as follows.

Problem 2 *Design an efficient algorithm that, given a total seeding capacity, decides how to allocate this capacity for serving peers' request for different substreams such that a network-wide utility function is maximized, e.g., the average video quality delivered to peers.*

P2P streaming with scalable videos has been studied by several previous works, e.g., [16–18]. Most of these works, however, do not consider the functionalities of seed servers. Allocation of seed servers for P2P streaming with nonscalable videos has also been studied in the literature [19]. However, the case for scalable video streams is more challenging as various substreams need to be considered and served to receivers. That is, the finite seeding capacity needs to be optimally allocated to requesting peers and requested layers such that a higher-quality video is delivered to all peers. We review previous works in details in Section 4.2. To the best of our knowledge, the problem of seeding scalable videos with limited seeding resources is not studied by any of them.

1.2.3 Thesis Contributions

The contributions of this thesis can be summarized as follows:

- We conduct a comprehensive analysis and *quantitative* comparison of the main schemes proposed in the literature to authenticate multimedia streams [20, 21]. Our analytical and simulation based comparisons along multiple performance metrics reveal the merits and shortcomings of each scheme. Moreover, our analysis provides guidelines on choosing the most suitable scheme for a given multimedia streaming application, and offers insights for further research on the stream authentication problem. For instance, our analysis has led us to propose a new authentication scheme for nonscalable streams, which combines the advantages of two previous schemes for on-demand streaming applications. Our evaluations show that the proposed authentication scheme outperforms others in the literature for on-demand streaming applications. We also show that current schemes for authenticating scalable video streams fail to support the full flexibility of recent scalable streams, and empirically demonstrate the importance of this problem by performing sample tampering attacks.

- We propose a new authentication scheme that supports the full flexibility of recent, three-dimensionally scalable video streams [22]. We analytically show that the proposed scheme guarantees correct verification of any valid substream extracted from the original stream. The algorithm is designed for *end-to-end* authentication of streams. That is, a third-party content delivery network in charge of delivering (and possibly adapting) the streams does not have to be aware of and compatible with the authentication scheme, which is an important advantage. We also propose an additional algorithm for minimizing the communication overhead imposed by the authentication scheme. The algorithm can be of interest in its own right, since it can be used to minimize the communication overhead of other scalable stream authentication schemes as well. Our simulation study with real video traces confirms that the proposed authentication scheme is robust against packet losses, incurs low computational cost and buffer requirement, has a short delay, and adds small communication overhead particularly after using the overhead reduction algorithm.

- We implement our authentication scheme for H.264/SVC streams in a prototype called *svcAuth*, which is available as an open source library and can be employed by any multimedia streaming application as a transparent software add-on, without requiring changes to the encoders/decoders.

- We address the problem of optimally allocating the resources of seed servers for streaming scalable videos over P2P networks [23]. We formulate this problem and show that it is NP-complete. We then propose two approximation algorithms for the problem, which complement each other. We analytically show that they produce near-optimal results while being computationally efficient and able to run in real-time. The results of evaluating the algorithms in a simulated P2P streaming system confirm the efficiency and near-optimality of the proposed algorithms, and show that higher-quality video streams are delivered to peers if our algorithms are employed for allocating seed servers.

1.3 Thesis Organization

We conduct an analysis and comparison of previous multimedia stream authentication schemes in Chapter 2. In Chapter 3, we present, analyze, and evaluate our proposed scheme for authentication of modern scalable video streams, and we give an overview of its implementation in the svcAuth library. In Chapter 4, the seed server allocation problem in P2P streaming systems, and our proposed algorithms, are presented. Finally, we conclude this thesis and highlight future research directions in Chapter 5.

Chapter 2

Analysis of Media Stream Authentication Schemes

In this chapter, we present a comprehensive analysis and comparison among the main schemes proposed in the literature to authenticate multimedia streams. We first see an overview of the analysis and the need for that, followed by the details of the analysis. According to our analysis and comparison, we draw several conclusions on the advantages and shortcomings of each authentication scheme and present recommendations for choosing the appropriate schemes for various streaming applications.

2.1 Introduction

The large prevalence of multimedia systems in recent years makes the security of multimedia communications an important and critical issue. Accordingly, the problem of multimedia stream authentication has been studied by many previous works, and several schemes have been proposed to address this problem in different settings [21, 24]. However, no rigorous analysis and *quantitative* comparison of the different schemes has been done in the literature, to the best of our knowledge. Detailed analysis of various authentication schemes is needed in order to discover the merits and shortcomings of each scheme. Moreover, side-by-side comparisons of authentication schemes along multiple performance metrics provide guidelines on choosing the most suitable scheme for a given multimedia streaming application, and offer insights for further research on the stream authentication problem.

We conduct a comprehensive analysis and quantitative comparison among different multimedia stream authentication schemes proposed in the literature. We consider authentication schemes for both nonscalable and scalable streams. We first analyze the main authentication schemes for nonscalable streams. To conduct this analysis, we define five important performance metrics, which are computation cost, communication overhead, receiver buffer size, delay, and tolerance to packet losses. Then, we derive analytic formulas for these metrics for all considered authentication schemes, and numerically analyze these formulas to explore the performance of the schemes for a wide range of parameters. In addition, we implement all authentication schemes in a simulator to study and compare their performance in different environments. The parameter values for the simulator are carefully chosen to mimic realistic settings. For example, we analyze the authentication schemes under two common models for packet losses: bursty and random. The bursty loss model is typical in wired networks where a sequence of packets may get dropped because of a buffer overflow in one of the routers on the network path from sender to receiver. Whereas the random loss model is usually used to capture bit errors in wireless environments. We highlight the advantages and disadvantages of each scheme, and draw several conclusions on their behavior. Furthermore, according to our analysis of the schemes, we propose a new authentication scheme for nonscalable streams, which combines the best features of two of the best-performing previous schemes for on-demand streaming applications. According to our evaluation study we believe that the proposed scheme is the most suitable one for on-demand streaming.

In the second part of this chapter, we extend our analysis to authentication schemes for scalable streams. We pay careful attention to the flexibility of scalable streams and analyze its impacts on authentication schemes. Based on our analysis, we present a number of recommendations for choosing the appropriate schemes for various streaming applications. Then, we analyze the application of current scalable stream authentication schemes to recent scalable video streams, e.g., H.264/SVC, and show that current schemes fail to support the full flexibility of these streams.

This chapter continues by reviewing the common cryptographic techniques used for authentication, and summarizing previous works related to this analysis in Section 2.2. Then, we define in Section 2.3 the performance metrics and notations used in our analysis. In Section 2.4, we present and analyze authentication schemes for nonscalable streams. For each scheme, we provide a brief overview of the scheme followed by the analysis of the different performance metrics. In Section 2.5, we conduct numerical analysis of the equations derived in Section 2.4. We also present our simulation analysis and summarize our findings in this section. We analyze authentication schemes for scalable multimedia streams and discuss their shortcomings for supporting modern scalable streams in Section 2.6. A brief summary of the chapter is given in Section 2.7.

2.2 Background and Related Work

In this section, we briefly review basic cryptographic techniques used in authentication schemes, and then summarize previous works related to our analysis.

2.2.1 Common Cryptographic Techniques Used for Authentication

The goal of an authentication scheme is to enable a receiver to make sure that the content it has received is originally generated by the desired content provider, and it is not tampered with by any other entity. To provide this service, a number of cryptographic functions and techniques are usually employed in authentication schemes [25]. A commonly used type of these functions are one-way hash functions. They take as input a message x of arbitrary length and produce a fixed-length output, $y = h(x)$, to which we refer as the hash value or the message digest. Given x, computation of $h(x)$ is easy, whereas if y is given, finding x or any other message whose digest equals y is infeasible. Moreover, finding two different messages x_1 and x_2 such that $h(x_1) = h(x_2)$ is infeasible. Therefore, if the authenticity of $h(x)$ is successfully verified, we know that message x is also authentic, because no attacker can change or replace x while preserving its hash value $h(x)$. SHA-1 [26] and MD5 [27] are two commonly used hash functions.

Digital signatures provide authentication as well as non-repudiation. That is, the receiver can make sure that the message is generated by the desired sender and is not modified by anybody else, and also the sender cannot repudiate or refuse the validity of his signature. RSA (by Rivest, Shamir, and Adleman) [28] and DSS (Digital Signature Standard) [29] are two commonly used digital signature algorithms. Since these algorithms are based on asymmetric cryptography, they are computationally expensive. To save in computational cost, the conventional method for signing a message of arbitrary length is to compute the digest of the message and sign the digest, i.e., $sign(h(x))$, because the digest has a short fixed length and its authenticity is enough for assuring the authenticity of the original message.

Message Authentication Codes (MAC) are an alternative way for ensuring the authenticity of a message using a secret key K. A MAC is a short digest calculated over an arbitrary-length message x using the secret key K as input, i.e., $y = MAC(x, K)$. Since they are based on symmetric cryptography, MAC codes are very inexpensive to compute. However, because of being based on a shared secret key, they cannot make a commitment between a message and one particular signer. That is, if one is able to verify a MAC, i.e., he has the key, then he is able to alter the message and compute a new forged MAC. Thus, MACs are more suitable for single-sender single-receiver communications where the two entities have already managed to securely share a secret key.

2.2.2 Related Work

A survey on authentication schemes for multicasting multimedia streams is given in [24]. The authors classify authentication schemes according to the core techniques underlying them, e.g., symmetric or asymmetric cryptography, partially sharing secrets with receivers, relying on time synchronization, replicating hash values, and signature amortization. The authors also provide a qualitative comparison among the schemes. However, unlike our work, the work in [24] does not provide a quantitative analysis and detailed comparison of the authentication schemes in realistic environments. In addition, the survey in [24] is relatively old and does not cover a number of recent authentication schemes that we consider in this chapter. Furthermore, the work in [24] does not discuss or analyze authentication schemes for multimedia streams encoded in scalable manner, which have become popular recently because of their ability to support a wide range of heterogeneous clients [8–10]. Previous authentication techniques for scalable JPEG 2000 images [30–32] are surveyed in [33]. However, we are not aware of previous works that analyze authentication of scalable video streams.

2.3 Notations and Performance Metrics

To rigorously evaluate various multimedia authentication schemes, we define five performance metrics that cover all angles of the authentication problem, and we analyze these metrics in different environments. The performance metrics are as follows.

- *Computation Cost.* It is the CPU time needed to verify the authenticity by the receiver. Evaluating the computation cost is important especially if the receiver has a limited processing capacity, e.g., a PDA or cell phone. Note that in streaming applications, the verification process of an incoming multimedia stream is invoked periodically and in real-time. Thus, if its computation cost is high, some receivers may not be able to support it.

- *Communication Overhead.* It is the number of additional bytes that the authentication scheme needs to transfer over the communication channel to the receiver in order to enable it to verify the authenticity of the received multimedia stream.

- *Tolerance to Packet Losses.* Multimedia streams are typically transmitted over the Internet or lossy wireless channels, where some packets may get lost. Due to the dependency that the authentication scheme imposes among packets, packet losses may affect verifiability of some packets that are successfully received. Thus, the robustness of the authentication scheme to packet losses is important to analyze for different packet loss ratios. We quantify this robustness as the percentage of the received packets that can be verified in presence of packet losses, and we call it the verification rate.

- *Receiver Buffer Size.* Some authentication schemes require the receiver to buffer a certain amount of data before it can start verifying the stream. The required buffer size specifies the minimum memory requirements, which is especially important for limited-capability receiver devices. Besides, the required receiver buffer determines the amount of delay at receiver side, which is included in the Delay metric below.

- *Delay Imposed by the Authentication Process.* Since most of the schemes designate one digital signature for a block of packets, they require the sender/receiver (or both) to wait for generation/reception of a certain amount of data before being able to transmit/verify it. This delay, which is the sum of sender side and receiver side delays, specifies whether or not the authentication information can be produced or verified online. For example, a delay beyond a few seconds is not suitable for live streaming. Note that we consider the delay imposed by the authentication process only; delays caused by transmission through the networks or by the media encoding/decoding process are not accounted.

Table 2.1: Parameters used in our analysis and their values.

Parameter	Value	Description
α	30 pkt/sec	Packet rate.
l	1400 bytes	Packet size.
n	128 pkts	Block size. It is 128 packets as long as it is not one of the variable parameters.
N	1000 to 1 M	Number of blocks transmitted.
ρ	0 to 0.5	Packet loss ratio.
d	8 pkts	Expected burst length.
n_{enough}	$0.8n$	Number of packets of a block that suffice for verification.
t_{sig}	500 ms	Time to verify a digital signature (1024-bit RSA with public exponent = 65537).
t_{hash}	0.1 ms	Time to compute a hash over a 512-bit (64-byte) block.
s_{sig}	128	Signature size in bytes (1024-bit RSA).
s_{hash}	20	Hash size in bytes (SHA-1).
n_{rows}	32	Number of rows in Butterfly graph (see Section 2.4.3).
s	0, 0.25, 0.5	An input to eSAIDA (see Section 2.4.5).
n_{sig}	Aug chain: 8, Btfly graph: searched	Number of signature replications in the block. It is set to $\frac{1}{16}n$ (i.e., 8) for Augmented Chain. For Butterfly Graph, it is obtained by a local search for best verification rate.
p, a	Searched	Inputs to Augmented chain (see Section 2.4.2).

The above metrics are analyzed under different scenarios. For example, we consider a wide range of packet loss rates and using two common loss models: bursty and random. Several other parameters are used in the analysis, such as packet rate α, packet size l, and block size n. For quick reference, we list all parameters used in this chapter and their notations in Table 2.1. We also mention the range of values used for each parameter. In the simulation section, we discuss why we use these values.

2.4 Authentication Schemes for Nonscalable Streams

Several schemes have been proposed for authenticating nonscalable multimedia streams. A multimedia stream is nonscalable if it is encoded as a single layer and only the complete layer is decodable [34]. We study and analyze the most important schemes in the literature. A naive solution for authenticating such stream may be to sign every packet. This clearly does not work in practice due to its high computational cost. Accordingly, to amortize this cost, authentication schemes often divide a stream into blocks of n packets, and designate

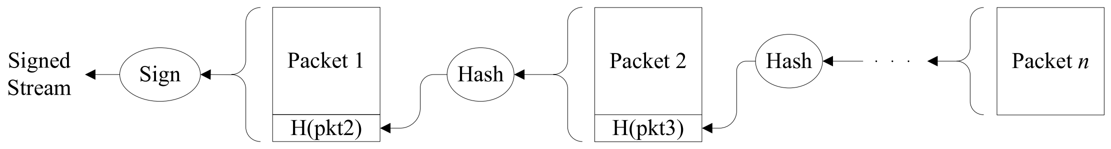


Figure 2.1: Stream authentication using hash chaining.

one digital signature for each block.[1] In each of the following subsections (Sections 2.4.1—2.4.7), we briefly describe the main idea of an authentication scheme and we analyze it using the performance metrics defined in Section 2.3. We also mention the authentication schemes that are not analyzed in this chapter and why we do not consider them in Section 2.4.8. We use the parameters listed in Table 2.1 in the analyses.

2.4.1 Hash Chaining

Hash chaining [35] is one of the simplest techniques to authenticate multimedia streams. Figure 2.1 illustrates its basic idea. Packets of the stream are divided into blocks, each of size n packets. Then, the hash of each packet is attached to its previous packet, and the first packet of each block is digitally signed. Due to the one-way property of the hash function, the signature authenticates the whole block.

Analysis of hash chaining is straightforward. For a block of n packets, hash chaining computes n hash values and verifies one digital signature. Therefore, the computation cost to verify a block is $t_{sig} + n\lceil l/64 \rceil t_{hash}$ seconds. The communication overhead is $s_{sig}/n + s_{hash}$ bytes per packet. Hash chaining does not tolerate any packet losses. There is no receiver buffer requirement for this scheme as packets can be verified as they arrive after receiving the first packet with the signature. The sender, however, needs to wait for n packets to be generated, because hash chaining starts at the last packet in the block. Thus, with a packet generation rate of α, the total delay is n/α.

To enable the hash chaining scheme to tolerate packet losses, the hash value of a packet is replicated and attached to multiple packets. According to the way hashes are replicated in packets, a block of packets can be modeled as a Directed Acyclic Graph (DAG), whose nodes represent data packets and each directed edge from node A to node B indicates that

[1]Note that we only consider authentication schemes that rely on digital signatures for assuring authenticity. That is, we do not consider schemes that only rely on Message Authentication Codes (MAC), which requires all parties of the streaming setting to be fully trusted.

the hash of packet A is attached to packet B, i.e., if packet B is verified then packet A can also be verified. In this DAG, a packet is verifiable if there is a path from its corresponding node to the signature node. When loss occurs among packets of a block, some nodes of the DAG and their associated edges are removed, which may threaten the verifiability of some of the received packets. The simple hash chaining described above can be viewed as a linear DAG with n nodes and $n-1$ edges, where the first node carries the signature. The following two subsections present two authentication methods that improve the robustness of a linear DAG to packet losses.

2.4.2 Augmented Hash Chaining

In the augmented hash chaining scheme [36], the authentication DAG is constructed as follows. First, the hash of packet p_i is attached to packets p_{i+1} and p_{i+a}, where a is an integer parameter that affects resistance against bursty losses as well as receiver delay and buffer. The last packet is designated as the signature packet. Then, $p-1$ additional packets (p is an input to the algorithm) as well as their relevant edges are inserted between each two packets of this chain to make it an *augmented chain*. Two methods are proposed for this insertion, which have equal resistance to bursty losses. The first method attaches the hash of each new packet to the packet preceding it and to the packet from the original chain succeeding it. Thus, the number of hashes carried by packets of the original chain grows linearly with p, while the average number of hashes per packet is two. The second method is more complex and follows a recursive structure to keep the degree of each node equal to two; we consider the second structure in our analyses. Since the delivery of the signature packet is vital, the scheme sends it multiple (n_{sig}) times within a block of packets. The packet loss tolerance of this technique depends on the loss model, i.e., it depends on the loss rate and loss pattern (random or bursty). We analyze this tolerance using simulation in Section 2.5.

Computations needed for verifying a block in augmented hash chaining take $t_{sig} + n\lceil l/64 \rceil t_{hash}$ seconds, and the communication overhead is $n_{sig}s_{sig}/n + 2s_{hash}$ bytes per packet. The receiver has to buffer a whole block, thus the receiver delay and buffer size are n/α seconds and n packets, respectively. Moreover, the sender needs to buffer p packets before transmission, which makes the sender delay p/α seconds. Since p is small compared to n, we consider the total delay to be n/α seconds.

2.4.3 Butterfly Hash Chaining

Zhang et al. [37] proposed to use Butterfly graphs to construct the authentication DAG. Assuming the number of packets of a block is $n = n_{rows}(\log_2 n_{rows} + 1)$, the nodes of the authentication DAG are arranged into $\log_2 n_{rows} + 1$ columns of the same length n_{rows}. Each node in a column is linked to two other nodes of the previous column, according to the column it belongs to. Nodes of the first column are all linked to the signature packet. These butterfly graphs, however, do not work for arbitrary number of packets, and make the size of the signature packet grow almost in proportion to the block size. To mitigate these limitations, the authors later extended their work to utilize a generalized Butterfly graph [38]. This graph is made more flexibly such that the number of rows n_{rows} is set independently of n and is taken as an input. Then, nodes are arranged into $\lceil n/n_{rows} \rceil$ columns, where the last column does not necessarily consist of n_{rows} nodes. The way the nodes are linked to each other is similar to the previous Butterfly graph. We consider the generalized version of Butterfly graph scheme in our analyses.

The computation cost of the butterfly authentication is the same as the augmented hash chaining: $t_{sig} + n\lceil l/64 \rceil t_{hash}$ seconds to verify a block of n packets. Denoting the number of rows in the butterfly graph by n_{rows}, the communication overhead of this scheme is equal to $n_{sig}(s_{sig} + n_{rows}s_{hash})/n + s_{hash}(2n - n_{rows})/n$ bytes per packet. According to losses, receivers need to buffer packets till a copy of the signature arrives. In the worst case they may need to have a buffer of up to n packets, though it is unlikely. Thus, for the total delay, we neglect the receiver delay when summing it with the sender of n/α seconds, which makes a total delay of n/α seconds. However, the receiver buffer required cannot be neglected even it fills infrequently. Similar to the augmented hash chaining, the loss tolerance of this schemes depends on the packet loss model, which we evaluate in the simulation section.

2.4.4 Tree chaining

Wong and Lam [39] proposed the use of Merkle hash trees [40] for stream authentication. In their scheme, one signature is designated for each block of packets. At the sender side, a balanced binary Merkle hash tree is built over packets of each block. Leaves of this tree are hashes of packets, and each interior node represents the digest of concatenation of its children. The root of this tree is then signed. Due to the collision-free property of the hash function, the whole set of leaf packets is authenticated if authenticity of the root of the tree

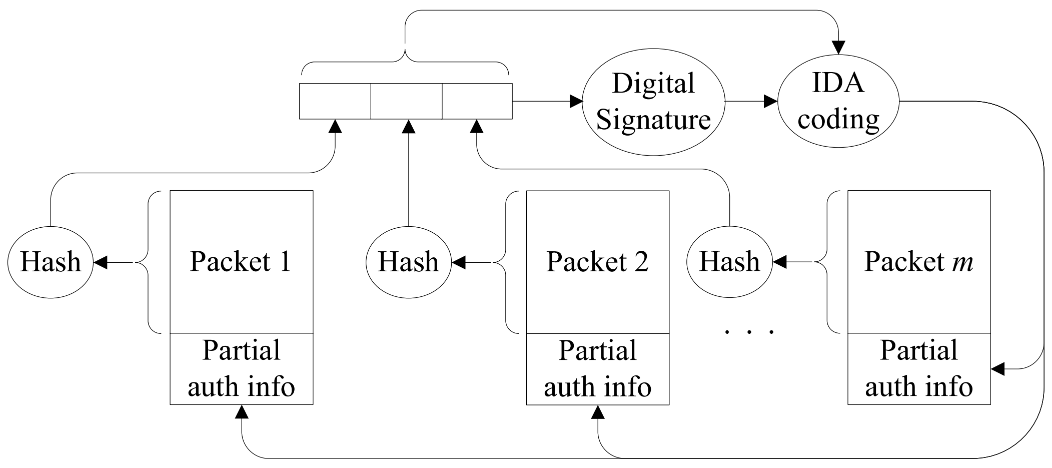


Figure 2.2: Stream authentication using SAIDA.

is successfully verified. Each packet is individually verifiable by traversing and partially reconstructing the tree from the bottom (the leaf node corresponding to the given packet) to top (the root) and verifying the root digest using the given signature. For this procedure, only siblings of the nodes on the path are needed. Therefore, in this scheme, each packet carries the block signature, its location in the block, and the set of siblings on the path from itself to the root. This makes each packet individually verifiable using the authentication information it carries.

The computations needed for verifying a block consist of one signature verification, $n\lceil l/64\rceil$ hash computations over packets, and $(\lceil n\log_2 n\rceil - n)\lceil 2s_{hash}/64\rceil$ hash computations interior to the tree, which in total takes $t_{sig} + t_{hash}(n\lceil l/64\rceil + (\lceil n\log_2 n\rceil - n)\lceil 2s_{hash}/64\rceil)$ seconds for a block. The communication overhead of this scheme is equal to $s_{sig} + \lceil \log_2 n\rceil s_{hash}$ bytes per packet. In tree chaining, there is no need to buffer any packet, thus a packet can be verified once it arrives. The total delay imposed by tree chaining, which consists of the sender delay only, is that of generating a block: n/α seconds. Moreover, loss resilience is always 100% given that a packet is either arrived or lost atomically.

2.4.5 SAIDA and eSAIDA

Park et al. [41] presented SAIDA (Signature Amortization using Information Dispersal Algorithm) for stream authentication. As shown in Fig. 2.2, SAIDA divides the stream into blocks of n packets. Then, it hashes each packet and concatenates the hash values. Let us denote the result of this concatenation by $H = h(p_1)||h(p_2)|| \cdots ||h(p_n)$, and the number of packets that are expected to be received out of a block of n packets by n_{enough}, i.e., $n_{enough} = (1-\rho)n$. H along with a signature on $h(H)$ is divided into n_{enough} ($n_{enough} \leq n$) pieces, IDA-coded[2] into n pieces and split over all the n packets of the block. Any n_{enough} pieces suffice to re-construct the hashes and the signature to verify authenticity of the entire block. Note that the signature itself is sufficient for authenticating the whole block if no loss occurs, but the concatenation of packet hashes is also IDA-coded and carried by packets so that the block is still verifiable if some packets are lost.

Computations needed by SAIDA to verify a block are n hash computations over packets, one hash over the concatenation of packet hashes, one signature verification, and one IDA-decoding. We disregard the cost of IDA-coding, because there are efficient algorithms for erasure correction, such as Tornado codes that use only XOR operations and operate in linear time of the block size, which can replace IDA-coding in SAIDA. Hence, the time it takes for a receiver to verify a block is $t_{sig} + (n\lceil l/64 \rceil + \lceil s_{hash}n/64 \rceil)t_{hash}$. The communication overhead of SAIDA depends on the parameters of the IDA algorithm (or any other FEC technique used instead), and is equal to $(s_{sig} + ns_{hash})/n_{enough}$ bytes per packet. The receiver needs to buffer at least n_{enough} packets, which typically is a significant fraction of n. Thus the receiver delay can be considered n/α, which results in a total delay of $2n/\alpha$ seconds when summed to the sender delay of n/α.

As an enhancement on SAIDA, Park and Cho presented eSAIDA [43]. In eSAIDA, one hash is designated for each pair of adjacent packets, rather than one for each packet as in SAIDA. This reduces the overhead, but will cause a packet to be unverifiable if its couple is not received. Thus, a packet in a block may also contain the hash value of its couple. The fraction of packets containing their couple's hash is parameterized by s ($0 \leq s < 1$) as an input, which governs a tradeoff between successful verification rate and communication overhead. Computations needed by eSAIDA per each block are $(1+s)n/2$ hash computations over packets, one hash over the concatenation of hashes of packet pairs, one signature verification, and one IDA-decoding that we neglect. Thus, the time it takes for eSAIDA to verify a block is $t_{sig} + (\lceil l/64 \rceil (1+s)n/2 + \lceil s_{hash}n/128 \rceil)t_{hash}$. The communication overhead of eSAIDA is $(s_{sig} + s_{hash}n/2)/n_{enough} + s_{hash}s$ bytes per block. The receiver buffer size and the total delay in eSAIDA are similar to those in SAIDA.

[2]See Rabin's Information Dispersal Algorithm (IDA) [42], which belongs to a broader set of erasure codes called Forward Error Correction (FEC) codes.

2.4.6 cSAIDA

Pannetrat et al. in [44] developed another improvement of SAIDA, which we call cSAIDA because it significantly reduces the communication overhead of SAIDA. Recall that in SAIDA, the concatenation of the packet hashes (H) along with a signature on $h(H)$ are FEC-coded (using the IDA algorithm [42]) and distributed among the n packets of the block. However, a considerable fraction of these packet hashes can be computed from the received packets. Thus, there is no need for the whole H to be transmitted. To achieve this, cSAIDA uses FEC coding twice as follows. First, a systematic erasure code is employed to encode H. A systematic erasure code encodes data pieces $D_1, D_2, \ldots, D_n$ into m $(m \geq n)$ pieces $D'_1, D'_2, \ldots, D'_m$ such that any subset of n pieces are sufficient for reconstructing the original data and the first n pieces of the encoded result are equal to the original data. That is, $D_i = D'_i$ $(1 \leq i \leq n)$. In this case, the extra redundancy pieces $D'_{n+1}, \ldots, D'_m$ are called parity check pieces. Denoting the expected loss rate by ρ $(0 \leq \rho < 1)$, in cSAIDA, the n pieces of H are systematically FEC-coded into $\lceil n + \rho n \rceil$ pieces $H'_1, H'_2, \ldots, H'_{\lceil n+\rho n \rceil}$. Then, only parity pieces $H'_{n+1}, \ldots, H'_{\lceil n+\rho n \rceil}$ and a signature on $h(H)$ are concatenated, divided into $\lfloor n(1-\rho) \rfloor$ pieces, and FEC-coded again into n pieces to be attached to all the n packets of the block. At the receiver side, if $\lfloor n(1-\rho) \rfloor$ (i.e., n_{enough}) packets are successfully received, then $\lfloor n(1-\rho) \rfloor$ of the hash values, the signature on $h(H)$, and the parity pieces $H'_{n+1}, \ldots, H'_{\lceil n+\rho n \rceil}$ can all be successfully retrieved. Thus, H can be reconstructed in order to verify the whole block using the signature.

Computations needed by cSAIDA to verify a block are equal to those of SAIDA, plus one extra FEC-decoding. Since we disregard the cost of FEC-decoding, the time it takes cSAIDA to verify a block is $t_{sig} + (n\lceil l/64 \rceil + \lceil s_{hash} n/64 \rceil) t_{hash}$ seconds. The communication overhead of cSAIDA is $(s_{sig} + (n - n_{enough}) s_{hash})/n_{enough}$ bytes per packet. The total delay and receiver buffer size of cSAIDA are similar to those of SIADA and eSAIDA.

2.4.7 TFDP

Habib et al. [45] presented TFDP (Tree-based Forward Digest Protocol) for offline P2P streaming, i.e., distribution of already-recorded media files. Similar to SAIDA, packets are hashed, and packet hashes are concatenated and hashed again to form the digest of the block. Unlike SAIDA, only one signature is generated for the whole stream, because the entire file being streamed is given initially. Similar to tree chaining [39], a Merkle hash tree [40] is built over blocks of the stream, whose leaves are block digests. The root of this tree is then signed. At the beginning of the streaming session, the client receives the signed root of the tree along with a list of senders. The client asks one of the senders for information needed to verify a number (x) of blocks, which includes hashes of packets of these blocks (FEC-coded), digests of the x blocks, and auxiliary digests in the tree needed for reconstructing and verifying the root digest. Having received the digests, the client checks their genuineness by re-calculating the root hash. Once verified, they can be used for verifying authenticity of the x data blocks, one by one once they arrive. The client repeats the same procedure for the next sets of blocks. Therefore, the communication overhead is amortized over a number of blocks.

Computations needed by TFDP for each block (assuming $x = 1$ for simplicity) are n hash computations over packets, one hash over the concatenation of packet hashes, and $\lceil \log_2 N \rceil$ hashes corresponding to nodes interior to the tree, which takes $(n\lceil l/64 \rceil + \lceil s_{hash}n/64 \rceil + \lceil \log_2 N \rceil \lceil 2s_{hash}/64 \rceil)t_{hash}$ seconds. Thus, compared to computations of other schemes, which all include one signature per block, TFDP's computations are much cheaper. The communication overhead of TFDP is at most equal to $s_{hash}(n/n_{enough} + (1 + \log_2 N)/n)$ bytes per packet, where N denotes the total number of blocks in the file. This worst case communication overhead occurs when the clients requests one block, i.e., $x = 1$. In TFDP, a receiver needs to buffer at most n packets. In addition, since TFDP works for offline streams only, the delay is not relevant.

2.4.8 Other Authentication Schemes

Perrig et al. proposed TESLA (Timed Efficient Stream Loss-tolerant Authentication) and EMSS (Efficient Multi-chained Stream Signature) in [46] to authenticate online streams in lossy networks. The core idea of TESLA is to take advantage of the online nature of the stream. That is, after a packet is received by all receivers, any possibility to forge the contents of that packet is of no danger, i.e., it is enough to eliminate this possibility only during the time the packet is in transit. TESLA and a later improvement on it [47] are very interesting techniques for authentication of online multicast streams. However, because they depend on the online nature of the stream and on a time synchronization (though loose) between the sender and receivers, they are not applicable to other streaming scenarios, such as video on demand, where receivers may be receiving different parts of the stream at the same time, or to P2P streaming, where delay constraining is extremely difficult. Therefore, TESLA cannot be a solution for the general stream authentication problem. The other method presented in [46], EMSS, does not depend on the online nature of the stream, which comes at the cost of more expensive computations. For a block of packets, the hash of each packet is put in multiple later packets of the block, and a signature is put in the last packet of the block. The signature suffices for verifying the entire block of packets. EMSS does not clearly specify how to designate hash links. It is one of the earliest techniques based on hash replication, and is outperformed by other schemes summarized in this section in terms of loss tolerability with less overheads. Thus, we do not include EMSS in our analysis.

The authentication schemes presented thus far do not use or depend on the characteristics of the video stream. There are other schemes that do use these characteristics in the authentication process. These are usually called *content-based* authentication schemes. Conducting a quantitative analysis of these schemes is difficult, because of the dependence on the video characteristics which are quite diverse and varying. Nonetheless, to make this chapter comprehensive, we summarize a few of such schemes in this category.

Liang et al. [48] proposed a method for extracting a content-based feature vector from video. The goal is to inexpensively extract the feature vector such that it is robust against transcoding operations. The feature vector is then embedded back into the video as a watermark. Sun et al. in [49] extended an earlier work of theirs on image authentication [50] for video authentication with robustness against transcoding. In [50], some content-based invariant features of an image are extracted, and the feature vector is FEC-coded. The resulting codeword can be partitioned into two parts: the feature vector itself and parity check bits. Only the parity check bits are taken as a kind of Message Authentication Code (MAC) for the image and are then embedded back into the image as a watermark. Furthermore, the whole FEC codeword (extracted feature vector and its parity check bits) is hashed and then signed to form the signature of the image. The signature can either be attached to the image or again be embedded back into the image as another watermark. At the receiver side, to verify the authenticity, the feature vector is extracted from the content, which together with the watermark (parity check bits) can re-construct the FEC codeword whose signed hash is received and ready for verification. Feature extraction and watermark embedding are done in the transform domain, since transcoding operations are usually performed in that domain. Robustness against re-quantization is achieved by following the approach taken in an earlier work [51].

Another instance of content-based approaches is the scheme presented by Pradeep et al. [52] for video authentication based on Shamir's secret sharing [53]. They divide a video into shots, and then extract a number of key frames from each shot. They define three levels of authentication and create one authentication frame for (i) each sequence of frames between two successive key frames, (ii) each shot, and (iii) the whole video.

For detecting forgery of offline (recorded) videos, Wang et al. in [54] approached authentication of MPEG-coded video by detecting the effects of double MPEG compression. Recall the video coding process where each Group of Pictures (GoP) consists of an I-frame followed by a number of P- and B-frames. When a sequence of frames is removed from (or inserted to) a video and the video is encoded again, the type (I, P, or B) of some frames could change. Accordingly, two frames belonging to a GoP in the new video might belong to two different GoPs in the old one. Since the retrieved content of P- and B-frames is mainly based on the motion vectors, there is a strong correlation between frames belonging to the same GoP. However, when some frames are in different GoPs in the old video and in the same GoP in the new one, this correlation becomes weaker, and the motion error is increased; this increased motion error is observed periodically in the video sequence. In addition to migration between GoPs, there is another effect when a video is compressed twice. For the I-frames that still remain I-frame in the new coded video, the quantization factor would be different if the old and new videos are encoded at different bitrates. Such double quantization results in a detectable statistical pattern that occurs periodically in the histogram of DCT coefficients. Taking advantage of these two behaviors, a forgery can be detected.

2.5 Evaluation of Authentication Schemes for Nonscalable Streams

In this section, we compare nonscalable stream authentication techniques summarized in Section 2.4. We first conduct a numerical analysis of the computation cost and communication overhead of the schemes. Then, we analyze their tolerance to packet losses using simulation under different loss models. Then, a summary of our findings is presented. Finally, we propose to combine the best features of two authentication schemes to design a more efficient one.

Table 2.2: Summary of the analysis of authentication schemes for nonscalable streams.

	Computation cost: time to verify a block (sec)	Communication overhead (bytes per packet)	Total delay (sec)	Receiver buff size (pkts)
Hash chaining	$t_{sig} + t_{hash} n \lceil l/64 \rceil$	$\frac{s_{sig}}{n} + s_{hash}$	n/α	1
Aug. chain	$t_{sig} + t_{hash} n \lceil l/64 \rceil$	$\frac{n_{sig} s_{sig}}{n} + 2s_{hash}$	n/α	n
Butterfly chaining	$t_{sig} + t_{hash} n \lceil l/64 \rceil$	$\frac{n_{sig}(s_{sig} + n_{rows} s_{hash})}{n} + \frac{s_{hash}(2n - n_{rows})}{n}$	n/α	n
Tree chaining	$t_{sig} + t_{hash} \Big(n \lceil l/64 \rceil + (\lceil n \log_2 n \rceil - n) \lceil 2s_{hash}/64 \rceil \Big)$	$s_{sig} + \lceil \log_2 n \rceil s_{hash}$	n/α	1
SAIDA	$t_{sig} + t_{hash} \Big(n \lceil l/64 \rceil + \lceil s_{hash} n/64 \rceil \Big)$	$\frac{s_{sig} + n s_{hash}}{n_{enough}}$	$2n/\alpha$	n
eSAIDA	$t_{sig} + t_{hash} \Big(\lceil l/64 \rceil (1+s) n/2 + \lceil s_{hash} n/128 \rceil \Big)$	$\frac{s_{sig} + s_{hash} n/2}{n_{enough}} + s_{hash} s$	$2n/\alpha$	n
cSAIDA	$t_{sig} + t_{hash} \Big(n \lceil l/64 \rceil + \lceil s_{hash} n/64 \rceil \Big)$	$\frac{s_{sig} + (n - n_{enough}) s_{hash}}{n_{enough}}$	$2n/\alpha$	n
TFDP	$t_{hash} \Big(n \lceil l/64 \rceil + \lceil s_{hash} n/64 \rceil + \lceil \log_2 N \rceil \lceil 2s_{hash}/64 \rceil \Big)$	$s_{hash}(\frac{n}{n_{enough}} + \frac{1}{n} + \frac{\log_2(N/x)}{nx})$	NA	n

2.5.1 Numerical Analysis

We present in Table 2.2 a summary of the analysis of all authentication schemes presented in the previous section. Each row corresponds to one authentication scheme, and the four columns represent four of the five performance metrics defined in Section 2.3. To shed some light on the performance of the different authentication schemes, we numerically analyze their computation cost and communication overhead as the number of packets in the group n varies. n is the most important parameter that impacts the performance of the authentication schemes. To conduct this analysis, we choose realistic values for other parameters as summarized in Table 2.1 and discussed below.

Choosing Values of the Parameters for Simulation. We first choose the values of packet size and packet sending rate. To improve the performance of video streaming applications over Internet, it is usually preferred to fit each application data unit in an IP packet, which should be smaller than the maximum transmission unit (MTU) [55]. Assuming a video encoding rate of 320 kbps and an MTU of 1,500 bytes, the data in a packet should be roughly 1,400 bytes. This takes into account the RTP/UDP/IP headers and authentication information attached to each packet. Thus, the packet rate (α) is equal to $\alpha = \frac{320 \text{ kbps}}{1400 \text{ bytes}} \simeq$ 30 packets per second.

Next, we estimate the computation costs of the digital signature and hashing operations. Digital signature operations are often very costly, because they involve modular multiplication of very large numbers. Since we assume that the signer is powerful enough, RSA [28] is an appropriate choice as the digital signature scheme, because its verification can be done efficiently when the public key is chosen properly, e.g., a value of 65537 ($2^{16} + 1$) for the public exponent. The size of a 1024-bit RSA signature is $s_{sig} = 128$ bytes. t_{sig} in Table 2.1 denotes the time it takes to verify a 1024-bit RSA signature with such exponent. That is estimated for a typical limited-capability device, by using a small fraction (5-10%) of its CPU time. It is experimented in [56] that 1024-bit RSA verification when the public exponent is 65537 takes about 5 milliseconds on an iPAQ H3630 with a 206 MHz StrongARM processor, 32 MB of RAM, and running Windows CE Pocket PC 2002. A similar experiment [57] measures 1024-bit RSA verification time on a number of J2ME-enabled mobile devices and reports that the time taken ranges from a few to more than a hundred milliseconds. Since the authentication scheme should not take more than a small fraction of CPU time, e.g., 5-10%, and considering a safety margin, we took the value $t_{sig} = 500$ millisecond in Table 2.1. For the hashing algorithm, SHA-1 [26] and MD5 [27] are two popular one-way hash functions, both of which operate on blocks of 512 bits. MD5 has higher performance and smaller digest size, but some successful cryptanalysis have been done on MD5 and algorithms have been proposed for finding collisions [58, 59]. Although these cryptanalysis on MD5 are far from being practical for breaking a system in real-time, we chose SHA-1 as the hash algorithm for our evaluations. The digest size s_{hash} for SHA-1 equals to 20 bytes.

Results of the Analysis. We plot in Figure 2.3 the computation costs for all considered authentication schemes as n varies from 0 to 150 packets. The figure shows that the TFDP scheme is more efficient than the others. This is because it does not verify a digital signature per each block. Recall, however, that to build the Merkle hash tree used in TFDP, the whole

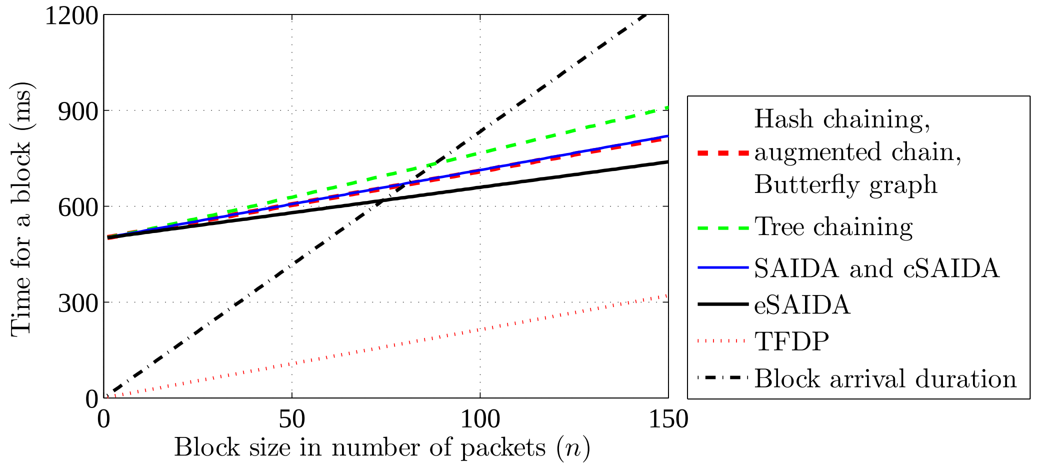


Figure 2.3: Computation cost (time to verify a block of packets) versus block size.

stream need to be available. This makes TFDP only suitable for on-demand streaming of pre-encoded video streams.

In Figure 2.3, we also plot the time it takes for a block of packets of size n to arrive at the receiver, which is computed from the packet generation rate α. Clearly, the block arrival time should be larger than the block verification time. Otherwise, the receiver will not have enough processing capacity to verify the authenticity of packets in real time. Therefore, by looking at Figure 2.3, we notice that small block sizes may not be suitable for all authentication schemes except TFDP. This implies that a minimum block size is required to support devices with limited processing capacity. For example, for the data used in producing Figure 2.3, a block size of approximately 100 packets would be needed to safely use any of the authentication schemes. This also indicates that the receiver needs to allocate a buffer of size at least 100 packets for most of the schemes (see buffering requirements in Table 2.2).

Next, we plot the per-packet communication overhead against the block size n for all authentication schemes in Figure 2.4. The communication overhead is the number of additional bytes added to each packet to implement the authentication scheme. The figure shows that the cSAIDA authentication scheme imposes the least amount of communication overhead. Moreover, the per-packet overhead stabilizes for block sizes greater than 50 packets for all schemes, except for the simple tree chaining scheme in which the overhead keeps increasing as the block size increases.

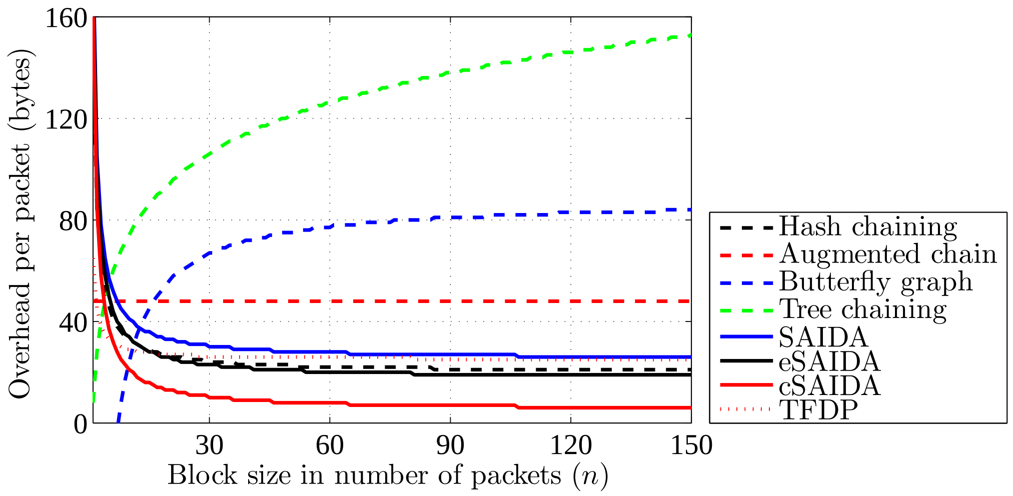


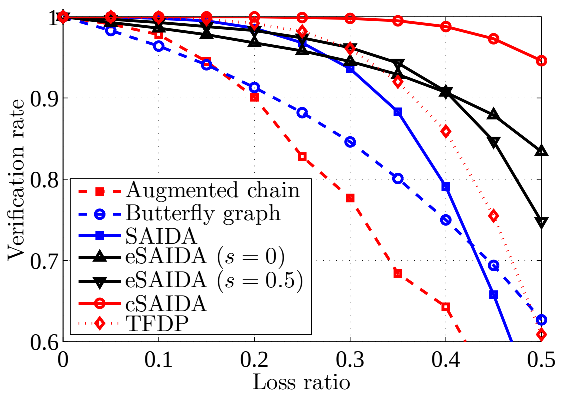
Figure 2.4: Communication overhead (bytes per packets) versus block size.

2.5.2 Simulation

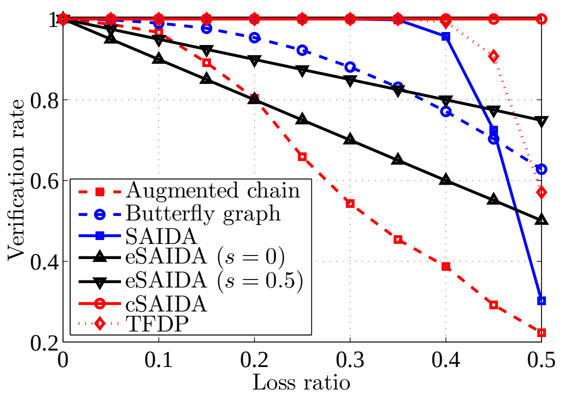
Simulation Setup. We have implemented all authentication schemes described in Section 2.4 in a simulator to study the impact of packet losses on their performance. The main performance metric used is the packet verification rate, which is the fraction of packets successfully verified over all received packets when packets carrying the authentication information could be lost. Notice that we are analyzing the loss tolerance for each authentication scheme, not the loss tolerance of the video decoder which may employ various error concealment methods.

We consider two common models for packet losses: bursty and random. The bursty loss model is typical in wired networks where a sequence of packets may get dropped because of a buffer overflow in one of the routers on the network path from sender to receiver. Whereas the random loss model is usually used to capture bit errors in wireless environments. Notice that some multimedia streaming techniques over wired networks use interleaved packetization of data, which can change the observed loss pattern at the receiver from bursty to random. Thus, it is important to analyze the performance of the authentication schemes under both models of packet losses.

For simulating bursty packet losses, we implemented a two-state Markov chain, as it has been shown to accurately model bursty losses [60]. In the two-state Markov chain, one state indicates that a packet is received and the other indicates the packet is lost. Transition



(a) Under the bursty loss model.



(b) Under the random loss model.

Figure 2.5: Verification rate versus packet loss ratio.

probabilities between these two states are computed based on the target average loss ratio and the expected burst length using the method in [41].

Values of the other parameters used in the simulation are listed in Table 2.1.

Simulation Results. The results for the packet verification rates versus average packet losses are given in Figure 2.5(a) for the bursty loss model, and in Figure 2.5(b) for the random loss model. The hash chaining and tree chaining schemes are not included in these figures, since the former one does not tolerate any packet loss and the latter always has a loss resilience of 100%; each packet in tree chaining carries all information needed for its verification. In figures 2.5(a) and 2.5(b), we fixed the communication overhead to 40 bytes per packet (except for the Augmented chain which has 42 bytes since it cannot work with 40 bytes).

A few observations can be made on these two figures. First, cSAIDA clearly exhibits the best resilience to the loss under both bursty and random loss models. For example, for bursty losses with an average loss rate of 40% in the authentication information, about 99% of the received packets can be verified. Second, the loss tolerance of the authentication schemes does indeed depend on the loss model, not only on the average loss rate. For example, with an average loss rate of 40% of the authentication information, the SAIDA scheme can verify up to 79% of the received packets under bursty losses, while this ratio is 96% under random losses. We now briefly discuss the reasons underlying these behaviors in the figures.

Table 2.3: Counteracting packet losses by different authentication schemes when the overhead is fixed to 40 bytes per packet (42 bytes for the Augmented chain).

Scheme	Loss counteraction
Augmented chain	Replicating the signature twice within a block.
Butterfly graph	The best compromise of number of edges and signature replication is local-searched.
SAIDA	An FEC factor of 1.9, i.e., 47% expected loss ratio.
eSAIDA ($s = 0$)	An FEC factor of 3.6, i.e., 72% expected loss ratio.
eSAIDA ($s = 0.5$)	An FEC factor of 2.7, i.e., 63% expected loss ratio.
cSAIDA	An FEC factor of 2.8, i.e., 65% expected loss ratio.
TFDP	An FEC factor of 2, i.e., 50% expected loss ratio.

Recall that loss is counteracted either by FEC-coding or by replicating some authentication information, i.e., digests and block signature. Let us call the fraction n/n_{enough} the FEC factor. Depending on the scheme, the overhead of 40 bytes per packet results in different FEC factors for FEC-based schemes (SAIDA variants and TFDP) or different number of replications for replication-based ones (Augmented chain and Butterfly graph), as shown in Table 2.3. For SAIDA, the 40 bytes per packet leads to FEC factors of 1.9, which means resistance to loss of up to 47% of a block, as indicated in the table. Calculation of FEC factors for cSAIDA and TFDP with 40 bytes per packet follow the same procedure. eSAIDA does not only rely on FEC. Because it couples every pair of packets together, it also attaches the hash value of a packet to its couple packet with a probability parameter s. The 40 bytes per packet for eSAIDA with $s = 0$ and $s = 0.5$ leads to the high FEC factors of 3.6 and 2.7, respectively. Thus, with losses up to 72% and 63%, the block signature and hash values of packet couples can be retrieved. However, since loss of a packet threatens verifiability of its couple, verification ratio of eSAIDA is not as high as cSAIDA even though its FEC factor is almost equal or higher. Augmented chain has a fixed number of edges and allows customization of loss resilience versus communication overhead only by varying the number of replications of the signature packet. The 42 bytes per packet allows it to replicate the signature twice within the block. The Butterfly graph allows customizing the communication overhead by replicating signature as well as varying the number of edges of the graph. These two parameters are in tradeoff with each other. We perform a local search to find the best balance for that in our simulations, given a fixed amount of communication overhead.

Recall that with 40 bytes per packet, the FEC factor of cSAIDA would be 2.8 and up to 65% loss is tolerated. This can be observed in Figure 2.5(b), which depicts that with random loss up to 50%, cSAIDA keeps the verification rate almost 1. However, the effect of bursty losses is different and more serious, as expected. Intuitively, if a loss of ratio 50% has a random pattern, for each block almost half of the packets are lost, which is easily resisted by the FEC factor of 2.8. On the other hand, with bursty loss of the same average ratio, a significant fraction of packets of a block could be lost during bursts, while some other blocks observe much less losses. This results in unverifiability of a few blocks, since the authentication information for those blocks cannot be retrieved from the received packets at all. This unverifiability can be seen in Figure 2.5(a), even though the loss ratio 50% is less than the ratio 65% we prepared the stream for. That is why the FEC-based schemes perform better under random loss model compared to bursty loss.

We can also notice the different decreasing behavior of FEC-based schemes (SAIDA variants and TFDP) with the two different loss patterns. With random losses, the verification rate sharply falls if the loss ratio exceeds the ratio supported by the FEC factor. This is clear in the plot for SAIDA and TFDP; same phenomenon happens to cSAIDA at 65% loss that is not shown in the figure. With bursty loss, on the other hand, the decreasing behavior is more smooth, because according to the above implication, there is no explicit loss ratio value, below which is easily tolerated and beyond which it suddenly gets too hard to resist.

The third point that can be noticed is the linear decreasing behavior of eSAIDA with random loss. The 40 bytes per packet allows eSAIDA with $s = 0$ (no packet hash value is attached to its couple) and with $s = 0.5$ (hash of half of packets is attached to their couples) to have FEC factors of 3.6 and 2.7, which resist 72% and 63% loss, respectively. That means, a random loss of up to 50% (Figure 2.5(b)) is easily tolerated by them. Hence, the decrease in verification ratio is not because of being unable to retrieve the FEC-coded authentication information of a block. Rather, the unverifiability of some packets, say p_x, is only because their couple, say p_{x+1}, is lost, and the hash of p_{x+1} is not attached to p_x, so the hash value $h(p_x||p_{x+1})$ cannot be reconstructed to be verified. The more the loss ratio, the more the number of packets that are missing the hash of their couples. Also, it can be observed that with $s = 0.5$, this increase in unverified packets ratio is less, as can be expected. With bursty loss, on the other hand, both packets of a pair are more likely to be lost together. That is, with each burst of loss, at most two packets can be left unverifiable: the ones right before and right after the burst begins and ends. Thus, unverifiability can be both due to not being able to retrieve authentication information of a block (which was

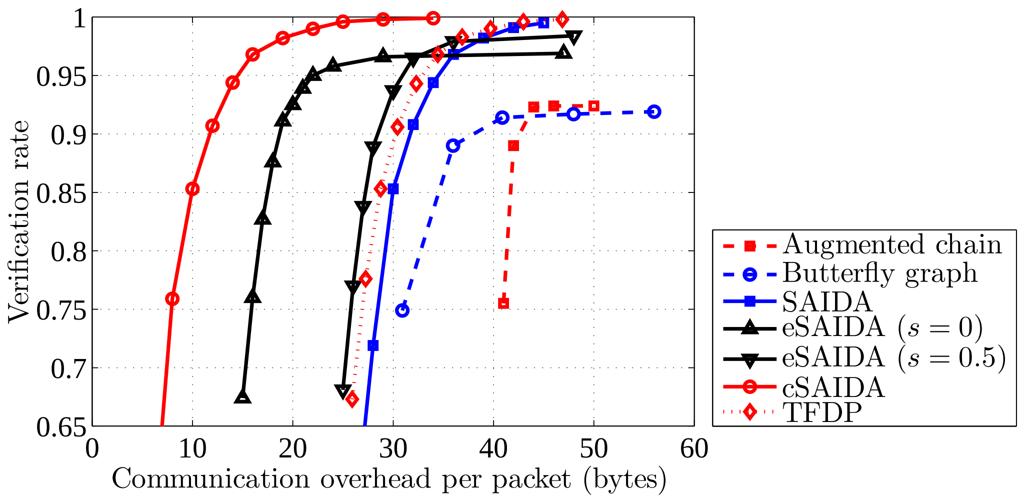


Figure 2.6: Verification rate versus communication overhead.

very unlikely with random loss below the loss ratio supported by the FEC factor) and not being able to verify a packet because its couple is missing. Therefore, this phenomenon, i.e., linear decrease of verification rate, does not take place.

We can also notice that, unlike most of the schemes, the Augmented chain performs worse under random loss model compared to the bursty one. That can be attributed to the structure of the augmented chain, which is designed to have least unverifiability effect with bursty losses. With each burst of loss, up to a few packets can be left unverifiable in the augmented chain. Thus, when each burst consists of one packet, i.e., random loss, the ratio of unverifiable packets increases. Also, the decreasing behavior of Augmented chain curves in Figures 2.5(a) and 2.5(b) is not so straight, which is most probably because we obtain the parameters p and a for the the Augmented chain scheme (see Section 2.4.2) by a local search for best verification rate, given a fixed amount of overhead.

Finally, we fix the average loss rate at 20% with bursty losses, and we vary the communication overhead per packet. That is done either by varying the FEC factor (for FEC-based schemes) or by changing the number of replication of the signature (for replication-based schemes). Then, we analyze the verification rates for different values of the per-packet overhead. The results for all authentication schemes are given in Figure 2.6. The figure confirms the efficiency of the cSAIDA scheme in carefully minimizing the number of bytes needed to encode the authentication information. With less than 30 additional bytes per packet, cSAIDA can achieve 100% verification rate under bursty losses with an average loss rate of 20%. Whereas other schemes need almost double this number of overhead bytes to achieve comparable loss resilience.

2.5.3 Summary and Discussion

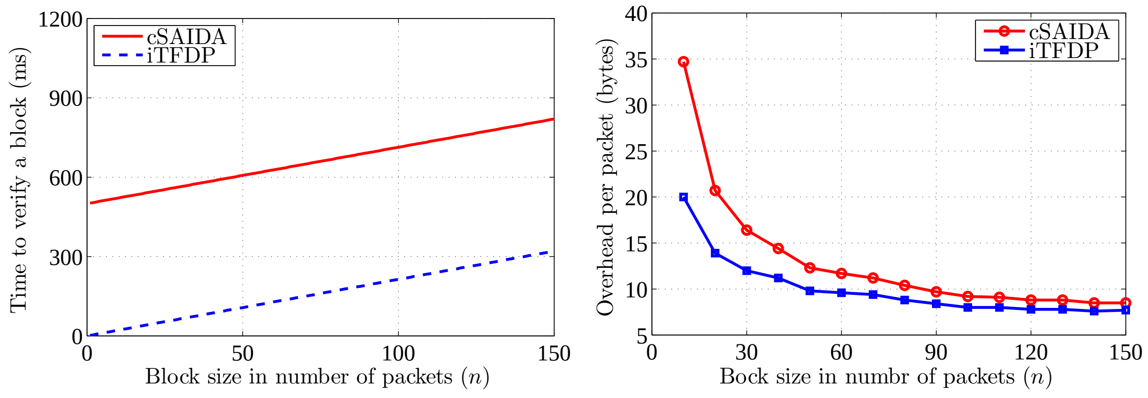
Our analysis and simulation with realistic parameters and in different environments indicate that cSAIDA—the improved version of the SAIDA authentication scheme proposed in [44]—imposes the least amount of communication overhead and achieves the best tolerance to the loss of the authentication information. cSAIDA capitalizes on the fact that not all hash values of packets in a block need to be transmitted, since a large portion of these hashes can be reconstructed from the received packets. cSAIDA, however, requires a digital signature verification per block, which is costly. The TFDP [45] scheme, on the other hand, is very efficient in terms of computation cost, but only for offline streams. That is because TFDP performs one digital signature verification for the whole stream, which requires the whole stream to be available. Therefore, TFDP is not suitable for live streaming applications where packets are generated online in real time.

In addition, as shown in Table 2.2, most of the authentication schemes for nonscalable video streams require the receiver to buffer a block of packets, which needs memory space. In case that the receiver has a limited memory space, the simple hash chaining [35] or tree chaining [39] authentication schemes can be used.

Furthermore, we mention that some streaming applications employ TCP to reliably transport data from the sender to receivers. TCP could be a possible option for streaming if there are infrequent packet losses and the round trip time is small. In this case, the simple hash chaining authentication scheme would suffice as loss resiliency and its associated complex operations in other authentication schemes are not needed.

2.5.4 iTFDP

We propose to combine the best features of cSAIDA (communication efficiency) and TFDP (computation efficiency) in designing an efficient authentication scheme for streaming of pre-encoded streams. We call this scheme the Improved TFDP and we refer to it by iTFDP. Similar to cSAIDA, iTFDP divides data into blocks, each with with n packets: $p_1, p_2, \ldots, p_n$ and it computes $H = h(p_1)||h(p_2)|| \cdots ||h(p_n)$. It also systematically FEC-codes the n pieces of H



(a) Computation cost. (b) Communication overhead.

Figure 2.7: Comparison between the proposed scheme (iTFDP) and cSAIDA.

into $m \geq n$ pieces $H_1, H_2, \ldots, H_m$, and only takes the parity symbols $H_{n+1}, H_{n+2}, \ldots, H_m$. These symbols are concatenated, divided into $\lfloor n(1-\rho) \rfloor$ pieces, and FEC-coded again into n pieces to be attached to all the n packets of the block. Unlike cSAIDA, iTFDP does not sign $h(H)$, whose authenticity is essential for verifying the block. Instead, it authenticates the values $h(H)$ of blocks using a Merkle hash tree, as done in TFDP. Accordingly, the computation cost can be significantly reduced since only one signature is designated for the whole stream. Therefore, iTFDP can achieve the low communication overhead of cSAIDA and the low computation cost of TFDP, without impacting other factors including delay and buffering requirements.

To validate the efficiency of iTFDP, we have implemented it in our simulator and compared it against cSAIDA. We stream a video sequence with 1 million packets in an environment with a bursty loss and average loss rate of 20%. All other parameters in this simulation are the same as in Table 2.1. In Figure 2.7, we plot the computation cost and communication overhead of iTFDP and cSAIDA. The figure clearly shows that iTFDP has a significantly smaller computation cost than cSAIDA, and iTFDP further reduces the (already-optimized) communication overhead of cSAIDA. In Figure 2.8, we compare the verification rate of the received packets for iTFDP versus cSAIDA in presence of bursty losses of the authentication information. The figure also shows that iTFDP improves the verification rate of cSAIDA. Therefore, iTFDP achieves its goals of reduced computation cost and communication overhead as well as high verification rate in presence of losses.

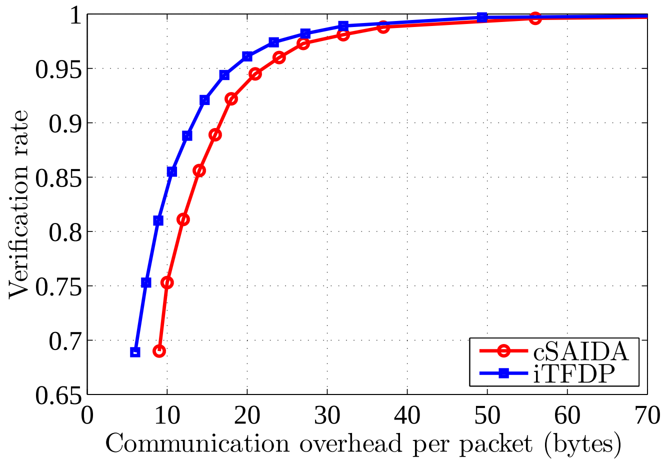


Figure 2.8: Verification rate of iTFDP versus cSAIDA for different per-packet overheads.

2.6 Authentication of Scalable Multimedia Streams

In the previous two sections, we presented and analyzed authentication schemes for multimedia streams that are encoded in a nonscalable manner. Nonscalable streams offer very limited flexibility in supporting heterogeneous receivers. In contrast, multimedia streams created using scalable video coding techniques can easily be adapted to support a wide range of diverse receivers and network conditions. This ease of adaptation comes at a cost of reduced coding efficiency, i.e., at the same bitrate a nonscalable stream yields a better visual quality than a scalable stream. Nevertheless, this quality gap has been significantly reduced with the recent H.264/SVC standard [8]. In this section, we present and analyze authentication schemes for scalable multimedia streams. It is important to note that any authentication scheme for a scalable stream must be able to authenticate all valid substreams that can be extracted from it. Substreams are truncated versions of the original full stream, created by dropping some layers. These truncations are intentionally made to customize the stream based on the capacity of the receiver and the network conditions, and they must be differentiated from illegitimate manipulations on the video.

Several authentication schemes for traditional scalable video streams have been proposed. These scalable streams consist of a base layer and a number of enhancement layers that progressively improve the video in terms of spatial resolution or visual quality. In other

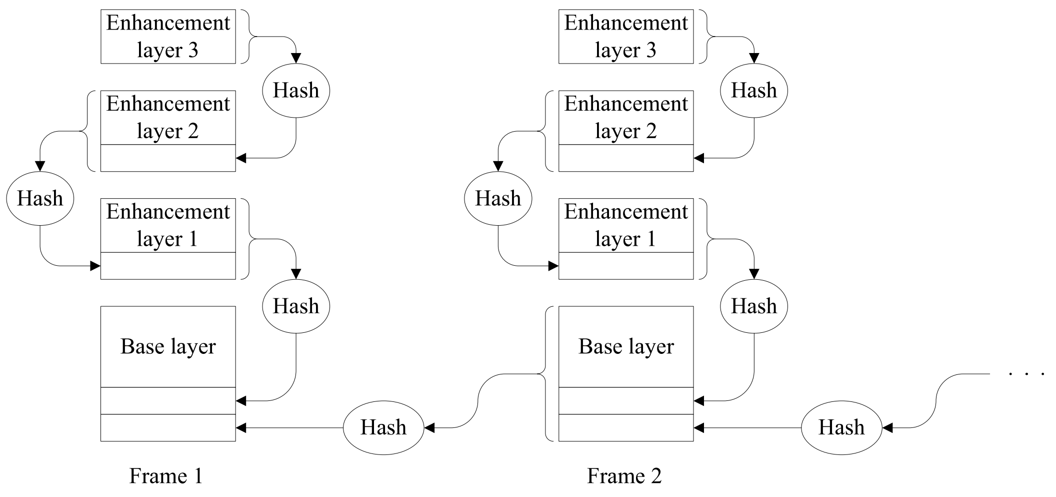


Figure 2.9: Authenticating scalable streams by hash chaining.

words, scalability is provided along one dimension and the layers in that dimension are cumulative, i.e., layer l is an atomic unit of data (cannot be truncated) and is only useful if all preceding $l-1$ layers are received. Authentication schemes for these scalable videos generally rely on two cryptographic techniques as their basis: hash chaining and Merkle hash trees [40], which are reviewed and evaluated in the following sections. At last, we discuss the shortcomings of these schemes for supporting recent scalable video streams which can provide different scalability types at the same time and with higher flexibility.

2.6.1 2-D Hash Chaining

A scalable stream is a sequence of video frames, where each frame consists of a set of layers. Clearly, we cannot treat the scalable stream as a set of independent incremental substreams to be authenticated separately, because in a typical substream the number of layers received out of consecutive frames may be different. Accordingly, we can think of a scalable stream as a sequence with two dimensions: horizontal and vertical. The horizontal dimension represents successive frames, while the vertical dimension represents the layers in each frame. An authentication scheme for scalable streams needs to validate the authenticity across the two dimensions.

The hash chaining scheme can be extended to scalable streams as follows [15, 61]. First, each enhancement layer of a frame is hashed and its hash is attached to its predecessor layer of the same frame. Thus, the base layer of a frame contains a digest of all enhancement layers

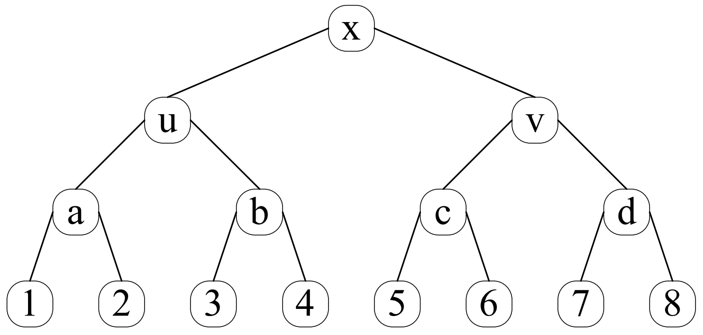


Figure 2.10: A Merkle hash tree built over 8 layers. As an example, for authenticating layer 1 only, the hash of layer 2 as well as b and v are needed.

of the frame. Then, the hash of the base layer (and the digest) of a frame is attached to the previous frame. The two-dimensional hash chaining scheme for scalable streams is illustrated in Figure 2.9. Notice that a substream with a base layer and 0 or more (consecutive) enhancement layers can be verified using the authentication information carried only in that substream—no additional information about the truncated layers is needed for the verification of the received layers. This is an important advantage especially for streaming in large-scale systems where proxy servers and/or third party content delivery networks can be involved in the streaming. In this case, these entities do not have to understand the authentication scheme and cooperate with it.

2.6.2 2-D Tree Chaining

In the tree chaining scheme for authenticating scalable streams, Merkle hash trees [40] are employed [13, 14, 62–65]. The enhancement layers of a video frame are hashed, and the hash values are arranged as leaves of a Merkle hash tree. Each interior node of this tree consists of the digest of its children. The root of the tree represents the frame digest. This is illustrated in Figure 2.10. Due to the collision-free property of the hash function, the whole set of layers represented by the leaves is authenticated if the root of the tree, i.e., the frame digest, is successfully verified. To authenticate the sequence of frames, another hash tree is built over the frame digests of a group of frames. Authentication schemes in [13, 62] specifically employ MPEG-4's tree-like structure for building the hash tree. In either case, upon removal of some layers, a receiver needs some extra digests for verifying the remaining layers, i.e., for reconstructing the root digest of the hash tree. For the example in Figure 2.10, if a client is receiving the base layer only (layer 1 in the figure), he/she needs digests 2, b, and v to reconstruct and verify the root. If the first five layers are being received, the client needs digests 6 and d. This means that a stream adaptation proxy on the delivery path must understand and be compatible with the authentication scheme so that it can attach the required hash values to the adapted substream. This may not be desirable as proxy servers of content delivery networks serve thousands of streams at the same time from different content providers who may be employing different authentication schemes.

To mitigate this problem, we can modify the 2-D tree chaining approach as follows. The content provider embeds in each layer i all information needed to authenticate the first i layers (excluding the information already carried by layers 0 through $i-1$). This trick makes the authentication scheme end-to-end, i.e., it no longer requires proxies to understand the scheme, given that the information are embedded in the stream in a transparent manner (compliant to the video stream format). However, this trick increases the communication overhead. For example, in Figure 2.10, to layer 1 the digests 2, b, and v are always attached, which are not necessary for many of the substreams. This increase in communication overhead is significant, as we analyze shortly.

2.6.3 Evaluation of Authentication Schemes for Scalable Streams

We analyze the above two approaches for authentication of scalable schemes. The computation cost analysis is similar to the analysis in Section 2.4, except that there is an additional hash computation for each enhancement layer, which is negligible. In addition, the delay and the receiver buffer size for the two-dimensional hash chaining and tree chaining are the same as the basic hash chaining and tree chaining schemes. The communication overheads, however, are different and they depend on whether an intermediate proxy server is aware of the authentication scheme or not.

We implemented the two-dimensional hash chaining and tree chaining in our simulator. Using this simulator, we analyze the communication overhead for both schemes. We encode a video sequence with 16 enhancement layers, with bitrates ranging from 128 kbps to nearly 1 Mbps. We create authentication information for the 16 layers using hash chaining and tree chaining. Then, we simulate a wide range of receivers with different bandwidth. Each receiver will obtain a truncated version of the stream based on its bandwidth. The truncated stream should contain the needed authentication information to verify the received layers.

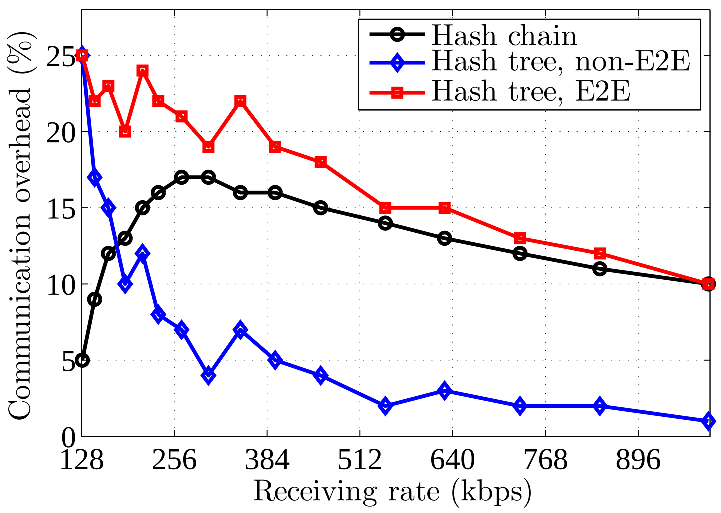
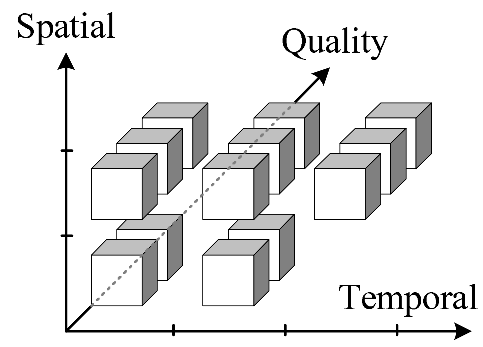


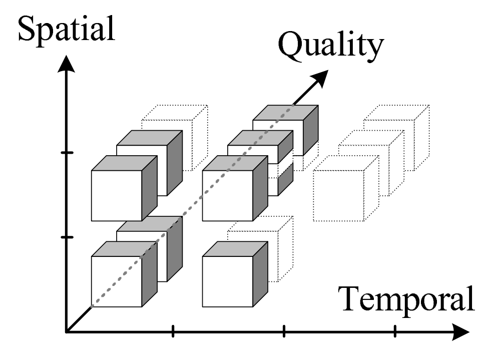
Figure 2.11: Communication overhead versus bitrate.

We measure the percentage of this authentication information to the total amount of data transmitted to the receiver. Figure 2.11 illustrates this overhead. In this figure, the vertical axis shows the overhead percentage, and the horizonal axis represents the bitrate of the substream being received, which is in proportion to the number of layers being received. The overhead of authentication through hash chaining is labeled "Hash chain" in the figure, that of hash trees is labeled "Hash tree, non-E2E", and the overhead of hash trees for end-to-end authentication, which is done through the trick described in Section 2.6.2, is labeled "Hash tree, E2E".

A few observations can be made on Figure 2.11. First, for low bandwidth receivers that obtain only a few layers, the hash chaining scheme imposes the least amount of communication overhead. Minimizing the overhead for such receivers is crucial as they have limited bandwidth in the first place. As the number of received layers increases, the tree chaining scheme becomes more desirable because of the efficiency in organizing the hashes in Merkle hash trees. The figure shows that the tree chaining scheme imposes lower overhead than hash chaining in most cases, except when the number of layers is very few. However, authentication through hash trees has limited applicability. When considering the general case of large-scale content distribution, where third-party proxies can be involved in adapting scalable streams, end-to-end authentication is desired. In this case, the use of hash chains is



(a) A video encoded in three temporal, two spatial, and up to three quality layers.



(b) A possible substream of Figure 2.12(a) with two temporal layers, both of the spatial layers, and a valid subset of quality layers.

Figure 2.12: A three-dimensionally scalable video and a possible substream of that.

more appropriate than hash trees since the “Hash chain” curve is always below the “Hash tree, E2E” curve in Figure 2.11 for all bitrates.

2.6.4 Limitations of Current Schemes for Modern Scalable Streams

Recent scalable video streams offer great flexibility while incurring much lower overheads than traditional scalable videos [6]. For example, the Scalable Video Coding (SVC) extension of the state-of-the-art H.264/AVC video coding technique, known as the H.264/SVC [8] standard, supports adapting a video stream along three scalability dimensions: temporal, spatial, and quality, which provide different frame rates, different spatial resolutions, and different visual qualities, respectively. This three-dimensional scalability model, which is depicted in Fig. 2.12(a), is more general than the previous, and much simpler, linear layered models. It allows different combinations of layers along the three dimensions. Even for the same number of layers, there could be several possible *paths* through the scalability cube to achieve it [8]. For example, a possible substream of Figure 2.12(a) is shown in Figure 2.12(b) with shaded cubes, in which the first two temporal layers, both of the spatial layers, and a valid subset of quality layers exist.

Because of the many possible combinations of layers, authentication schemes for the traditional, and much simpler, cumulative layered video, are not applicable to this model. Even for traditional layered videos, these schemes may incur a significant communication overhead if the stream is providing more than a limited flexibility, because the number of hash values added to the stream will be in proportion to the number of layers of a video frame; we discuss this issue in Section 3.6. In addition, the new useful coding tools designed for scalable coding, such as hierarchical prediction of frames and medium-grained scalability, also require new authentication algorithms. Accordingly, an efficient and end-to-end authentication service for modern scalable streams cannot be provided by current authentication schemes or by simple extensions of them such as forming a hash chain or tree on each scalability dimension. For example, if previous schemes are applied on a temporal scalable stream, frames in a temporal layer of the stream has to be all kept or all dropped together, since these schemes operate on a layer basis. In addition, applying previous techniques to authenticate quality enhancement packets may result in unverifiability of many of the received packets, because they are not necessarily dropped in a cumulative manner. To the best of our knowledge, there are no authentication schemes in the literature that can efficiently support the full flexibility of the three-dimensional scalability model.

In the next chapter, we take into account the complete scalability structure of three-dimensional scalable streams to authenticate all their valid substreams. We also propose an additional algorithm for significantly reducing the communication overhead.

2.7 Summary

In this chapter we have surveyed, analyzed, and compared the most important solutions proposed in the literature for the problem of verifying the authenticity of multimedia streams. We carried out numeric analyses and simulations for the schemes to study their performance in terms of computation cost, communication overhead, delay, receiver buffer size, and tolerance to packet losses. We considered authentication schemes for both nonscalable and scalable multimedia streams. For nonscalable streams, we found that the scheme proposed in [44] (denoted by cSAIDA) imposes the least amount of communication overhead and achieves the best tolerance to the loss of authentication information. On the other hand, the scheme proposed in [45] (named TFDP) is very efficient in terms of computation cost, but only for offline (on-demand) streaming. We then proposed an authentication scheme for on-demand streaming that combines the advantages of both cSAIDA (communication efficiency) and TFDP (computation efficiency), i.e., it has a significantly lower computation cost than cSAIDA while it maintains its communication efficiency and loss tolerance. Our evaluations show that our proposed authentication scheme is the best performing one for on-demand streaming applications; for live streaming, on the other hand, cSAIDA is a good candidate.

For scalable multimedia streams, we showed that authentication schemes based on hash trees are more efficient in terms of communication overhead, but only when proxies involved in the delivery and adaptation of streams are compatible with the authentication scheme. Schemes based on hash chaining, on the other hand, do not require this cooperation and can provide end-to-end authentication. Nevertheless, we showed that current schemes for scalable video streams fail to support the full flexibility of recent scalable streams, e.g., H.264/SVC, which provides three scalability types at the same time and with higher flexibility, yet with higher coding efficiency than traditional scalable videos.

Chapter 3

The Proposed SVC Stream Authentication Scheme

In this chapter, we design, analyze, and implement a new authentication scheme for end-to-end secure delivery of scalable video streams. We also propose an additional algorithm for minimizing the communication overhead of the authentication scheme. At last, we evaluate different performance aspects of the proposed solutions.

3.1 Introduction

As reviewed in the previous chapter, recent Scalable Video Coding (SVC) techniques can generate highly flexible video streams and with higher coding efficiency, compared to traditional layered videos [6]. This is achieved by employing a new three-dimensional scalability model that enables three types of scalability at the same time. In addition, new coding tools are employed for providing these scalability types, such as hierarchical prediction of frames, new intra-frame prediction possibilities, and Medium-Grained Scalability (MGS). The state-of-the-art realization of this scalability model is the H.264/SVC video coding technique, which is standardized recently and is being increasingly adopted in many applications, e.g., [9–12]. However, to the best of our knowledge, there are no schemes in the literature that can efficiently support the new scalability model and provide an end-to-end authentication service for SVC streams.

We propose a new authentication scheme that supports the full flexibility of three-dimensionally scalable video streams. We analytically show that the proposed scheme guarantees correct verification of any valid substream extracted from the original stream. The scheme is designed for *end-to-end* authentication of streams. That is, a third-party content delivery network in charge of delivering (and possibly adapting) the streams does not have to be aware of, or be compatible with, the authentication scheme, which is an important advantage. Furthermore, we propose an additional algorithm for minimizing the communication overhead imposed by the authentication scheme, which also can be used to minimize the communication overhead of other scalable stream authentication schemes (for traditional scalable streams). We then implement our authentication scheme for H.264/SVC streams in a prototype called *svcAuth*, which is available as an open source library and can be employed by any multimedia streaming application as a transparent add-on, without requiring changes to the encoders/decoders. We conduct a simulation study with real video traces to evaluate different aspects of our scheme. Our results show that our scheme is robust against packet losses, incurs low computational cost and buffer requirement, and is suitable for live streaming as it has short delay. Furthermore, it adds small communication overhead, particularly after using the overhead reduction algorithm.

In the rest of this chapter, we first review the related works in Section 3.2. Then, we provide a background in Section 3.3 on the structure of H.264/SVC scalable streams and how it realizes the three-dimensional scalability model, and we demonstrate the importance of authenticating all extractable layers. Our authentication scheme is described in Section 3.4, and its security and complexity are analyzed in Section 3.5. Section 3.6 presents an algorithm for minimizing the communication overhead. We evaluate our scheme in Section 3.7. Section 3.8 gives an overview of the svcAuth library. A summary of this chapter is provided in Section 3.9.

3.2 Related Work

We reviewed and analyzed previous schemes for authenticating multimedia streams in the previous chapter. To authenticate scalable video streams in particular, we saw that previous schemes are designed for traditional cumulative layered videos, and they or their simple extensions cannot support scalable streams encoded using recent scalable coding techniques. Another work related to ours is a high level framework for Secure Scalable

Streaming (SSS) [66], which mainly focuses on the encryption of scalable videos and on enabling the proxies to perform adaptations without requiring decryption. For authentication, however, the SSS framework again recommends using Merkle hash trees, which requires the cooperation of proxies (see Section 2.6.2).

For supporting adaptation of video streams, an alternative approach to authenticating substreams could be to follow a content-based approach. In this approach, e.g., [48, 52], as reviewed in Section 2.4.8, the general procedure is to extract a feature set from the video content that is robust against adaptations, but fragile against malicious manipulations. The features are then signed and attached to the stream. However, in these approaches, there is no clear boundary for differentiating valid changes to the content from malicious ones, e.g., [52] relies on threshold numbers provided as input. In addition, it is not clear how significantly one can tamper with the video while preserving the feature set, e.g., [48] uses the energy distribution of I-frames as the feature set, which is not difficult to preserve while changing the content. An alternative way for making sure the video is not tampered with is that the sender embeds/hides a *watermark* inside the video. The watermark could be a shared secret between the sender and receivers [67], or a digital signature on the video content [68]. The former case needs to trust all receivers, which is not desirable. In the latter case, the problem of deciding how to extract robust features to sign still exists. In fact, content-based and watermarking approaches are more suitable for authenticating video streams that are adapted by traditional stream adaptation techniques such as transcoding and re-compression.

3.3 Background

3.3.1 Overview of H.264/SVC

The recently standardized H.264/SVC video coding standard [8] adds scalability to the widely used H.264/AVC video coding technique [7]. In addition to generating highly flexible video streams, H.264/SVC significantly outperforms previous scalable coding techniques in terms of coding efficiency [6]. That is, at the same bitrate, it provides a higher visual quality. H.264/SVC supports temporal, spatial, and quality scalability at the same time.

Temporal scalability is achieved by employing a hierarchical prediction structure among video frames belonging to the same Group-of-Pictures (GoP), as shown in Figure 3.1. In this structure, frames of higher temporal layers can only be predicted from lower temporal

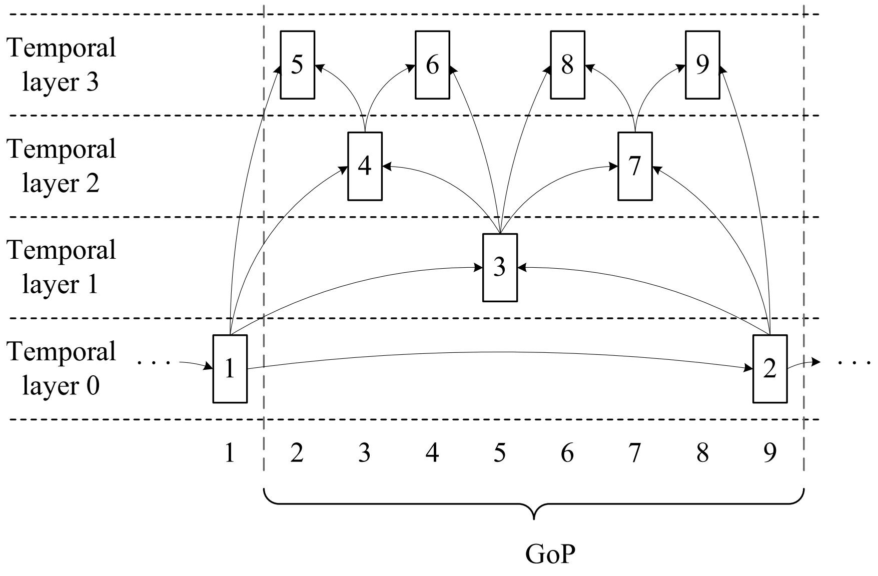
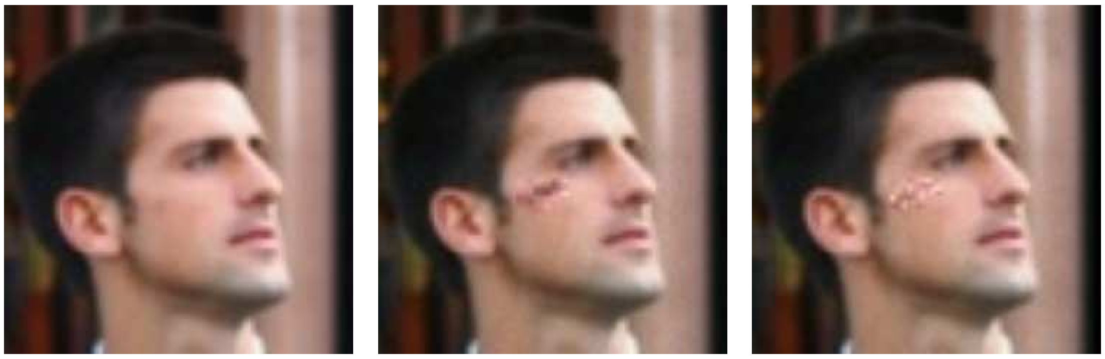


Figure 3.1: Hierarchical prediction structure of H.264/SVC temporal scalability. Arrows represent prediction. Numbers listed in the bottom row show the displaying order, while numbers inside frames show the coding order.

layers. A GoP consists of one frame in the temporal base layer, which is generally coded as P-frame, and several hierarchically coded B-frames that are located between the temporal base layer frames. In the spatial scalability of SVC, a spatial layer s of a frame can be predicted from the s-th spatial layer of some other frames (in lower temporal layers), as well as lower spatial layers in its own frame. For providing quality scalability, there are two different possibilities. The first one follows the spatial scalability structure, but assigns the same resolution and different quantization parameters to layers. This produces a Coarse-Grained Scalable (CGS) video with limited number of quality layers. A finer granularity can be provided by the second possibility, which uses Medium-Grained Scalability (MGS) coding to divide a single CGS quality layer into multiple sub-layers, which are referred to as MGS layers. This is done by partitioning the residual DCT coefficients of a CGS layer into multiple MGS layers. A stream can be truncated at any CGS or MGS layer. In addition, some packets of an MGS layer can be discarded, while the remaining ones can still be decoded to improve quality. Packet discarding can be done in different ways, depending on the bitstream extraction process [69]. H.264/SVC allows Up to 7 temporal, 8 spatial, and 16 quality layers [8].

In H.264/SVC, the coded video data and other related information are organized into



(a) Original picture. (b) Result of tampering with the first MGS layer (after 4 frames). (c) Result of tampering with the second MGS layer (after 4 frames).

Figure 3.2: Results of tampering with one MGS packet.

Network Abstraction Layer (NAL) units, which we alternatively refer to as a *video packet* or a *truncation unit* because these are the smallest units that can be truncated from an SVC stream. Each NAL unit, has *temporal_id*, *spatial_id*, and *quality_id* field in its header, which identify to which temporal, spatial, and quality layer the NAL unit belongs to. NAL units can be Video Coding Layer (VCL) units, which contain the coded video data, or non-VCL NAL units, which contain associated additional information.

3.3.2 Importance of Protecting All Layers

We aim at authenticating every video packet in a received substream. For this purpose, we need to protect every packet from potential malicious manipulations. One might argue that this may not be necessary and it could suffice, for example, to authenticate every two or more packets together; the two packets then, if both are received, can be verified, but an individual one cannot. This argument is based on the conjecture that exploiting a few single packets which are typically in enhancement layers can only have a quality enhancement/degradation effect, and thus cannot lead to a successful manipulation of the video content.

We show in the following that it is necessary to verify all portions of the received data and discard the unverifiable packets. In H.264/SVC for quality scalable coding with MGS, the highest quality picture of a frame is often used for motion compensation at receiver side [8]. Consequently, a change in the small unprotected portion of some highest few MGS layers will fast propagate through frames of a GoP, since those layers are used as prediction references for other frames. To confirm the importance of authenticating every packet, we empirically demonstrate a simple attack on an SVC video by manipulating one enhancement packet only. The original video is a short sequence consisting of 10 frames which shows the face of a man with no motion, as depicted in Figure 3.2(a). The video is encoded as one base and one CGS quality enhancement layer, whose quantization factors are 30 and 0 (no quantization), respectively. Transform coefficients of the CGS enhancement layer are further divided into three MGS layers, consisting of 4, 6, and 6 coefficients in the zigzag traversal order of the 4×4 coefficient table. Each MGS layer consists of a few small video packet. The tampering is done by modifying a small portion of the *enhancement data*, trying to add a simple scratch to the face. We first assume we are allowed only to tamper with up to one packet of the first MGS layer, as the base layer is protected. We gradually modify over a few successive frames one packet of the first MGS layer in each frame. After only four frames, we could make a meaningful alteration on the video by creating a scratch on the face of the man, as shown in Figure 3.2(b). In another attack, we assume that packets of the base layer and the first MGS layer are protected, and we (as attacker) are allowed only to modify a single packet in the second MGS layer. Again after a few frames, we were able to meaningfully tamper with the video content, as shown in Figure 3.2(c)

This experiment highlights the risk of leaving any portion of the video data unprotected. Notice that, we tried to tamper with the video in a *very simple* way by replacing the small number of unprotected transform coefficients with those of the desired picture. Now consider a real attacker who chooses the target video content carefully and/or employs more complicated image processing techniques for replacing unprotected coefficients. Clearly this attack may successfully make significant changes to the video content.

3.4 The Proposed Scheme

In this section, we present our method for authenticating SVC streams. We first present an overview of the scheme, followed by the details of the authentication steps.

3.4.1 The Proposed Authentication Scheme

At a high level, the proposed authentication scheme works as follows. First, the content provider prepares the additional information needed for verification, and attaches them to the stream. Each receiver either receives the whole or a subset of the original stream, along with the corresponding authentication information. The task of substream extraction may be carried out by stream adaptation proxies belonging to the delivery network, which do not have to understand the authentication scheme. The authentication information is transparent to these proxies; it is attached to specific NAL units in a video format-compliant manner. Some packets of the stream may be lost during transmission. Unlike loss of a video packet that can be tolerated to some extent by error concealment techniques, loss of the authentication information may have a serious effect: some layers cannot be verified and thus cannot be used, although they are successfully received. We therefore need to appropriately protect the authentication information against loss. If the video is being transmitted over the Internet, where bursts of packets can be lost, it is a common practice to distribute video data over network packets in an interleaved manner [70], which changes the loss pattern from bursty to random. Relying on such packetization technique, we assume packet losses have a random pattern.

An SVC stream is a sequence of GoPs. Each GoP consists of a number of video frames, each of which belongs to a certain temporal level. Each frame, in turn, contains multiple spatial layers. A spatial layer then includes a few CGS quality layers, each one possibly partitioned into several MGS layers. Each MGS layer can be divided into multiple NAL units, to which we alternatively refer as *truncation units* or *video packets*, because these are the smallest units that can be truncated from an SVC stream. A video packet can be transmitted as more than one network packet, but without loss of generality, we assume that a video packet fits in a network packet, i.e., it is encoded at a size not exceeding the desired packet size for network transmission; otherwise, the video packet is divided into multiple video packets. According to this structure, the server prepares the authenticated video using the algorithm in Figure 3.3, which is described in the following. First, within each spatial layer of each frame, quality layers are authenticated and one hash value is computed as the *spatial layer digest*. Then, in each frame, spatial layers and their digests are authenticated and the *frame digest* is created. The next step authenticates the frames of each GoP according to their temporal levels, which results in a *GoP digest*. In the last step, the whole stream is authenticated by dividing the sequence of GoPs into blocks, and digitally signing the *block digest* computed on the block. The authentication information generated in each step is embedded into the stream. In this bottom-up approach, each of the aforementioned digests hides all scalability details of the digested unit of data. For example,

SVC_Authenticate

```
AuthenticateStream ( $GoPs[]$, $n$, $k$, $\alpha[]$ )
1. $GoPblocks$ = DivideToBlocks($GoPs$, $n$)
2. for $block \in GoPBlocks$ do
3.     $X$ = null  //contains GoP digests
4.     foreach $GoP \in block$ and $i = 0$ to $n - 1$
5.         $X = X$ || AuthenticateGoP($GoP$, $k$, $\alpha$)
6.     Embed { $X || Sign(h(X))$ } in layer $(0, 0, 0)$ of all of the GoPs in $block$
AuthenticateGoP ( $GoP$, $k$, $\alpha[]$ )
1. $T = GoP.NumTemporalLayers$
2. $n_{t \in [0, T-1]} = GoP.NumFramesAtLayer[t]$
3. $n_1 = 2$
4. $ToEmbed[]$ = MakeEmptyArray($n_T$)
5. for $t = T - 1$ to 2 do
6.     $X$ = null  //contains frame digests
7.     for $i = 0$ to $n_t - 1$
8.         $X$ = AuthenticateFrame( $GoP.FramesAt$- $Layer[t].get(i)$, $k$, $ToEmbed[i]$ ) || $X$
9.     $x[]$ = DivideToPieces( $X$, $\lceil \alpha[t] \times n_{t-1} \rceil$ )
10.    $ToEmbed[]$ = FEC_encode($x$, $n_{t-1}$)
11. $f_0, f_1 = GoP.FrameAtLayer[0], [1]$
12. $y$ = AuthenticateFrame($f_1$, $k$, $ToEmbed[1]$)
13. $x$ = AuthenticateFrame($f_0$, $k$, $ToEmbed[0]$)
14. return $\{x \,||\, y\}$
AuthenticateFrame ( $frame$, $k$, $W$ )
1. $S = frame.NumSpatialAndCGSlayers$
2. $ToEmbed[]$ = MakeEmptyArray($S$)
3. for $s = S - 1$ to 0 do
4.     $layer = frame.layer[s]$; $Q = layer.NumMGSes$
5.     $F'[]$ = MakeEmptyArray($Q$)
6.     for $q = 0$ to $Q - 1$
7.         $p[] = layer.MGS[q].Units()$
8.         $F'[q] = h(p[0]) \,||\, \ldots \,||\, h(p[\text{size}(p) - 1])$
9.         Embed $F'[q]$ in $layer.MGS[q]$ in $k$ copies
10.    $F = h(F'[0]) \,||\, \ldots \,||\, h(F'[Q-1]) \,||\, ToEmbed[s]$
11.    if $s = 0$ then
12.        Embed $\{F || W\}$ in $layer.MGS[0]$ in $k$ copies
13.        return $h(F || W)$
14.    Embed $F$ in $layer.MGS[0]$ in $k$ copies
15.    $ToEmbed$ [ $layer.HighestRef()$ ] $||= h(F)$
```

Figure 3.3: The proposed authentication scheme.

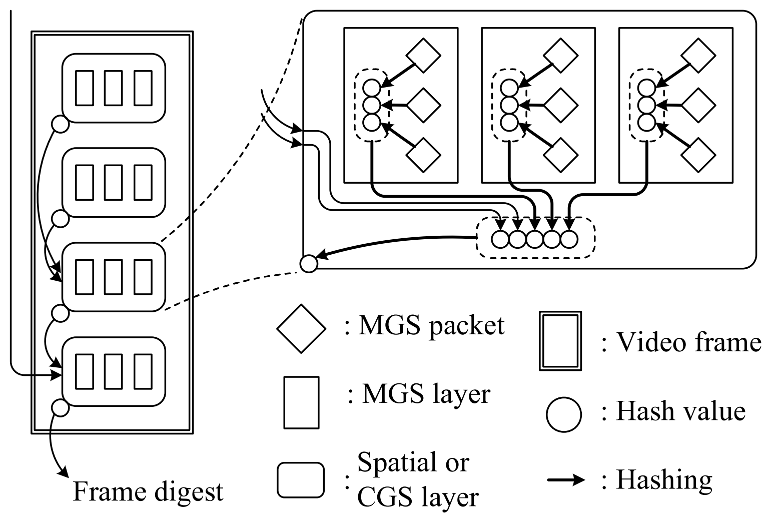


Figure 3.4: Authenticating a video frame.

a frame digest hides spatial and quality scalability, and makes the frame a transparent unit of data for the next step of authentication.

The verification process proceeds in the same way as generating the authentication information. Given a valid substream and its authentication information, a receiver recomputes spatial layer, frame, GoP, and block digests from the reconstructed video. In case of any mismatch between the recomputed digest and the digest provided by the server in the substream, the mismatching part of data, such as a video frame, is marked as unauthentic and is discarded. The remaining part of the received substream is known as authentic if and only if the digital signature of the corresponding block is successfully verified.

3.4.2 Authentication of Quality Layers

Quality scalability can be provided in SVC by encoding one or more CGS layers, and partitioning each CGS layer into multiple MGS layers. As discussed earlier, CGS layers are encoded the same way as spatial layers, and follow the same dependency structure. We therefore treat CGS layers as spatial layers; when dealing with authentication of quality scalability, we only consider MGS layers within a spatial or CGS layer. Note that if a video is not encoded with Medium-Grained Scalability, without loss of generality, we consider each spatial/CGS layer as if having only one MGS layer. MGS quality layers in SVC, unlike previous scalable videos, can be extracted in many possible ways. Moreover, MGS layers are no more atomic units. Rather, a single MGS layer can be truncated at packet level [69], and the H.264/SVC standard does not dictate any specific quality packet extraction process. Therefore, we form the authentication information of MGS layers/packets of each spatial/CGS layer in a two-level hierarchy as shown in Figure 3.4 and lines 4–10 of the "AuthenticateFrame" function in the pseudocode (Figure 3.3).

Let Q be the number of MGS layers, $MGS[q]$ $(0 \leq q \leq Q-1)$ be the MGS layers themselves, P_q be the number of video packets constituting the q-th MGS layer, and $p_{q,0}, \ldots, p_{q,P_q-1}$ be the packets themselves. First, packets of each MGS layer $MGS[q]$ are hashed using a secure hash function $h(\cdot)$, and their hashes are concatenated as $F'_q = h(p_{q,0})|| \ldots ||h(p_{q,P_q-1})$ (see Figure 3.4 and line 8 of the code). Then, F'_q values are hashed and concatenated again as $F = h(F'_0)|| \ldots ||h(F'_{Q-1})||H_{x_0}||H_{x_1}|| \ldots$, where the values (typically one or very few) H_{x_i} are the spatial layer digests of some higher layers (the *ToEmbed* array in line 10 of the code), as introduced shortly. Moreover, to the lowest spatial layer could possibly be added, as one of the aforementioned H_{x_i} values, a part of the authentication information of the next temporal level (string W in the code). This is depicted in Figure 3.4 as the arrow going from outside the frame to the lowest spatial layer. The *spatial layer digest* is obtained as $H = h(F||W)$. Each F'_q is attached to the MGS layer $MGS[q]$ so that any subset of packets of this MGS layer can still be verified without requiring the whole set of packets of the layer. Similarly, F is attached to the corresponding spatial/CGS layer. This enables a receiver to reconstruct the spatial layer digest H, and all required intermediary values, even if some layers and packets are not received at all.

3.4.3 Authentication of a Video Frame

In SVC spatial (and CGS) scalability, the dependency structure among spatial layers of a frame, i.e., intra-frame dependency, does not have to be linear, as it was in previous scalable videos. Rather, it is in general a Directed Acyclic Graph (DAG), where for example, layer 5 does not necessarily depend on layer 4, but instead it may use layers 1 and 3 as reference. We thus attach each spatial layer digest H_s to its highest reference layer. This is shown in Figure 3.4, and lines 10 and 15 of the pseudocode in Figure 3.3. It is possible, though unlikely, that a spatial enhancement layer s is not dependent on any lower layer in the same frame; it is predicted only from the s-th spatial layer of other frames. In this case, we attach the layer digest H_s to the s-th layer of the closest reference frame (not shown in Figures 3.4 and 3.3). The digest of the lowest spatial layer of a video frame represents the *frame digest*.

To protect the authentication information of quality and spatial layers against loss, we need to add redundancy. The common technique for this purpose is the use of Forward Error Correction (FEC) codes. However, since there can be several quality and spatial layers, the computational cost of performing many FEC operations per each frame can be too high. Therefore, we replicate the authentication information of each MGS/spatial layer in two or more packets, rather than FEC-coding this information. The number of copies of the authentication packet, denoted by k in Figure 3.3, can balance a tradeoff between loss tolerance and the communication overhead. We will show in the evaluation section that only 2 copies are enough for resisting against typical loss ratios.

3.4.4 Authentication of a GoP

We take care of temporal scalability at frame level using frame digests provided in the previous step. In a temporally scaled SVC substream, there exist all frames of temporal layers below (excluding) a certain level, T, and possibly a fraction of frames at level T. Thus, to be able to verify any valid substream, we embed the digests of frames of each temporal layer in the frames of the lower temporal layer according to Figure 3.5 and the "AuthenticateGoP" function in Figure 3.3; GoPs with non-dyadic structure can be authenticated in a similar manner. In this scheme, frame digests of temporal level t are first concatenated. Denoting the number of frames in level t by n_t, the concatenation of frame digests of temporal level t is divided into $k_t = \lceil \alpha[t] \times n_{t-1} \rceil$ pieces (line 9), FEC-coded into n_{t-1} pieces, and distributed over the n_{t-1} frames of level $t-1$. In this way, adaptation of the video to any frame rate will not affect our authentication scheme. At the receiver side, authentication information of any k_t out of n_{t-1} frames of temporal level $t-1$ leads to successful verification of all frames at level t. Since $k_t = \lceil \alpha[t] \times n_{t-1} \rceil$, loss of the authentication information of up to $(1-\alpha_t)\ n_{t-1}$ frames of the n_{t-1} frames at temporal layer $t-1$ can be tolerated.

Figure 3.5 also shows how the *GoP digest* is obtained, whose authenticity is essential for the authentication of any subset of the GoP. Note that, for different temporal levels, different robustness to loss can be provided, because lower temporal levels are more important as they are used for prediction of more frames. Assessing the importance of frames in SVC prediction structure, and unequally protecting them against loss is an orthogonal work to ours, and we leave that to, for example, [71, 72]. Nevertheless, this practice is more effective than FEC-coding the whole set of authentication information equally, as done for scalable authentication schemes in [65] and [13]. As a case in point, FEC-coding some

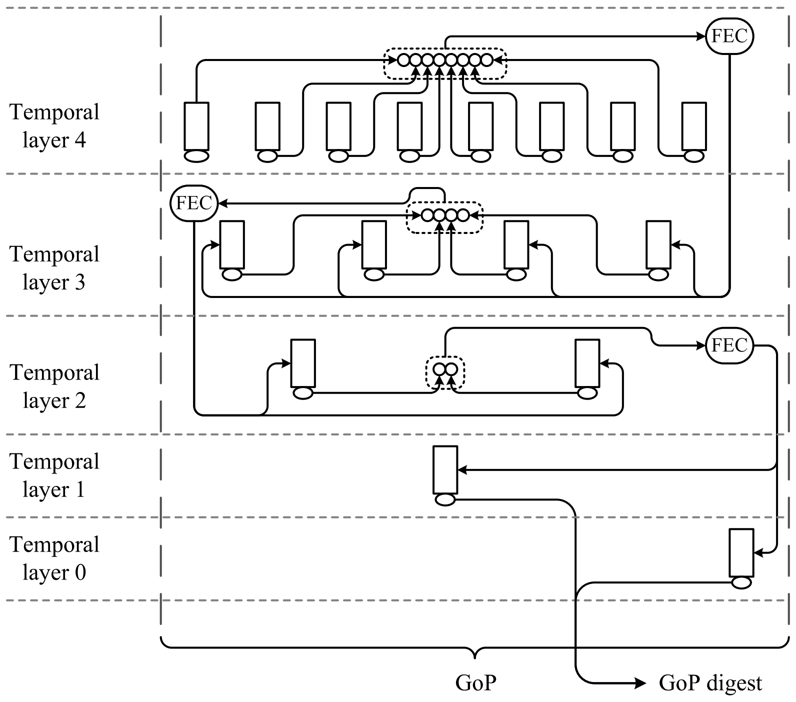


Figure 3.5: Authenticating frames of a GoP.

information into two parts is actually equivalent to replicating the information in two copies. Thus, for temporal layers 2 and below, the FEC coding operation consists of replicating the concatenation of frame digests of the higher temporal layer. Since there are only a few FEC operations performed per each GoP, the computational cost is kept low.

3.4.5 Authentication of a Sequence of GoPs

No GoP can be dropped from the stream for adaptation purposes. Thus, a sequence of GoPs can in general be thought of as a stream of data packets. Accordingly, we first consider to authenticate the sequence of GoPs by applying one of the data stream authentication techniques proposed in the literature. To authenticate a stream of data packets, as we reviewed in Section 2.4, the common practice, e.g., [41, 44], is to divide packets of a stream into blocks of size n packets, and designate one digital signature for each block. This amortizes the signature size and the computation cost of signature verification over several packets. For preparing the authentication information of a block of GoPs, we follow the same approach.

For distributing the authentication information of a block in its packets, various methods are proposed aiming to best resists against bursts of packet loss. The cost of this is some delay at the sender/receiver side for generating/receiving a block, which is especially important for live streaming, and some buffering requirement for receivers to keep almost a complete block before being able to verify any of the packets. However, a sequence of GoPs has different characteristics than a sequence of data packets: it has a low rate, each GoP is very large compared to a packet, and GoPs are not likely to be lost in bursts, since bursts of loss have short periods [60]. Taking advantage of these considerations, we authenticate each block of GoPs as described in the "AuthenticateStream" function of Figure 3.3. Compared to applying classic packet stream authentication techniques for authenticating the sequence of GoPs, this method has the following benefits: (i) it is more robust against loss, since receiving the authentication information of any GoP is sufficient for verifying the whole block, (ii) it incurs lower delay, because a receiver does not have to wait for receiving almost a complete block, (iii) it requires receivers to buffer one or very few GoPs, since any successfully received GoP authentication information suffices for verifying the GoPs, and (iv) it imposes only a small communication overhead ($<$ 1 KB per second) because the information attached to each GoP consists of a few hash values and a digital signature, and the sequence of GoPs has a low rate (very few GoPs per second).

3.5 Analysis and Security of the Proposed Scheme

3.5.1 Security Analysis

We prove that the proposed scheme authenticates any valid substream extracted from the original stream, provided that the underlying cryptographic primitives (hash function and digital signature) are secure on their own. The scheme enables a client to assure that the received content has gone under no manipulation, such as changes in the contents of any video frame, frame insertion, frame reordering, frame removal, or any other type of tampering—frame removals due to frame rate reduction are legitimate as long as they are compliant to the SVC standard, which cannot be used for malicious purposes such as cropping a number of consecutive frames.

Theorem 1 *The proposed scheme ensures the authenticity of any substream extracted from the original SVC stream.*

Proof. Recall the hierarchical structure of an SVC video stream, whose levels consist of GoPs, temporal layers, frames, spatial/CGS layers, MGS quality layers, and finally, video packets. We prove the authenticity of any valid substream in a bottom-up manner in this structure. We first show that the authenticity of any valid subset of quality layers can be successfully verified if the corresponding spatial layer digest is authentic. The procedure continues similarly for the digests of spatial layers, frames, GoPs, and GoP blocks. Finally, successful verification of the digital signature of a GoP block is shown to be the *necessary and sufficient* condition for authenticity of any valid substream, which proves the theorem.

Step 1: Authenticity of quality layers. We first analyze the deepest level of the scalability hierarchy, which corresponds to quality layers. We prove that any valid subset of video packets of a spatial/CGS layer, given the authentication information of the packets and the digest of the corresponding spatial/CGS layer, are authentic iff the spatial/CGS layer digest is authentic. The forward direction of the statement, i.e., no manipulated/inserted video packet is accepted, is proven as follows. Since the spatial/CGS layer digest $h(F)$ (where F is obtained in line 10 of the code) is authentic and $h(\cdot)$ is a collision-free hash function, the concatenation $F = h(F'[0])||\ldots||h(F'[Q-1])||\mathit{ToEmbed}[s]$ is also authentic. This proves the authenticity of all $h(F'[i])$ values, which are present in the received substream since they are included in the authentication information of the spatial/CGS layer; they enable a receiver to reconstruct F even if it has received no packet from some of the quality layers. For each MGS layer q from which at least one packet is received, the attached $F'[q]$ value can be verified since $h(F'[q])$ is authentic, meaning that no $F'[q]$ value can be forged or inserted. Since an MGS layer q's $F'[q]$ equals $h(p[0])||\ldots||h(p[\mathit{size}(p)-1])$ (line 8 of Fig. 3.3), the authenticity of $F'[q]$ proves the authenticity of $h(p[i])$ values, and accordingly, the packets $p[i]$ that exist in the substream. Any change to the contents of a packet will result in a change in the corresponding $h(p[i])$, $F'[q]$, $h(F'[q])$, F, and $h(F)$ values, and no change to a packet $p[i]$ can preserve these values since $h(\cdot)$ is collision-free. Moreover, no video packet can be inserted, as the integrity of $F'[q] = h(p[0])||\ldots||h(p[\mathit{size}(p)-1])$ is already proven. This shows that no content other than a subset of authentic packets can pass the verification process. The backward direction of the statement, i.e., no original subset of packets is rejected, is clear. First, one can reconstruct the spatial/CGS layer digest out of any valid subset of quality packets. Moreover, out of an authentic subset of packets the same $h(F)$ value as the one provided by the content provider is calculated by a receiver. Hence, any subset of packets will pass the verification iff it is authentic.

Step 2: Authenticity of spatial/CGS layers. Having authenticated each of the received or partially received spatial/CGS layers, we now show that in a valid subset of spatial/CGS layers of a frame, each layer that has a path of digests to the base layer is authentic iff the frame digest is authentic. Each layer that does not have this path, which in turn has a path to another frame digest, is authentic iff the digest of that frame is authentic. This statement is obvious for the base layer, since a frame digest is actually the digest of the base layer along with its attached authentication information. Any other spatial/CGS layer has its digest embedded in a lower layer, which makes the authenticity of the lower layer necessary and sufficient for the authenticity of the higher one. Hence, this relationship is created between frame digests and all of the layers, because from each layer there is a path to the base layer of the same frame; or in the unlikely case that a layer is not predicted from any lower layer, there is a path from it to the base layer of another frame. Hence, the authenticity of frame digests is necessary and sufficient for authenticity of any valid subset of spatial/CGS layers.

Step 3: Authenticity of video frames. We now show that the authenticity of the digest of a GoP is necessary and sufficient for the authenticity of the digests of its frames. Recall that a valid subset of frames consists of all video frames of temporal layers $1, 2, \ldots, T-1$, possibly a fraction of frames at temporal level T, and no frame from levels $T+1$ and higher. In the GoP authentication procedure, if a sufficient number of frames of level t are received by a client, the concatenation of frame digests of level $t+1$ can be verified, which assures for any subset of frame digests of level $t+1$ that all digest have integrity and no frame (with its digest) is inserted. Therefore, the authenticity of the frame digests of each temporal level is necessary and sufficient for the authenticity of those in the next temporal level. This enables the authenticity of the GoP digest to propagate from the temporal base layer to all of the received temporal layers and assures the authenticity of all frame digests.

Step 4: Authenticity of GoPs. A GoP block digest, which is digitally signed, is nothing but a hash value over the digests of the GoPs of the block. This clearly shows the bidirectional dependency between the authenticity of GoP digests and that of the GoP block digest; it prevents any changes to a GoP digest or insertion/removal of a GoP.

According to the above steps, the correctness of Theorem 1 can now be readily proven as follows. The stream consists of independently signed blocks of GoPs. According to steps 4 through 1, successful verification of the digital signature of a GoP block is *necessary and sufficient* for the authenticity of all packets of an extracted substream. □

3.5.2 Complexity Analysis

Computation cost. We calculate the computation cost in terms of the number of hash computations, FEC coding operations, and digital signature generations/verifications per each GoP block. Since there is one hash computation per each GoP, frame, spatial/CGS layer, MGS layer, and truncation unit, the total number of hash operations is $n+F+S+Q+P$, where F, S, Q, and P represent the total number of frames, spatial layers, quality layers, and video packets in a block of n GoPs. Since a hash operation can be performed very fast compared to decoding of a video packet, we can practically ignore the computation cost of the above hashes. The total number of FEC operations per block of n GoPs equals the number temporal layers times n, which is typically a few per second. This cost is negligible too, as there are fast FEC coding algorithms to be employed. For example, Tornado codes for FEC coding use only XOR operations and operate in linear time of the input block. The dominant computation cost is that of digital signature operations, the number of which in our scheme is one per n GoPs. This cost is still low in our scheme as we show in the evaluation section.

Communication overhead. We denote by s_{hash} the size of a hash, by s_{sig} the size of a digital signature, by k the number of copies of authentication information of quality and spatial/CGS layers, and by α the FEC factor for authenticating temporal layers, i.e., the fraction of pieces enough for re-constructing the original data. The communication overhead of our scheme for each block of n GoPs, C, is as follows:

$$C = (k \times (P + Q + S) + F/\alpha + n) \times s_{hash} + s_{sig} \tag{3.1}$$

This communication overhead can become non-negligible for highly flexible scalable streams that provide many possibilities for extracting substreams. We propose an algorithm to reduce this communication overhead in the next section.

3.6 Reducing the Overhead

The amount of authentication information that needs to be added to an SVC stream can be non-negligible if the stream is providing a high flexibility, i.e., when there is a large number of truncation points. We refer to this information as the communication overhead of the scheme. Note that the non-negligible communication overhead is not specific to our scheme and will be suffered from by other authentication schemes as well. This is because a hash value is needed for each truncatable unit of data, and the number of these units grows with flexibility of scalable streams.

In order to reduce the communication overhead, we compute one hash value for a *group of truncation units* rather than for each unit. Note that the atomic unit of authentication would then be a group of units, i.e., partial groups cannot be authenticated. That is because unverified video packets of a partially received group have to be discarded, as shown in Section 3.3.2. Accordingly, by aggregating truncation units into groups, on one hand we reduce the communication overhead of hashes, and on the other hand, we may also reduce the flexibility of the stream. That is, some receivers may receive a smaller number of layers than the number of layers they would have received if there was no grouping. In this section, we propose an optimal algorithm for grouping the truncation units in order to minimize this twofold overhead. The application of this algorithm is not limited to our authentication scheme, and it can be used with other scalable video authentication techniques in the literature, such as those for traditional scalable streams, e.g., [15, 61], for optimizing their communication overhead. Thus, this overhead reduction algorithm can be of interest in its own right.

Since only a complete group of units can be authenticated, in order for the grouping process to perform well, the selection of quality truncation units in the substream extraction process should not be arbitrary. Otherwise, it is possible that in a received substream, for example, all groups are received partially, none of which is then verifiable. Thus, we can embed (during encoding) the quality extraction information inside the stream so that stream adaptation proxies follow the same extraction procedure. The H.264/SVC standard provides specific means for signalling this information in the streams [69]. This enables us to assume an ordering on truncation units, and cases such as the aforementioned example will not happen. That is, at most one group of units can be received partially. Note that this assumption does not restrict the flexibility of SVC quality scalable streams, as the discarding of quality packets and layers can still be non-cumulative. However, grouping units and not allowing the use of partial groups limits this flexibility, since it reduces the number of possible truncation points. Clearly, there is a tradeoff between the flexibility of streams and the communication overhead imposed by the authentication scheme, and this tradeoff is controlled by the size of truncation unit groups.

Our grouping algorithm works on a frame basis and divides the truncation units of each frame into a number of groups. Note that the number of truncation units of a frame can grow to hundreds as we show for actual videos in the evaluation section. The algorithm uses the communication overhead as a cost function and employs dynamic programming to determine the grouping of truncation units that minimizes this cost.

Suppose there are n truncation units in a frame f, and denote them by $u_1, u_2, \ldots, u_n$. These units are to be partitioned into a number of, say h, groups as follows:

$$\underbrace{u_1 \ \ldots \ u_{x_1}}_{x_1} \mid \underbrace{u_{x_1+1} \ \ldots \ u_{x_1+x_2}}_{x_2} \mid \ldots \mid \underbrace{u_{x_1+\cdots+x_{h-1}+1} \ \ldots \ u_n}_{x_h}.$$

The algorithm finds the optimal number h^* of (non-empty) groups, and divides the n units into h^* groups. The cost of grouping $u_i, \ldots, u_j$ of frame f is $c_f(i, j)$, which consists of the streaming bitrate that users lose when we omit truncation points at $u_i, \ldots, u_{j-1}$, as well as the overhead of an s-bit hash value that will be designated to the group. For calculation of the overall streaming bitrate that users lose by grouping u_i through u_j, we can also take into account the distribution of user bandwidths, leading to a more accurate calculation of $c_f(i, j)$. Calculation of $c_f(i, j)$ based on the truncation units of a frame and user bandwidth distribution is discussed at the end of this section. The cost of multiple groups equals the sum of the group costs. It could be possible, although unlikely, for some highest-layer truncation units not to be in any being-hashed group. Let $c'_f(i)$ denote the cost for not having $u_i, \ldots, u_n$ in any hashed group, i.e., these units are preferred to be ignored.

The proposed dynamic programming algorithm is based on solving subproblems (h, l), which represent the grouping of the first l units into h groups $(1 \le h \le l \le n)$ where the h-th group exactly ends at the l-th unit. The minimum cost for the (h, l) subproblem is kept in a matrix $A_f[h, l]$ for frame f. For the first row of the matrix A_f, which refers to the case where only one hash value is to be calculated for the frame, $A_f[1, l]$ indicates that units $\{u_1, \ldots, u_l\}$ will constitute the only group. The rest of the rows of the matrix A_f are calculated as follows:

$$A_f[h, l] = \min \begin{cases} A_f[h-1, l-1] + c_f(l, l), \\ A_f[h-1, l-2] + c_f(l-1, l), \\ \quad \vdots \\ A_f[h-1, h-1] + c_f(h, l). \end{cases} \tag{3.2}$$

In Eq. (3.2), the i-th row $(1 \le i \le l - h + 1)$ represent the case where the h-th group consists of the $l-i+1$-st through the l-th units. Thus, the cost equals the cost of having the previous $l-i$ units in $h-1$ groups, which is $A_f[h-1, l-i]$, as well as the cost of grouping the $l-i+1$-st through the l-th units together, which is $c_f(l-i+1, l)$. To maintain how each of the subproblem solutions is constructed, we keep another $n \times n$ matrix B_f where $B_f[h, l]$ represents the number of units in the h-th group when grouping l units in h groups, i.e., the row index in Eq. (3.2) that led to the minimum cost. Therefore, the solution to subproblem (h, l) consists of $B_f[h, l]$ units in the h-th group, $B_f[\ h-1, l-B_f[h, l]\ ]$ units in the $h-1$-st group, and so on.

Having filled the cost matrix A_f with the minimum cost of all valid subproblems, we now calculate the minimum cost $\hat{A}_f[h]$ of the optimal solution for grouping all units into h groups. If we were sure that all units belong to some group, i.e., no unit is decided to be ignored, $\hat{A}_f[h]$ would have been equal to $A_f[h, l]$. However, note that it is possible, though unlikely, that the optimal solution leaves some units $u_{x+1}, u_{x+2}, \ldots, u_n$ out of any group, e.g., hashing those truncation units does not worth its communication overhead. Thus, $\hat{A}_f[h]$ is calculated as:

$$\hat{A}_f[h] = \min\{A_f[h, x] + c'_f(x+1) \mid h \leq x \leq n\} \tag{3.3}$$

To obtain the final optimal solution, we first determine the best number of groups h^* as $h^* = \text{argmax}_h\{A'[h]\}$. Then, to construct those h^* groups, we first note that the number of truncation units considered in those h^* groups, denoted by x^* $(1 \leq x^* \leq n)$, is actually the x value that minimized Eq. (3.3). In the optimal solution, the number of units in the i-th group $(1 \leq i \leq h^*)$, denoted by $X[i]$, is obtained from $X[h^*]$ to $X[1]$ as follows. Let p be a pointer representing the number of units in the remaining groups, and thus initially assigned as $p = x^*$. According to the construction of the matrix B, we have $X[h^*] = B[h^*, p]$. Having put $X[h^*]$ units in the h^*-th group, we update the pointer p as $p = p - X[h^*]$ and obtain the number of units in the second last group as $X[h^*-1] = B[h^*-1, p]$. Updating p as $p = p - X[h^*-1]$, we get $X[h^*-2] = B[h^*-2, p]$, and similarly are obtained the rest of the $X[i]$ values. The values h^* and $X[i]$ $(1 \leq i \leq h^*)$ form the optimal grouping solution.

The following theorem shows the optimality and complexity of the proposed algorithm.

Theorem 2 *The proposed dynamic programming solution for aggregating truncation units of a frame into groups finds the minimum-cost grouping. It has a memory requirement of $O(n_{max}^2)$, where n_{max} is the maximum number of truncation units in a frame, and it has a time complexity of $O(n_{max}^3)$.*

Proof. The key point of the proof is the optimality of matrix A_f that keeps the minimum costs of subproblems and is calculated as Eq. 3.2. We show this optimality by showing that the problem exhibits the *optimal substructure* property: an optimal solution to the problem, which is h^* optimally formed groups of $X[1], X[2], \ldots, X[h^*]$ truncation units, contains within it optimal solutions to subproblems, which is optimal formation of the first h $(1 \leq h \leq h^*)$ groups using the first l $(h \leq l \leq n)$ units. This is true because in the final optimal solution, if the grouping of the first $l = \sum_{i=1}^{h} X[i]$ units in the first h groups is not optimal, we simply replace this part with an optimal sub-solution and obtain a smaller cost, leading to a contradiction. Given the optimality of the matrix A_f, the optimality of $\hat{A}_f[h]$ values (Eq. (3.3)) immediately follows.

The memory required by the algorithm for a frame f is that of two $n \times n$ matrices, A_f and B_f, and one vector $\hat{A}_f$ of length n. Thus, the memory required is of $O(n_{max}^2)$, where n_{max} is the maximum number of truncation units in a frame and hardly reaches 1000. The required memory is thus a small amount.

The algorithm calculates only the upper triangular half of matrices A_f and B_f, i.e., indices $[i, j]$ where $i \leq j$. For each $A_f[i, j]$ and $B_f[i, j]$ together, $j - i + 1$ comparisons are performed. Hence, the running time in terms of the total number of comparisons Ω_f for a frame f is:

$$\Omega_f = \sum_{i=1}^{n} \sum_{j=i}^{n} (j - i + 1) = \frac{1}{6} n(n+1)(2n-5) + n^2 + n. \tag{3.4}$$

This number of iterations makes the running time of the algorithm $O(n_{max}^3)$, though its constant factor is considerably small (1/3).

□

We have run our algorithm on a commodity PC for high bitrate video streams (> 10 Mbps) with 500-byte truncation units, which makes 30 to 300 units per frame, and it could easily perform the grouping faster than the frame rate of the stream.

3.6.1 The Cost Function

This section describes the details of computing cost values $c_f(i, j)$, which is the cost of grouping the i-th through the j-th truncation units of frame f together. This is done based on the bitrate of the video up to each truncation unit, and optionally, the distribution of users bandwidths.

The cost $c_f(i,j)$ consists of two parts: the loss of truncation points at truncation units $u_i, \ldots, u_{j-1}$, as well as an s-bit overhead of the hash value being designated to the group of units. To have an accurate calculation of these costs for a frame f, we need the time duration $d(f)$ it takes for f to be downloaded, which may not be equal among different frames. $d(f)$ is calculated based on the bitrate of the video and the sizes of other frames as we see shortly. Let $b(u_i) = size(u_i)/d(f)$ denote the bitrate of a unit u_i, $b_i = \sum_{x=1}^{i} b(u_x)$ the bitrate of the frame under process when it contains up to unit u_i, and $\hat{s}_f = s/d(f)$ the bitrate overhead of a hash value for frame f. The cost function, which is reflecting the communication overhead, can be thought of as the bandwidth waste of all clients, that is, the difference between the bitrate of the authenticated video that a client will receive and the bitrate of the video if there was no authentication information attached and no grouping performed. To calculate these costs, we can also take into account a priori knowledge about the streaming scenario in terms of the distribution of client bandwidths $\mathcal{Z}$, leading to a more efficient overall cost minimization. Clearly, a uniform distribution can be assumed if no such priori knowledge can be obtained. The cost $c_f(i,j)$ for a group of units consists of the cost of a hash value and that of making clients with bandwidths in $[b_i, b_j)$ receive a video bitrate of b_{i-1} rather than b_i, b_{i+1}, $\ldots$, or b_{j-1}. The cost $c_f(i,j)$ is calculated as:

$$c_f(i,j) = \hat{s} + \sum_{x=i}^{j-1} \Big((b_x - b_{i-1}) \times \int_{b_x}^{b_{x+1}} Pr(\mathcal{Z} = z)\ dz \Big) \tag{3.5}$$

Similarly, the cost $c'_f(i)$ of not having units $u_i, \ldots, u_n$ in any hashed group is calculated as:

$$c'_f(i) = \sum_{x=i}^{n} \Big((b_x - b_{i-1}) \times \int_{b_x}^{b_{x+1}} Pr(\mathcal{Z} = z)\ dz \Big) \tag{3.6}$$

Calculation of the cost values $c_f(i,j)$ impacts the running time of the algorithm negligibly, since they can be calculated for each frame prior to running the grouping algorithm for the frame. For each frame f, a lookup matrix C_f is created, which is discarded after the vector $\hat{A}_f$ of the frame is obtained. The matrix is constructed as $C_f[i,j] = c_f(i,j)$. Thus, during the calculation of A_f and $\hat{A}_f$ in the grouping algorithm, each $c_f(i,j)$ is immediately retrieved. Calculation of the matrix C_f can be done efficiently due to the progressive nature of the cells. That is, having $C_f[i,i] = c_f(i,i) = \hat{s}$, for $1 \le i < j \le n$ we have:

$$C_f[i,j] = C_f[i,j-1] + (b_{j-1} - b_{i-1}) \times \int_{b_{j-1}}^{b_j} Pr(\mathcal{Z} = z)\ dz \tag{3.7}$$

Hence, each $C[i, j]$ is calculated in $O(1)$ and calculation of the entire matrix takes $O(n^2)$, which can be neglected compared to other calculations of the overhead reduction algorithm. The memory required by this matrix is also of $O(n^2)$, which is similarly negligible.

The last step is to compute $d(f)$, the expected time it takes for frame f to be downloaded. Denoting the frame rate of the video by α, $d(f)$ simply equals $1/\alpha$ for a Constant Bitrate (CBR) video. For a Variable Bitrate (VBR) video, however, frames do not have the same size and thus are not expected to take the same time to be downloaded. Therefore, we calculate $d(f)$ values as follows. Our algorithm works on a basis of a few (w) GoPs, and no information about frames of further GoPs is known—accordingly, for live streaming scenarios w has to be small, preferably $w = 1$, in order to impose a very short delay. Let the size of the given w GoPs be S bits and their playback duration be D seconds, making the bitrate of the corresponding sequence of GoPs be S/D bps. In order for a client to keep the download rate of the video constantly equal to its bandwidth, the download rate of each frame f should also be S/D bps, since the download rate does frequently vary from frame to frame. Hence:

$$\frac{\sum_{i=1}^{n} size(u_i)}{d(f)} = \frac{S}{D} \Rightarrow d(f) = \frac{D}{S} \sum_{i=1}^{n} size(u_i) \tag{3.8}$$

Clearly, this is done in $O(1)$ for each truncation unit.

3.6.2 The Impact of Packet Losses on the Overhead Reduction Algorithm

The proposed algorithm for minimizing the communication overhead may be vulnerable to packet losses. When gathering a number of truncation units in a group and designating one hash value for the group, loss of any of the units will result in unverifiability of the rest of the units in the group. Nevertheless, the proposed algorithm can gain over 50% overhead reduction when streaming over reliable channels or channels with low loss ratios, which is the typical case in today's Internet. For example, in a large-scale measurement study [73], over 85% of traces experienced a loss ratio of below 1%, and 99% of traces experienced a loss of less than 10%. If the loss ratio is significant ($> 10\%$) the algorithm will have a tendency to form smaller groups. Consequently, the overhead saving by the algorithm reduces when loss ratio increases. Clearly, the algorithm never results in a higher overhead than authentication with no grouping.

Recall that with simple authentication with no grouping, we send the authentication information of quality layers in two copies for lossy transmission scenarios. For grouping the video packets to reduce the overhead, we send one copy of the hashes of units as it is, and apply the optimal grouping on the replication copies. In this case, Eq. (3.5) is updated as follows to capture the *expected* cost of losing a unit, which is the loss of other truncation units:

$$\begin{aligned} c_f(i,j) \;=\; \hat{s} \;+\; \sum_{x=i}^{j-1} \Big((b_x - b_{i-1}) \times \int_{b_x}^{b_{x+1}} Pr(\mathcal{Z}=z)\; dz \Big) + \\ \rho \Big(1 - (1-\rho)^{j-i+1} \Big) (b_j - b_{i-1}) \int_{b_j}^{+\infty} Pr(\mathcal{Z}=z)\; dz \end{aligned} \tag{3.9}$$

where ρ is the expected loss ratio, and thus the probability that at least one unit from the group is lost is $\Big(1-(1-\rho)^{j-i+1}\Big)$, and the probability that the hash values of the units of the group (carried in the same authentication NAL unit) gets lost is ρ.

3.7 Performance Evaluation

We use trace-based simulations to evaluate the performance of our scheme in terms of computation cost, delay, buffer requirements for receivers, loss tolerance, and communication overhead. As described in Section 3.2, we are not aware of other schemes in the literature designed for end-to-end authentication of scalable video streams that support the flexible, three-dimensional, scalability. Previous authentication schemes are not applicable to such streams, since they cannot authenticate all their possible substreams. Hence, quantitatively comparing our scheme against them is not possible.

3.7.1 Simulation Setup

We simulate the transmission of H.264/SVC scalable video streams over a channel with packet losses. The packet size is 1 KB. We consider three diverse videos from the Joint Video Team (JVT) test sequence set, namely "Crew", "Soccer", and "Harbour", and encode them using the H.264/SVC reference software, called JSVM. Each encoded stream consists of 4 temporal layers (GoP size 8) and 2 spatial layers providing CIF and 4CIF resolutions. In each of the streams, both spatial representations provide a high quality of approximately 40 dB in terms of Y-PSNR. The considered streams are quite diverse in their content, resulting in different bitrates of 1.5 Mbps, 2.1 Mbps, and 3.7 Mbps for the CIF representation of the stream, and 5.5 Mbps, 8.6 Mbps, and 12.3 Mbps for the 4CIF representation. Each spatial

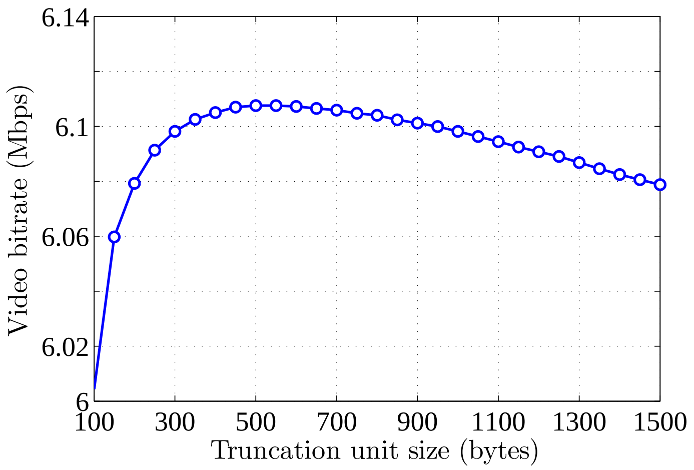
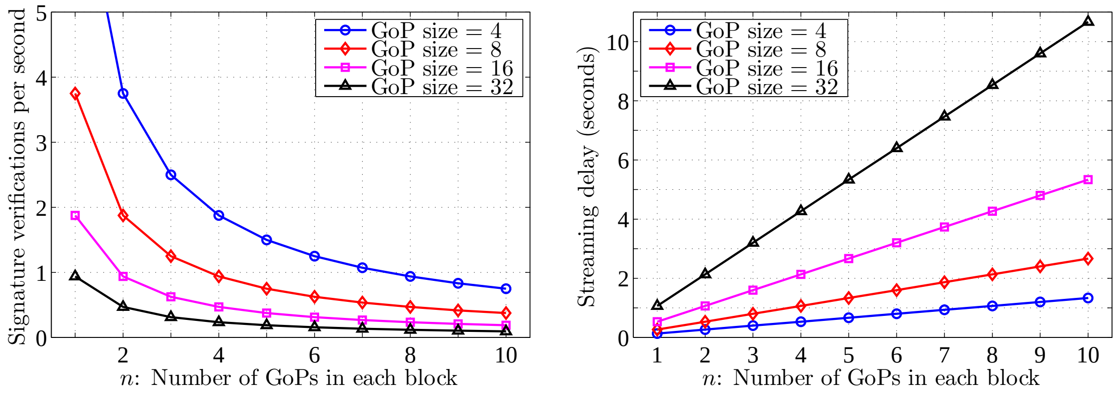


Figure 3.6: Average bitrate extracted from the video versus the size of truncation units.

layer contains 2 CGS quality layers. The CGS base layer of each spatial layer provides the minimum quality for that resolution, and has a bitrate between 85 kbps and 140 kbps (at full frame rate) for the considered streams. The CGS enhancement layer of the first and the second spatial layer is divided into 4 and 5 MGS layers, respectively. Each MGS layer in turn is divided into multiple truncation units.

To obtain the size of a truncation unit, we performed a local search in the tradeoff between having small truncation units, which means finer granularity but higher NAL header overhead, and having large truncation units, which incurs lower NAL header overhead but provides coarser granularity.

Figure 3.6 illustrates this tradeoff as the average bitrate that can be extracted from the video versus the unit size for the "soccer" stream; the other two sequences demonstrate similar results. In this tradeoff, for smaller truncation unit sizes, the overhead of NAL headers becomes significant and makes users receive video data at lower bitrates. For higher unit sizes, the granularity of the scalable stream is coarsened. Consequently, the bitrates of the substreams that users receive is no longer so close to their demanded download rates. According to Figure 3.6, we chose 500 bytes as the unit size, which means that two such units can fit in a single packet. A high degree of flexibility is provided by the considered streams since a subset of 500-byte packets of any MGS quality layer can be discarded. We determined the 500-byte truncation units of each video frame using our own utilities for parsing SVC streams—this was needed to be done as a post-processing of the stream that is created by



(a) Rate of signature verifications. (b) Streaming delay.

Figure 3.7: Computation cost, delay, and loss resilience of the proposed authentication scheme.

the JSVM encoder, because the JSVM encoder often embeds all video data of an MGS layer in one (possibly big) NAL unit. We then add the authentication information to the layers and transfer the authenticated streams to the receiver through a loss simulator. The loss simulator drops a number of packets according to the desired loss ratio. We employ SHA-1 as the hash function (20-byte hashes), and RSA as the digital signature scheme (128-byte signatures) due to its inexpensive verification, which is an advantage for accommodating limited-capability receivers.

3.7.2 Simulation Results

Computation Cost. This is the most important performance factor of an authentication scheme. If some receivers cannot afford the computations needed by the scheme, they cannot verify the video at all; we assume the server is powerful enough for providing the authenticated stream in real-time. The dominant operation in the verification process is verifying the digital signatures, as discussed in Section 3.5.2. Fig. 3.7(a) depicts the number of signature verifications needed per second for different values of n (the number of GoPs in a signed block). The value of n can balance a tradeoff between delay and computation cost, since the content provider needs to generate a complete block of n GoPs before being able to transmit any of them. Assuming that one to two signature verifications per second are easily affordable by nowadays limited-capability video playback devices, as reviewed in

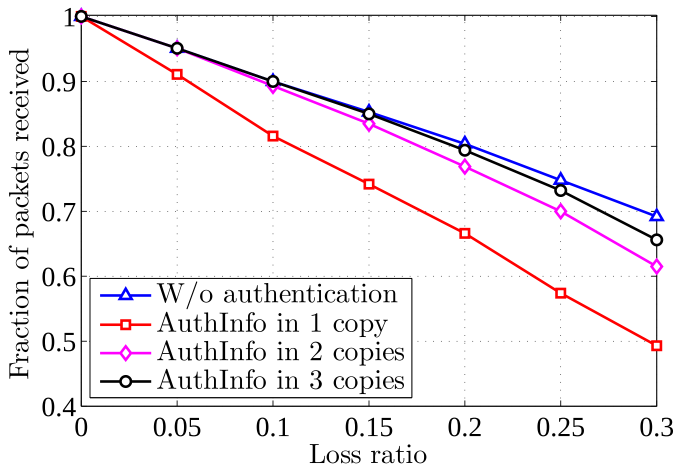
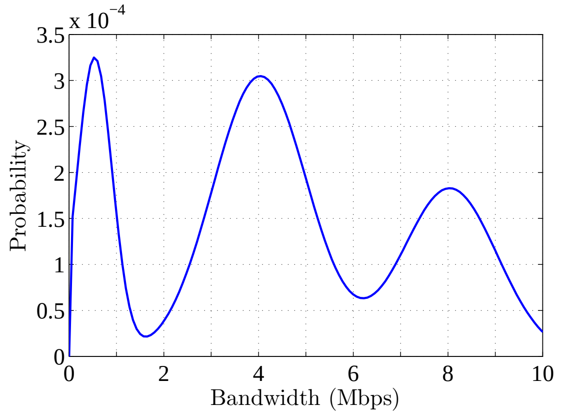


Figure 3.8: Fraction of received packets that can be verified.

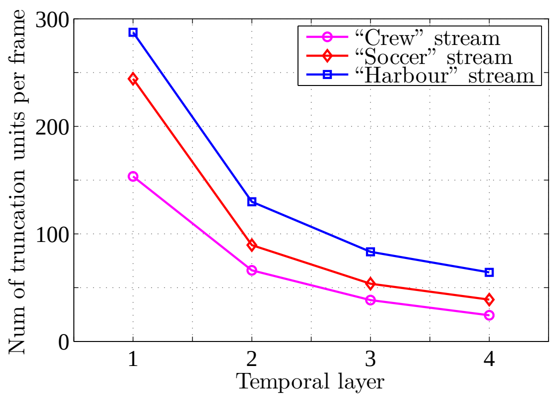
Section 2.5.1, Fig. 3.7(a) shows that gathering only $n = 5$ GoPs in each block suffices for having the authentication operations affordable by all receivers.

Delay and Buffering Requirements. When streaming live content, the delay is in proportion to the block size. Fig. 3.7(b) depicts the delay caused by the authentication scheme for different values of n. For example, with $n = 5$ and a GoP size of 8, the delay is less than 2 seconds, which is quite acceptable. Moreover, a value of $n = 5$ indicates that in the worst case, where the authentication information of the first four GoPs of a block is lost, the receivers need to buffer 5 GoPs. Therefore, receivers need a small buffer only: less than 2 MB if receiving the highest-bitrate version of the stream. Note that this represents the buffering required by the authentication scheme; the streaming application may already be buffering a few seconds of video data before playing back, which can be utilized by the authentication scheme and in this case no additional buffering is needed.

Robustness Against Loss. Packet losses can negatively impact an authenticated video in two ways. First, some video packets can be lost. Second, some packets, although received, can be unusable as they cannot be verified. Thus, authentication may amplify the effect of losses. We show that our scheme does not suffer from these issues. In our simulator, once a GoP is received, the receiver checks the attached authentication information and may drop a subset of packets from the stream, which cannot be authenticated due to the loss of the corresponding authentication information. The result of this process is shown in



(a) Distribution of user bandwidth.



(b) Number of truncation units per frame.

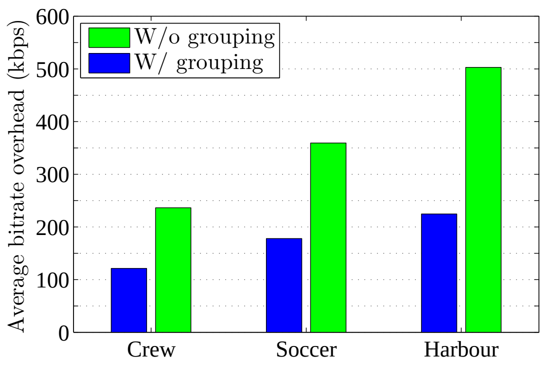
Figure 3.9: User bandwidth distribution and the number of truncation units per frame at different temporal levels.

Fig. 3.8, where the fraction of the packets received and verified over the total packets is depicted.

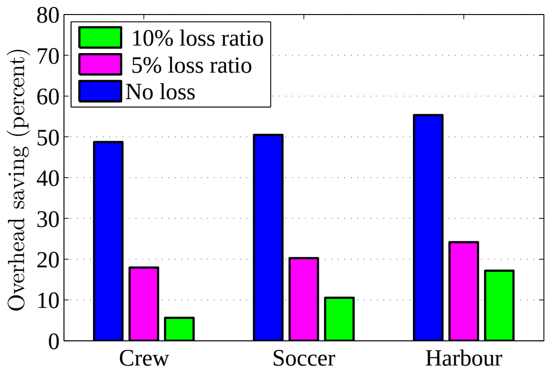
As discussed earlier, the authentication information of quality/spatial layers is replicated in a few, k, copies for protection against loss. The first finding by Fig. 3.8 is that our authentication scheme increases the impact of loss only marginally: as the loss ratio increases from 0 to higher values, the gap between the plain (unauthenticated) video and the authenticated video with $k = 2$ or 3 increases negligibly. As a second finding, Fig. 3.8 also helps us to determine how many copies the authentication information packet should be sent in: $k = 1$ results in high sensitivity to loss, $k = 2$ is suitable for reasonable loss ratios ($< 10\%$), and $k = 3$ becomes the preferred choice as the loss ratio grows higher. Notice that lower k values are always preferred, since the amount of communication overhead is directly proportional to k.

Communication Overhead. We measure the communication overhead as the additional bandwidth a receiver has to consume to receive the authentication information, averaged over all receivers. The overhead for the three video streams, when not employing the overhead reduction algorithm, is shown with the green bars in Fig. 3.10(a).

To reduce this overhead, we run the proposed algorithm for grouping truncation units of each frame. The number of truncation units in a frame can vary from frame to frame, especially for frames at different temporal layers. Figure 3.9(b) shows the average number of



(a) Communication overhead.



(b) Saving in communication overhead.

Figure 3.10: The efficiency of the overhead reduction algorithm.

truncation units in the frames of each temporal layer. As expected, frames of lower temporal layers have many more packets—as these frames are used for prediction of more frames, they are encoded with lower quantization parameters to provide stronger prediction signals. There are at least a few dozens of truncation units in each frame, which we optimally group using the proposed overhead reduction algorithm. We assume a multi-modal Gaussian distribution for modeling user bandwidth in a heterogenous environment. In this distribution, shown in Figure 3.9(a), three major concentrations of user bandwidths are assumed at 512 kbps, 4 Mbps, and 8 Mbps. By applying the proposed algorithm, the average overhead is considerably reduced for streams as shown in Fig. 3.10(a) and 3.10(b). The saving is higher in the third stream, which has a higher bitrate and a higher number of truncation units per frame. More truncation units per frame provides opportunity for more efficient grouping. We note that packet losses can impact the efficiency of our grouping algorithm, as discussed in Section 3.6.2, though even in the presence of losses the algorithm still benefits us in reducing the overhead. The gain, however, is reduced from 50% to around 20% and 10% when a loss of ratio 5% and 10% is introduced. For higher loss ratios, although this benefit diminishes to less than 10%, the algorithm never results in an overhead worse than that of not grouping.

As discussed earlier, on one hand the grouping process reduces the communication overhead by reducing the number of hash values needed. On the other, the grouping itself may

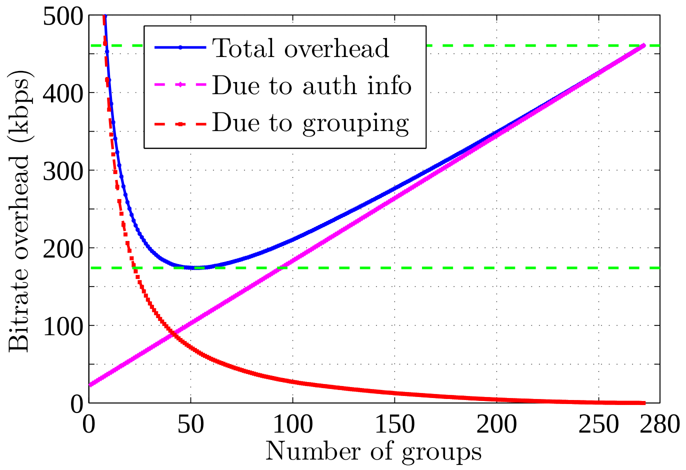


Figure 3.11: Deciding the best number of groups.

cause an overhead, because it omits some truncation possibilities and makes some users receive a lower bitrate video than they would have received if there was no grouping. Fig. 3.11 depicts this tradeoff for an average frame in the temporal base layer, and shows how the obtained optimal grouping results in a significantly lower overhead than the two ends of the tradeoff.

3.8 The svcAuth Library

We have implemented the proposed authentication scheme for H.264/SVC streams in a prototype called *svcAuth*.[1] svcAuth is available as an open source library implemented in Java to support portability across different platforms. It can be employed by any video streaming application as a transparent software add-on, without requiring any change to the encoders/decoders. As illustrated in Figure 3.12, we add an Authentication Module to the provider side, which performs post-processing of the encoded stream, and creates and embeds the authentication information in the stream. At the receiver, we add a Verification Module which verifies the received stream using the information embedded in it, and passes the verified stream to the player. Note that receivers that do not have the svcAuth Verification Module can still decode streams, since svcAuth is transparent.

[1]The latest version of svcAuth and the related documentations can be found at `http://nsl.cs.sfu.ca/wiki/index.php/svcAuth`.

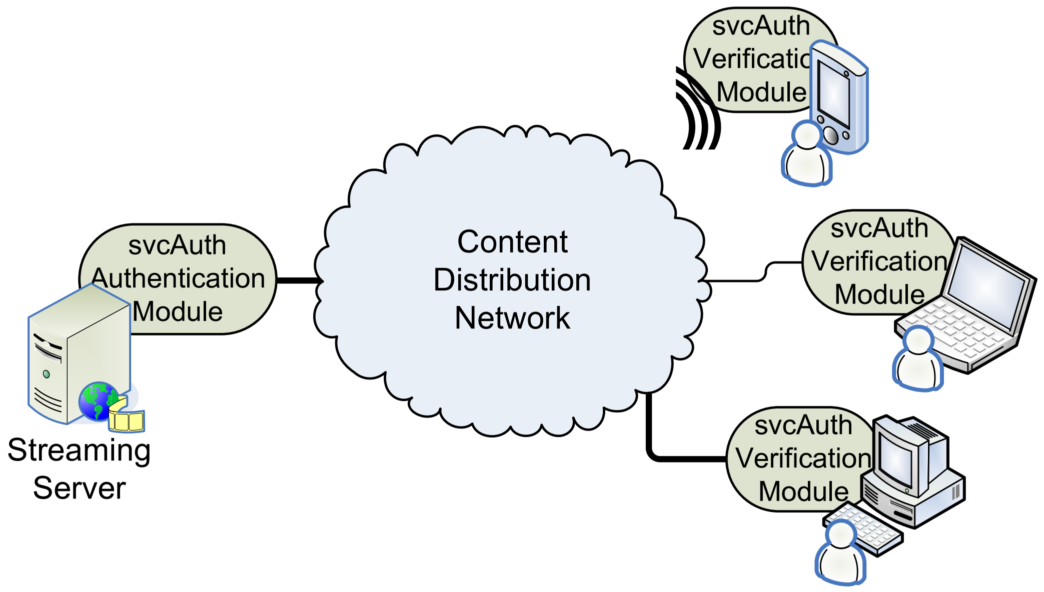

svcAuth Verification Module
svcAuth Authentication Module
Content Distribution Network
svcAuth Verification Module
svcAuth Verification Module
Streaming Server

Figure 3.12: Deployment of svcAuth Authentication and Verification Module.

svcAuth is a rather large library with over 7,000 lines of code, including sample secure streaming usages and utilities for working with SVC streams. Here we briefly review the svcAuth Authentication module. This module, which is placed after the video encoding process and before transmission, is shown in Fig. 3.13 and operates as follows. The video bitstream is first parsed by the Stream_Parser component, which extracts NAL units from the bitstream, parses their headers, and delivers them as logical objects to the SVC_Reader component. The SVC_Reader component determines the structure of the SVC stream using the NAL units. For this purpose, as shown in the figure, it needs to buffer a number of NAL units, e.g., to determine the last NAL unit of the current video frame which is done by detecting the first NAL unit of the next frame. SVC_Reader outputs a logical view of the stream as GoPs, frames, and different types of layers. We refer to these entities as SVC Elements. Each SVC Element of this structure in the logical view returned by SVC_Reader contains an array of authentication information messages, which is initially empty. These arrays are filled by the SVC_Auth component. The SVC_Auth component implements the algorithm described in Figure 3.3. It takes as input a block of n GoPs, computes the required authentication information, and adds them to the SVC Elements of those n GoPs.

The info-added SVC Elements are delivered to the SVC_Writer component, which converts back the logical structure to a raw bitstream. This is done by encapsulating the

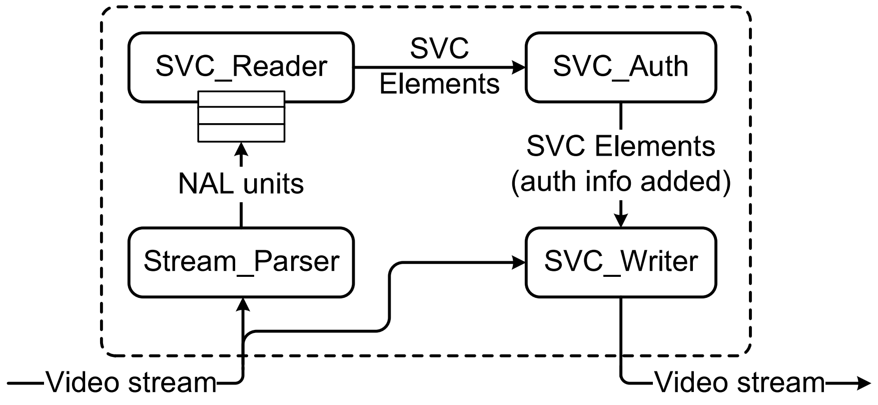


Figure 3.13: The svcAuth Authentication Module.

authentication information as appropriate NAL units and inserting them in specified locations in the original bitstream. For this purpose, we exploit the Supplemental Enhancement Information (SEI) NAL units of SVC [8]. These NAL units (NAL unit type 6) are non-VCL NAL units that can carry auxiliary information related to decoding, displaying, or other processing operations of the video. An SEI NAL unit can contain one or more SEI Messages. To attach some information to a specific layer, we embed it in an Unregistered User Data SEI Message, relate it to the desired temporal/spatial/quality layer by encapsulating (nesting) it in a Scalable Nesting SEI Message [74], and we finally encapsulate the result in an SEI NAL unit.

3.9 Summary

In this chapter, we proposed an authentication scheme for scalable video streams encoded using recent scalable coding techniques and the three-dimensional scalability model. These streams offer higher flexibility and higher coding efficiency than traditional layered scalable streams. Our authentication scheme enables verification of all possible substreams, and we analytically proved its security. We have implemented this scheme for H.264/SVC video streams as an open source library called *svcAuth*, which can be employed as a transparent add-on component by any streaming application. We designed an additional algorithm for minimizing the communication overhead, which can become non-negligible for highly flexible streams. The overhead could be reduced by the proposed algorithm by more than 50% in our experiments. The overhead reduction algorithm can also be used with other authentication schemes in the literature, which are designed for traditional scalable videos, to optimize their overhead. We conducted a simulation study with real video traces, which shows that our authentication scheme is robust against reasonable packet loss rates ($< 20\%$), has low communication overhead, incurs negligible computational cost, adds only a short (1–2 sec) delay, and requires no significant buffering (< 2 MB) by receives.

Chapter 4

Seed Server Allocation in P2P Streaming Systems

In this chapter, we study the problem of allocating the resources of seed servers for serving scalable video streams in P2P streaming systems. We formulate this problem, show its NP-completeness, and propose two approximation algorithms to solve it. We then evaluate the proposed solutions to confirm their efficiency and near-optimality.

4.1 Introduction

In recent years, several P2P streaming systems have been designed and deployed for large-scale user communities [75–78]. Most of these systems, however, still use nonscalable video streams [79]. Our goal in this chapter is to leverage scalable video streams to improve the quality observed by diverse clients. Among the different challenges that need to be dealt with for running an efficient P2P streaming system [4,80], we focus on efficient management of the resources of seed servers. These servers are needed in high-quality P2P streaming systems to make up for the limited upload bandwidth of peers compared to their demanded download rates. For example, while an average-to-good quality video stream requires about 1–2 Mbps bandwidth, the typical upload capacity of home users with DSL and cable connections is often less than a few hundred kilobits per second.

When serving scalable video streams in a P2P network, the data demanded/possessed by peers gets heterogeneous. Accordingly, allocating seed resources for serving peers' requests arbitrarily, as in most of today's P2P streaming systems [79], will result in poor management of resources and inefficient utilization of data. Moreover, the flexibility offered by scalable video streams should be appropriately taken advantage of to best satisfy the demands of peers using a given limited resource.

We study efficient allocation of seed servers resources in P2P streaming systems. We consider both live streaming and video on-demand scenarios. These seed servers have finite serving capacity and are often loaded with a volume of requests larger than their capacity. We formulate the problem of allocating this capacity for optimally serving scalable videos. We show that this problem is NP-complete, and propose two approximation algorithms to solve it, which complement each other. The first one allocates seeding resources for serving peers based on dynamic programming, and is more suitable for small seeding capacities ($\leq$ 10 Mbps). The second algorithm follows a greedy approach and is more efficient for larger capacities. We evaluate the proposed algorithms analytically and in a simulated P2P streaming system. The results confirm the efficiency and near-optimality of the proposed algorithms, and show that higher-quality videos are delivered to peers if our algorithms are employed for allocating seed servers.

This chapter is organized as follows. Related works are summarized in Section 4.2. In Section 4.3, the considered P2P streaming model is presented, and the seed server allocation problem is formulated and proven to be NP-complete. Two approximation algorithms for the problem are presented in Section 4.4. We evaluate the proposed algorithms in Section 4.5. Section 4.6 provides a summary of this chapter.

4.2 Related Work

Cui et al. [81] and Rejaie et al. [16] study P2P streaming systems with scalable videos, focusing on the tasks of peers. An algorithm is presented in [81] to be run on each peer independently that decides how to request video layers from a given set of heterogeneous senders, assuming layers have equal bitrate and provide equal video quality. Hefeeda et al. [18] study this problem for Fine-Grained Scalable (FGS) videos, taking into account the rate-distortion model of the video for maximizing the perceived quality, which is more accurate than assuming all layers have equal quality enhancements as supposed in [81]. We too consider video layers with heterogenous rates and quality enhancements. In the framework presented in [16], the problem of requesting from a set of senders is studied from a practical perspective. A receiver periodically sends an ordered list of requested packets to each sender, and the sender provides packets in the given order according to its TCP-friendly congestion control mechanism.

Lan et al. [17] present a high level architecture for data-driven P2P streaming with scalable videos. The authors propose a scheduling algorithm for peers to request data from senders. This algorithm, however, does not explicitly take the scalable nature of the video into account. The packet scheduling problem for scalable video steams is more challenging than nonscalable streams. Due to their adaptability to bandwidth variations, naively fetching video data from other peers may result in frequent variations in the number of video layers. This causes fluctuations in the video quality, which may be even worse than just watching a low quality video [82]. This packet scheduling problem is studied in [83,84].

All of these works do not consider the functionalities of seed servers, which are critical to provide high-quality video streaming services. This is because the upload bandwidths of peers are often far less than their demanded download rates. For example, an average-to-good quality video stream requires about 1–2 Mbps, whereas the average upload capacity of home users with DSL and cable connections is often less than a few hundred kilobytes. To make up for this asymmetry, a number of seed servers need to be deployed in the network. Xu et al. [19] study the functionality of seed servers for P2P streaming. However, their work is only for nonscalable video streams, and they also assume that peers' upload bandwidths can take a number of certain values only. The case for scalable video streams is more challenging as various substreams need to be handled. In [81], seed servers are assumed to always have enough capacity to serve all requests, which is not realistic. In this chapter, we consider a more practical scenario in which seed servers have finite capacity, and this finite capacity needs to be optimally allocated to requesting peers such that a higher-quality video is delivered to all peers.

4.3 System Model and Problem Statement

In this section, we describe the considered system model and state the resource allocation problem addressed in this chapter.

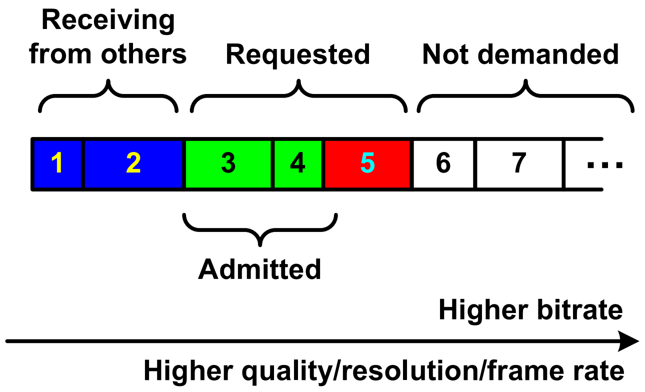


Figure 4.1: A request from a peer who is demanding the first five layers in total, and is receiving the first two from other peers and the next two from a seed server.

4.3.1 System Overview

The considered P2P streaming architecture consists of trackers, seed servers, and peers. Peers join the system by contacting one of the trackers. The tracker receives periodic update reports from peers, informing it about their available data and capacity. This enables the tracker to monitor its network and keep track of the set of active peers, their contribution, and their data availability. Note that the tracker does not keep track of the topology of the network, i.e., the list of partners of each peer. A number of seed servers exist in the network to serve requests when there is not enough capacity in the peer population. Our problem is to decide which subset of requests should be served by the seed servers to maximize a system-wide utility function. This problem is important because the volume of requests to be served often exceeds the seeding capacity. Allocating seeding resources optimally will lead to better utilization of seed servers, and higher video quality for users, especially during periods with excessive loads which are typically the most difficult to handle in real systems.

Peers are expected to use their limited upload bandwidth for serving lower layers first, so as to avoid having some peers starving while other peers are receiving highest rates. Moreover, peers try to serve as many layers as they can upload. For example, if all layers have a rate of 100 kbps and a peer has 250 kbps upload bandwidth, it will upload the two lowest layers at rate 100 kbps and the third one at 50 kbps.

Table 4.1: List of notations used in the seed server allocation problem.

Parameter	Description
Δ	The tracker decides allocation of the seed servers every Δ seconds.
V	The set of videos or TV channels.
P_v, T_v	The set of peers watching and the number of video segments of video $v \in V$.
v_p, u_p	The video/TV channel being watched by and the upload capacity (bps) of peer p.
$r_{v,l}$	Rate (bps) of the l-th layer of video v.
C	The total capacity (bps) of seed servers.
K	Number of requests in the queue.
req_k	The k-th request in the queue: $\{req_k.p,\ req_k.t,\ req_k.l_1,\ req_k.l_2\}$.
n_k	$n_k = req_k.l_2 - req_k.l_1 + 1$ is the number of layers requested with req_k.
$req_{k,j}$	A sub-request of req_k considering only the j lowest requested layers ($1 \leq j \leq n_k$).
$c_{k,j}$, $b_{k,j}$	Cost and utility of sub-request $req_{k,j}$.

4.3.2 Problem Statement and Hardness

Peers' requests are gathered in the tracker's request queue. The tracker decides every Δ seconds, which is a few seconds, and accepts some requests (to be served by a seed server) and rejects others. Let V denote the set of video files in an on-demand session or the set of channels in a live streaming scenario. We divide a video into short time intervals, called video segments, the number of which is T_v for each video $v \in V$. A video segment is considered an atomic unit of adaptation, meaning that the number of layers received by a peer is assumed constant during a media segment, but may vary between consecutive segments. P_v is the set of peers currently participating in the streaming session of a video $v \in V$. At each time the tracker solves the allocation problem, there are K requests in the queue. Each request req_k is in the form $\{req_k.p,\ req_k.t,\ req_k.l_1,\ req_k.l_2\}$, meaning that peer $req_k.p$ is requesting layers $req_k.l_1$ through $req_k.l_2$ (inclusive) of the stream, starting at segment $req_k.t$; the peer could be receiving layers 1 through $req_k.l_1 - 1$ from other peers. an example of this is illustrated in Figure 4.1. Since req_k is for $n_k = req_k.l_2 - req_k.l_1 + 1$ layers and may be admitted partially, we break it to n_k *sub-requests*, denoted by $req_{k,j}$ where $1 \leq j \leq n_k$. A sub-request $req_{k,j}$ represents a request for the j lowest requested layers, i.e., $req_{k,j}$ corresponds to layers $req_k.l_1$ through $req_k.l_1 + j - 1$. Let $r_{v,l}$ denote the bitrate (bps) of the l-th layer of the video v, and u_p be the upload capacity (bps) of peer p. A list of notation is provided in Table 4.1 for quick reference.

Serving each sub-request $req_{k,j}$ has a cost $c_{k,j}$ for seed servers which is the sum of the bitrates of the j requested layers. Letting ν denote the requested video $\nu = v_{req_k.p}$ in req_k, we denote the costs of req_k's sub-requests by:

$$c_{k,j} = \sum_{l=req_k.l_1}^{req_k.l_1+j-1} r_{\nu,l} \quad (1 \le k \le K \ , \quad 1 \le j \le n_k). \tag{4.1}$$

Moreover, by admitting $req_{k,j}$, a utility (benefit) $b_{k,j}$ is gained by the system, which consists of the utility of serving the associated layers to the corresponding peer, that is, $\sum_{l=req_k.l_1}^{req_k.l_1+j-1} b_{self}(req_k.p, l)$, and the utility gained when the peer shares those layers with the network, denoted by $\sum_{l=req_k.l_1}^{req_k.l_1+j-1} b_{share}(req_k.p, l)$. Our algorithms are not restricted to a specific $b_{self}(p, l)$ function; we see in Section 4.4.3 two sample utility functions to maximize the average quality received by peers and to provide max-min fairness among quality received by peers according to their demands. For calculating $b_{share}(p, l)$, we need to consider the peer serving those layers (or part of them) to its partners, those partners serving (partially) to their partners, and so on. Taking these neighborhood details into account requires knowledge of the network topology, which is difficult to maintain for dynamic P2P systems. We therefore compute $b_{share}(p, l)$ as the *expected* utility that the system gains when a peer shares some video layers with the network. In Section 4.4.3 we see how to calculate these expected utilities according to $b_{self}(p, l)$.

Problem 3 (Seed Server Allocation) *Given the requests $req_1, \dots, req_K$, their costs $c_{k,j}$ bps and utilities $b_{k,j}$ $(1 \le k \le K,\ 1 \le j \le n_k)$, and a seeding capacity C bps, find the x_k $(0 \le x_k \le n_k)$ value for each req_k which indicates that sub-requests $req_{k,1}, req_{k,2}, \dots, req_{k,x_k}$ should be served out of req_k in order to maximize the system-wide utility.*

This problem is formulated as follows. Find x_k in order to:

$$\max \qquad \sum_{k=1}^{K} b_{k,x_k} \tag{4.2a}$$

$$\text{s.t.} \qquad \textstyle\sum_{k=1}^{K} c_{k,x_k} \le C \tag{4.2b}$$

$$x_k \in \{0, 1, \dots, n_k\} \quad (1 \le k \le K) \tag{4.2c}$$

Theorem 3 *The seed server allocation problem defined in Eq. (4.2) is NP-complete.*

Proof: We prove the NP-completeness by reducing the Knapsack Problem [85] to a simplified version of the seed server allocation problem. Suppose that all videos are single-layer coded and thus all requests are for the first layer. In this case, all x_i values are either 0 or 1. This special case of the problem is equivalent to the 0-1 Knapsack Problem. In addition, a solution for the seed server allocation problem can easily be verified in polynomial time. Hence, the seed server allocation problem is NP-complete. $\square$

4.4 Problem Solution

In this section, we present two approximation algorithms for the seed server allocation problem. The first algorithm produces close-to-optimal results for small seeding capacity C, but as the capacity increases, it has to get far from the optimal in order to operate in real-time. The second algorithm runs in a time independent of the seeding capacity and can always operate in real-time. It provides close-to-optimal results for large seeding capacities, but becomes far from the optimal for small capacities.

4.4.1 SRA_DP: Seed Resource Allocation using Dynamic Programming

Since our server allocation problem has some similarities with the Knapsack problem, it is intuitive to check the applicability of Knapsack solutions to our problem. The Knapsack problem has an interesting optimal solution using dynamic programming, and a consequent approximation solution [85]. However, if to be applied to the seed server allocation problem, this algorithm can function only for the single-layer allocation problem (see the proof of Theorem 3). We propose a dynamic programming algorithm for the general case with multi-layer videos. Unlike the approximation algorithm for the Knapsack problem, our algorithm accounts for the consistency in serving sub-requests of each request req_k. That is, no higher layer must be served unless all of its lower layers are already served. We first transform all utility values $b_{k,j}$, which are real numbers, to integers $b'_{k,j} = \lfloor \frac{b_{k,j}}{M} \rfloor$ where M is a constant real number greater than zero, e.g., we set $M = 0.1$ for neglecting the second and further decimal points of $b_{k,j}$ values. We then optimally solve the problem with $b'_{k,j}$ values. The value M determines the approximation factor and the running time of the algorithm, as we analyze shortly.

The dynamic programming algorithm, denoted by SRA_DP, operates as follows. Let B'_{max} denote the maximum $b'_{k,j}$ for all valid (k, j) values, C_{min} the minimum $c_{k,j}$, i.e., the bitrate of the base layer, and $K' = \sum_{k=1}^{K} n_k$ the total number of sub-requests. Thus, C/C_{min} is an upper bound on the number of sub-requests that can be served, and $I = \frac{C}{C_{min}} B'_{max}$ is an upper bound on the total utility that can be gained. We define $a[k,i]$ $(0 \le k \le K,\ 0 \le i \le I)$ as the minimum cost that a subset of requests $req_1, \ldots, req_k$ can have, whose total utility exactly equals i; $a[k,0]$ for all $0 \le k \le K$ is set to 0 and $a[0,i]$ for all $1 \le i \le I$ is assigned ∞. If no subset of sub-requests with a total utility of i can be formed, $a[k,i]$ is set to ∞. Having initialized $a[k,0]$ and $a[0,i]$ values, the rest of the matrix is calculated as in Eq. (4.3):

$$a[k,i] = \min \begin{cases} a[k-1,i], & \\ a[k-1, i-b'_{k,1}] + c_{k,1}, & \text{if } i \ge b'_{k,1} \\ a[k-1, i-b'_{k,2}] + c_{k,2}, & \text{if } i \ge b'_{k,2} \\ \vdots & \\ a[k-1, i-b'_{k,n_k}] + c_{k,n_k} & \text{if } i \ge b'_{k,n_k} \end{cases} \tag{4.3}$$

In Eq. (4.3), the first option represents the case that no layer of req_k is served, the second option represents the case when one layer is served, and so on. The min-cost value among these options is chosen. Suppose x layers are to be served out of the n_k layers of req_k in order to make a total utility of i, which is only possible if $b'_{k,x} \le i$. Then, the total cost will be that of x layers from req_k, i.e., $c_{k,x}$, as well as the minimum cost for obtaining a utility of $i - b'_{k,x}$ using the previous $k-1$ requests, which add up to $b[k-1, i-b'_{k,x}] + c_{k,x}$ as represented in Eq. (4.3). The optimal utility that can be gained using the capacity C is obtained by finding the maximum i such that $a[K,i]$ does not exceed C, and we denote this utility by i^*.

To keep track of how these optimal sub-solutions are built, i.e., to obtain x_k values in Eq. (4.2), we keep another matrix $y[k,i]$ $(0 \le k \le K,\ 0 \le i \le I)$. Each $y[k,i]$ for nonzero k and i values holds the number of sub-requests served from the request req_k in the solution to subproblem (k,i). $y[0,i]$ and $y[k,0]$ are set to zero. In each iteration where an $a[k,i]$ is calculated according to Eq. (4.3), $y[k,i]$ is set to the number of sub-requests, i.e., the row index (starting from 0) in Eq. (4.3), that makes the minimum. Finally, we obtain the $x[k]$ values for Eq. (4.2) as follows. Let s be a pointer that is initially set to i^*. First, $x[K]$ is set as $x[K] = y[K,s]$. Then, s is reduced by $b'_{K,x[K]}$, i.e., the utility of serving $x[K]$ sub-requests out of the K-th request. The new s value points to the best utility we got using requests $req_1, \ldots, req_{K-1}$. Thus, $x[K-1]$ is obtained as $x[K-1] = y[K-1,s]$. Then, the pointer

s is updated accordingly, i.e., reduced by $b'_{K-1,x[K-1]}$, and so on. The running time of the algorithm is analyzed shortly.

We now derive the approximation factor for the proposed algorithm. The proof proceeds in a similar way to the proof of the Fully Polynomial Time Approximation Scheme (FPTAS) for Knapsack [85]. We, however, provide a tighter approximation factor by using the characteristics of our problem.

Theorem 4 *The SRA_DP algorithm returns a solution for the seed server allocation problem (Problem 3) with worst-case approximation factor of* $1-\dfrac{CM}{C_{min}B_{max}}$, *where* C $(C \geq c_{k,j})$, C_{min}, *and* B_{max} *are the seeding capacity, the bitrate of the base layer, and the maximum utility among all sub-requests, respectively. Furthermore, the time complexity of the SRA_DP algorithm is of* $O(K'\lfloor\dfrac{CB_{max}}{C_{min}M}\rfloor)$, *where* K' *is the total number of sub-requests.*

Proof: According to the definition and usage of $x[k]$ values, it is clear that the serving of sub-request is consistent. To obtain the approximation factor, we first need to make sure that the proposed dynamic programming algorithm is optimal with rounded utility values $b'_{k,j}$. For this purpose, it is enough to show that the problem has *optimal substructure* property, since the optimality of the algorithm (with $b'_{k,j}$ values) immediately follows this property. The optimal substructure property means that the optimal solution to the problem contains within it optimal solutions to subproblem. This is true for our problem, because if in the optimal solution (with utility $a[K,i^*]$) the solution to a subproblem (k,i) is not optimal —meaning that a utility of i could have been achieved out of the first k requests at a lower cost that $a[k,i]$— we can simply replace that part of the final optimal solution and obtain a better utility than $a[K,i^*]$, which is contradiction. We now analyze the approximation factor. Suppose the optimal solution to the original problem with $b_{k,j}$ values is the set O of sub-requests, and the solution to the problem with $b'_{k,j}$ values is O'. Let the function $b(O)$ denote the sum of the $b_{k,j}$ utilities of sub-requests in O; likewise for $b'(O)$ that sums $b'_{k,j}$ values. Thus, the obtained utility and the optimal one are $w = b(O')$ and $OPT = b(O)$, respectively. Since we round down $b_{k,j}$ values by a factor of M, we have:

$$b_{k,j} \geq Mb'_{k,j} \Rightarrow b(O') \geq Mb'(O')$$

$$b_{k,j} - Mb'_{k,j} \leq M \Rightarrow b(O) - Mb'(O) \leq |O|M \leq \frac{C}{C_{min}}M$$

where the latter inequality is based on the observation that any subset O of sub-requests that fits in a capacity of C has at most C/C_{min} elements. Because the subset O' is optimal with $b'_{k,j}$ values, we have:

$$b'(O') \geq b'(O) \Rightarrow w = b(O') \geq Mb'(O') \geq Mb'(O) \geq b(O) - \frac{C}{C_{min}}M \Rightarrow$$
$$w \geq (1 - \frac{CM}{C_{min}OPT})OPT \Rightarrow w \geq (1 - \frac{CM}{C_{min}B_{max}})OPT. \tag{4.4}$$

For each row k of the matrices $a[k,i]$ and $y[k,i]$ together, $O(In_k)$ comparisons are performed, where $I = \frac{C}{C_{min}}B'_{max}$ is an upper bound on the total utility that can be gained. Thus, the running time of the algorithm is:

$$\sum_{k=1}^{K} O(In_k) = O(K'I) = O(K'\lfloor\frac{CB_{max}}{C_{min}M}\rfloor). \tag{4.5}$$

□

We numerically analyze the running time of the algorithm and its relation to the approximation factor $\gamma = 1 - \frac{CM}{C_{min}B_{max}}$. As Eq. (4.4) and Eq. (4.5) show, by decreasing the rounding factor M, the approximation factor approaches 1, which, on the other hand, increases the computational complexity. To have an estimate of this relation, suppose the number of iterations that the tracker can perform in Δ seconds, i.e., till next turn to run the seed server allocation algorithm, is given as Ω. Each iteration of the algorithm consists of a few lookups, one addition, two comparisons, and possibly one assignment. Also suppose the number of iterations needed by the algorithm is simply $K'\frac{CB_{max}}{C_{min}M}$, which must not exceed Ω. Therefore, according to Eq. (4.4) and Eq. (4.5), the approximation factor γ will be as follows:

$$M \geq \frac{K'CB_{max}}{\Omega C_{min}} \Rightarrow \gamma \geq 1 - \frac{K'}{\Omega}(\frac{C}{C_{min}})^2. \tag{4.6}$$

This is illustrated in Figure 4.2 where the guaranteed approximation factor γ is depicted versus the computational power, assuming that the base layer of the videos is at 100 kbps, there are $K' = 1000$ sub-requests in the queue, and that each iteration roughly takes 100 machine instructions to execute. According to Figure 4.2, a tracker with seed servers of total capacity 10 Mbps whose allocation is decided every second, can make sure that the approximation result is at least as good as 90% of the optimal if it can perform 10 giga low-level instructions per second.

In sum, the SRA_DP algorithm obtains answers very close to the optimal for small seeding capacities, but it has to sacrifice some non-negligible approximation factor for large seeding capacities. We propose another approximation algorithm for these cases in the next subsection.

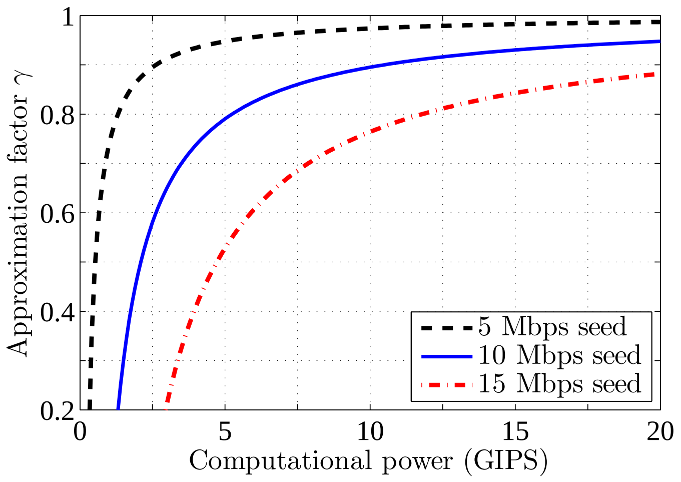


Figure 4.2: Tradeoff between approximation factor and computational complexity.

4.4.2 SRA_GREEDY: Seed Resource Allocation using a Greedy Algorithm

We present another approximation algorithm for the seed server allocation problem, whose running time is independent of the seeding capacity and only depends on the number of requests in the queue. We show that the result of this approximation gets very close to the optimal answer as the seeding capacity increases, e.g., $C = 50$ Mbps, though is worse than the algorithm SRA_DP if the seeding capacity is small, e.g., $C = 10$ Mbps. Our approximation is based on relaxing the Integer Programming (IP) problem in Eq. (4.2) to its equivalent Linear Programming (LP) problem, in which the constraint (4.2c) is relaxed to (4.7a) and (4.7b).

$$0 \leq x'_k \leq n_k \quad (1 \leq k \leq K) \tag{4.7a}$$

$$x'_k \in \mathbf{R}. \tag{4.7b}$$

In other words, we now allow a layer to be partially served, though it is not meaningful in practice. Having solved the LP problem in Eq. (4.7) and obtained the x'_k values, we obtain a valid solution to the original IP problem by rounding down all x'_k values: $x_k = \lfloor x'_k \rfloor$. Clearly, x_k values form a valid answer for the IP form in Eq. (4.2), since they satisfy both constraints (4.2b) and (4.2c). We will see shortly that after this relaxation and down-rounding, how close the objective function (4.2a) will be to the optimal solution. The proposed algorithm is called

SRA_GREEDY

```
GreedyAllocation ($K$, $C$, $n[]$, $b[][]$, $c[][]$)
1. // $K$: number of requests, $C$: seeding capacity
2. // $n[k]$: number of sub-requests in the $k$-th requests ($1 \le k \le K$)
3. // $b[k][j]$, $c[k][j]$: utility and cost of the $j$-th sub-request of the $k$-th request
   ($1 \le j \le n[k]$); $b[k][0]$ and $c[k][0]$ are assigned 0
4. // Output: $x[]$: number of sub-requests to serve from the $k$-th request
5. $K' = \sum_{k=1}^{K} n[k]$      // total number of sub-requests
6. $\mathbf{S}[] \longleftarrow$ the $K'$ sub-requests sorted in decreasing order of utility-to-cost ratio
7. $z \longleftarrow 0$               // will be the total obtained utility
8. $x[] \longleftarrow$ CreateArray($K$, 0)
9. for $(k, j) \in \mathbf{S}$ do
10.    if $x[k] > j$ then continue
11.    $cost \longleftarrow c[k][j] - c[k][x[k]]$
12.    $utility \longleftarrow b[k][j] - b[k][x[k]]$
13.    if $cost \le C$ then
14.      $C \longleftarrow C - cost$
15.      $z \longleftarrow z + utility$
16.      $x[k] \longleftarrow j$
17. done
18. return $z$, $x[]$
```

Figure 4.3: The SRA_GREEDY algorithm for the seed server allocation problem.

SRA_GREEDAY and is shown in Figure 4.3. Sub-requests are sorted in decreasing order of utility-to-cost ratio and are picked one by one in each iteration. Since some sub-requests are overlapping, i.e., each sub-request (k, j) is a subset of sub-requests $(k, j+1), \dots, (k, n_k)$, in each iteration we take those layers from sub-request (k, j) that are not already served by another sub-request $(k, j' < j)$. The algorithm consists of sorting sub-requests, which runs in $O(K' \log K')$ where K' is the total number of sub-requests, and performing K' iterations of $O(1)$, which makes the total running time $O(K' \log K')$. This is easily practical in real-time for reasonable K' values (< 500K).

Theorem 5 *If all costs $c_{k,j}$ are bounded as $c_{k,j} \le c_{max} < C$ for all valid k, j values, the algorithm SRA_GREEDY is a $\frac{c_{max}}{C - c_{max}}$ -factor approximation for the seed server allocation problem, i.e., $z \ge (1 - \frac{c_{max}}{C - c_{max}})OPT$.*

Proof: Consider the following two modifications to the algorithm in Figure 4.3: (i) after line 16 in the code, if *cost* $> C$ then pick a portion θ $(0 \leq \theta < 1)$ of the current sub-request (k, j) which can fill the capacity C, and quit the loop, (ii) after line 16 in the code, if *cost* $> C$ then just quit the loop. Case (i) refers to the solution to the LP problem in Eq. (4.7) and provides the optimal answer for it; the proof is simple and is similar to that of the solution to Knapsack's LP-relaxation. Let z^* denote the utility obtained by this solution, which is at least as good as OPT since it refers to the LP-relaxation of the original problem. Case (ii) is only for a comparison purpose. Since its solution is a subset of the solution obtained by our algorithm in Figure 4.3, the utility obtained by case (ii), denoted by z', is a lower bound on our obtained utility z, i.e., $z' \leq z$.

Let $m + 1$ and m be the number of sub-requests taken by modifications (i) and (ii), respectively. Also let $b_1, \ldots, b_{m+1}$ and $c_1, \ldots, c_{m+1}$ be the short for the utilities and costs of these sub-requests. Since the $m + 1$-st sub-request served has a utility-to-cost ratio less than or equal to each of the previous m ones, we have:

$$\frac{b_{m+1}}{c_{m+1}} \leq \frac{b_i}{c_i} \text{ for all } i \in \{1, 2, \ldots, m\} \Rightarrow \frac{b_{m+1}}{c_{m+1}} c_i \leq b_i \Rightarrow b_{m+1} \leq c_{m+1} \frac{\sum_{i=1}^{m} b_i}{\sum_{i=1}^{m} c_i} \tag{4.8}$$

The margin between z and OPT is bounded as follows:

$$\begin{aligned}
& z' \leq z \leq OPT \leq z^* \\
& \Rightarrow \epsilon = 1 - \frac{z}{OPT} \leq 1 - \frac{z'}{z^*} = \frac{z^* - z'}{z^*} = \frac{\sum_{i=1}^{m} b_i + \theta b_{m+1} - \sum_{i=1}^{m} b_i}{\sum_{i=1}^{m} b_i + \theta b_{m+1}} \leq \\
& \quad \frac{b_{m+1}}{\sum_{i=1}^{m} b_i} \leq c_{m+1} \frac{\sum_{i=1}^{m} b_i}{\sum_{i=1}^{m} c_i} \frac{1}{\sum_{i=1}^{m} b_i} \leq \frac{c_{max}}{\sum_{i=1}^{m} c_i} \leq \frac{c_{max}}{C - c_{max}} \\
& \Rightarrow z \geq (1 - \frac{c_{max}}{C - c_{max}})\ OPT
\end{aligned} \tag{4.9}$$

□

For example, for a 2 Mbps video and a seeding capacity of 25 Mbps, it is guaranteed that the greedy approximation solution will produce results as good as 91% of the optimal. For larger seeding capacities, this factor approaches 1. For small seeding capacities (≤ 10 Mbps), however, the approximation factor is low, e.g., 75% for $C = 10$ Mbps. In this case, employing the SRA_DP algorithm is recommended as its factor is close to 1 for small seeding capacities.

In sum, according to the seeding capacity, the computational power, and the maximum bitrate of the videos being served, one can determine the guaranteed approximation factor of both algorithms SRA_DP and SRA_GREEDY using Equations (4.6) and (4.9) choose the more appropriate one.

4.4.3 The Utility Function

The proposed allocation algorithms are general and can adopt different utility functions to suit the objective of various practical systems. In this section, we define two sample utility functions, which are to (i) maximize the average quality received by peers, and (ii) provide max-min fairness among quality received by peers according to their demands.

Recall that the utility function consists of two parts, $b_{self}(p, l)$ and $b_{share}(p, l)$, as discussed in Section 4.3.2. Let us first calculate the former part, $b_{self}(p, l)$. This function is calculated differently in cases (i) and (ii) mentioned above. For case (i), we have:

$$b_{self}(p, l) = q(v_p, l) - q(v_p, l - 1), \tag{4.10}$$

where v_p is the video being watched by peer p and $q(v_p, l)$ denotes the video quality, e.g., Y-PSNR, of the first l layers of the video. The value $q(v_p, 0)$ needs to be realistically defined; note that a blank video can produce a Y-PSNR of 10 to 20 dB, depending on the original video. The $q(v_p, 0)$ value determines the importance of the base layer and does not significantly affect the way the enhancement layers are served in our algorithm: $q(v_p, 1) - q(v_p, 0)$ is the utility of the base layer that can be defined superior to any other layer in order to make sure everyone will receive the base layer. We set $q(v_p, 0)$ as 25 dB in our experiments in Section 4.5.

In case (ii), let d_p denote the the quality demand of a peer p, e.g., the Y-PSNR of the video substream that the peer requests to download, as the case in our evaluations in Section 4.5. Denoting by q_p the quality of the video that a peer p receives, we refer to the fraction $\frac{q_p}{d_p}$ as *peer satisfaction*. The goal is to maximize the minimum peer satisfaction among all peers. For example, if the resources in the network are half the total demanded, in the ideal case every peer should receive half of its demanded quality, not some peers completely satisfied and others starving. To achieve this goal, we define the utility function as:

$$b_{self}(p, l) = \frac{d_p - q(v_p, l - 1)}{d_p - q(v_p, 0)}, \tag{4.11}$$

At each instance, for peers with different demands d_p and different video qualities q_p being currently received, the utility function in Eq. (4.11) will make those peers be selected

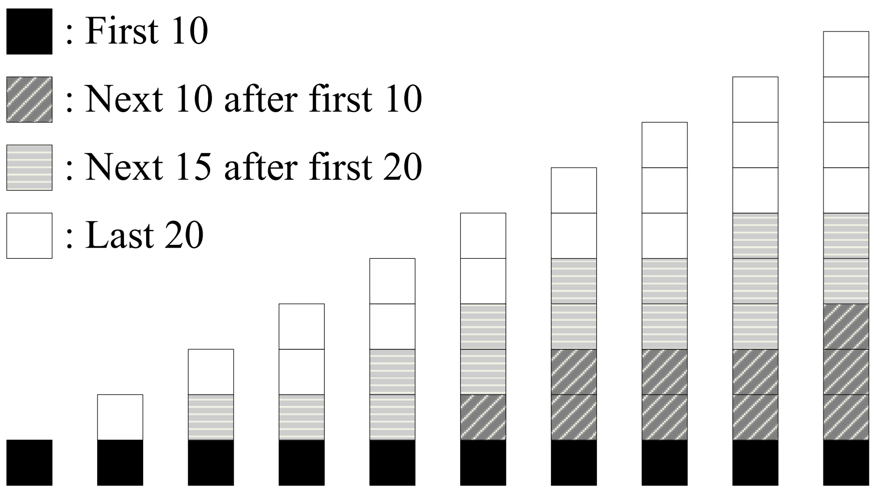


Figure 4.4: Max-min fairness using Eq. (4.11) for serving layers to different demands with limited serving capacity.

for being served that are currently receiving the least fraction of their demand. This is illustrated in Figure 4.4, in which, for simplicity, it is assumed that layers have equal bitrate and quality. In this figure, each stack of l boxes represents a user with a demand of l layers. The figure shows that by using the utility function defined in Eq. (4.11), which layers will be served to which demands if the serving capacity is enough only for serving 10, 20, 35, and 55 layers.

We now calculate the utility $b_{share}(p,l)$ according to the function $b_{self}(p,l)$ and peers upload bandwidths. As discussed earlier (Section 4.3.2), $b_{share}(p,l)$ refers to the *expected* utility gained by the system when peer p shares layer l with the network. Let L_p denote the number of layers that peer p demands. According to the way the peers are expected to share their upload bandwidth for serving different layers (Section 4.3.2), we know the rate $u_{p,l}$ at which a peer p will serve each layer l of video v:

$$u_{p,l} = \min\left\{u_p - \sum_{i=1}^{l-1} u_{p,i},\ r_{v,l}\right\}\ (1 \le l \le L_p). \tag{4.12}$$

If the upload bandwidth of peer p is higher than the bitrate of the demanded video, the bandwidth remained from Eq. (4.12) is equally divided among layers, i.e.,

$$u_{p,l} = \min\left\{u_p - \sum_{i=1}^{l-1} u_{p,i},\ r_{v,l}\right\} + \frac{u_p - \sum_{i=1}^{L_p} r_{v,i}}{L_p}. \tag{4.13}$$

Denote by $P_{v,l,p}$ the set of peers who demand the video v at layers higher than or equal to l, and who can possibly be served by the peer p. Then, the function $b_{share}(p,l)$ is calculated as:

$$b_{share}(p,l) = \frac{u_{p,l}}{r_{v,l}} \times \frac{1}{|P_{v,l,p}|} \times \sum_{x \in P_{v,l,p}} \Big(b_{self}(x,l) + b_{share}(x,l) \Big). \tag{4.14}$$

Calculation of Eq. (4.14) for each peer is in the order of total number of peers for arbitrary $b_{self}(p,l)$ functions, which is not efficient. However, for specific cases such as providing max-min fairness, this can be estimated efficiently. Given the upload/download bandwidth distribution of peers, which can be obtained and updated during the streaming session, we can calculate the chance that a peer p's layer demand L_p equals a given number l, to which we briefly refer as $Pr_\ell(l)$. This reduces the complexity of Eq. (4.14) from being in the order of number of peers to the order of number of video layers. For the case of maximizing the average quality, where $b_{self}(p,l) = q(v_p,l) - q(v_p,l-1)$, the share utility function in Eq. (4.14) is simply calculated as:

$$b_{share}(p,l) = \frac{u_{p,l}}{r_{v,l}} \times (\ q(v_p,l) - q(v_p,l-1)\ ). \tag{4.15}$$

For providing max-min fairness, where $b_{self}(p,l)$ is given as Eq. (4.11), we have:

$$b_{share}(p,l) = \frac{u_{p,l}}{r_{v,l}} \times \frac{1}{\sum_{i=l}^{L} Pr_\ell(i)} \times \sum_{i=l}^{L} Pr_\ell(i) \frac{i - q(v_p,l-1)}{i - q(v_p,0)}. \tag{4.16}$$

In these two calculations, a peer sharing data with its partners is taken into account, whereas further hops, i.e., those peers sharing, is not relied on, as it makes the accuracy of the calculated utilities very sensitive to network dynamics. Neglecting them, on the other hand, may underestimate the utility of peers sharing the layers, though underestimating the share utility similarly for all peers is more acceptable. At last, each of the $b_{k,j}$ values is calculated as:

$$b_{k,j} = \begin{cases} 0 & j = 0 \\ \sum_{l=req_k.l_1}^{req_k.l_1+j-1} \Big(b_{self}(req_k.p,l) + b_{share}(req_k.p,l) \Big) & 1 \le j \le n_k \end{cases} \tag{4.17}$$

4.5 Evaluation

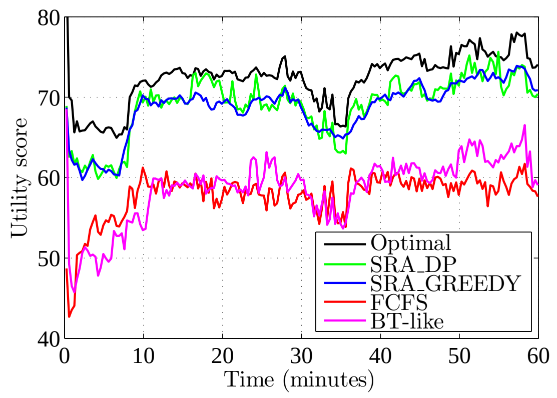
4.5.1 Simulation Setup

We simulate on-demand distribution of a video file encoded in 10 quality layers at a bitrate of 2 Mbps, which a Y-PSNR quality of 27 dB to 40 dB. The video length is 7 minutes.

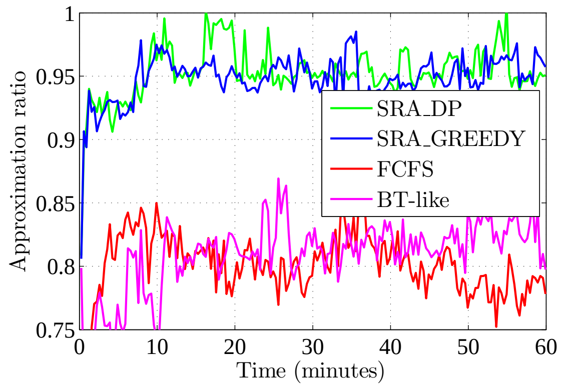
We refer to the fraction of video quality received by a peer over its demanded quality as peer satisfaction. The objective of the system is to maximize the minimum peer satisfaction (max-min fairness), as discussed in Section 4.4.3. Peers join the network according to a Poisson distribution with expected 1 arrival per second. The network consists of 400–500 peers on average. The simulation runs for 60 minutes. Each peer, once finished watching the video, stays in the network for up to 3 minutes for serving others. Each peer may leave at any time according to an exponential probability distribution, by which 25% of peers leave the network before they finish watching the video and doing the expected seeding. For generating download and upload bandwidths of peers, two classes of peers are considered. The first class has 80% of peers and represents home users, which have download bandwidths between 100 kbps to 4 Mbps and upload bandwidth between 100 kbps to 1 Mbps. The second class has 20% of peers and represents campus users, which can have download and upload rates between 100 kbps to 4 Mbps. Each peer is thus capable of downloading and uploading the base layer at least.

Each request of a peer to the tracker is first tried to be served by a list of potential senders. If there is no peer free to serve the requested layers, the requesting peer is served by the seed server. A peer might receive a layer from multiple peers, with the total receiving rate equal to the bitrate of the layer. The seed server, on the other hand, only serves whole layers. The tracker gathers requests in its queue and runs the seed server allocation algorithm once every 10 seconds. In each such run, all streaming requests in the queue along with those that are currently being served are considered together and a new set of requests to be served is determined. Each sender peer disconnects its connection to a receiver according to a Bernoulli probability distribution with an expected value of 1 minute. This is done to make enough dynamics in the network and to simulate peer failures. Video segment length is assumed 10 seconds. The simulation runs following an event-driven procedure with 10-second steps, i.e., events are gathered during a time step and applied at once at the end of the time step. To avoid wide quality fluctuations at receiver side, we take a simple heuristic that does not allow more than 1 layer change in the number of layers in two consecutive segments, i.e., the heuristic at each peer drops the enhancement layers that violate this criterion.

Allocation algorithms. In addition to the proposed allocation algorithms, we consider two other algorithms: First-Come First-Serve (FCFS) and BitTorrent-like (BT-like). SRA_DP, SRA_GREEDY, and FCFS operate as follows. First, each request in the ordered



(a) Utility score.



(b) Gained utility over the maximum possible utility that can be gained.

Figure 4.5: Near-optimality of the proposed algorithms.

request queue is tried to be matched with available peers. The ordering of the request queue is based on the utility-to-cost ratio for SRA_DP and SRA_GREEDY, and based on arrival time of the requests in FCFS. After matching requests to available peers, a subset of the remaining requests is selected to be served according to the employed seed server allocation algorithm: SRA_DP, SRA_GREEDY, or serving in the order of arrival (FCFS). In the fourth method, BT-like, a different procedure is employed: requests are first responded by the seed servers in an FCFS manner, then the remaining requests are responded by a set of up to 30 randomly selected peers who do have the requested data but might not have enough available capacity. Being randomly selected to be served by a seed server might result in quality fluctuation at the receiving peer. Thus, a request that is randomly selected for serving is continuously served at least for 1 minute.

4.5.2 Simulation Results

We first evaluate how close the utility gained by different allocation methods is to the optimal. Figure 4.5 depicts the result of this evaluation. The optimal utility in this figure is calculated as follows. At each time step, we assume a virtual seed server that is formed by aggregating the capacity of all peers as well as the actual seed server, ignoring all data availability constraints at peers. Therefore, the optimal utility that can be gained using the virtual seed server is greater than or equal to the optimal utility that can be actually

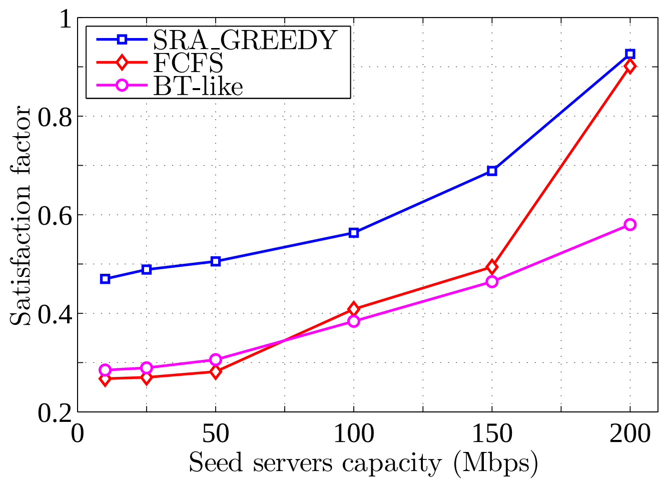


Figure 4.6: Satisfaction experienced by at least 90% of peers.

gained from the network. Since finding the optimal utility of the virtual seed server is an NP-complete problem (Theorem 3), we consider the optimal answer of the LP-relaxation of the problem, which is at least as large as the actual optimal utility; see Section 4.4.2. Thus, the optimal utility that we consider in Figure 4.5 is an upper bound on the maximum utility that can be gained in the network.

Figure 4.5(a) shows how close the two proposed algorithms are to the optimal. This figure depicts the utility gained by the system (normalized to $[0, 100]$) over time using different seed server allocation algorithms. Figure 4.5(b) illustrates the near-optimality of our proposed algorithms. In this figure, the proposed algorithms SRA_DP and SRA_GREEDY always gain beyond 90% of the optimal with an average of 95%. The seed server capacity is 10 Mbps in Figures 4.5(a) and 4.5(b). The theoretical approximation ratio for this case is 96%, while we see it has reached slightly lower values in practice. This is due to dynamics of the network that were not involved in the approximation analyses, i.e., the experimental ratio would have been always higher that 96% if all peers stayed in the network as expected, they let the tracker (and the tracker was able to) decide and update their partnerships at every 10-second step, and all peers did obey our assumptions about sharing their upload bandwidths among layers, which we intentionally made them disobey by deviating by up to 50% from their supposed values in Eq. (4.13). Figure 4.5 also depicts that the two approximation algorithms operate almost equally efficiently for a seed server capacity of 10

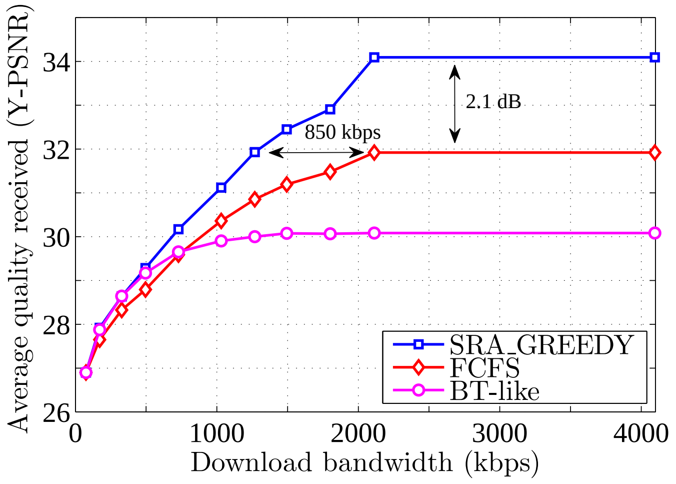


Figure 4.7: Video quality for peers with different download bandwidths.

Mbps; since the network consists of hundreds of peers we do not consider seeding capacities below 10 Mbps as such seed servers will not be realistic. For larger capacities, the dynamic algorithm takes a significant time to operate with reasonable approximation factor. Thus, we do not consider this method in the following evaluations as they deal with large seeding capacities.

We now evaluate the increase in the overall peer satisfaction, which is the fraction that a peer receives out of its demanded video quality. Figure 4.6 plots the satisfaction experienced by at least 90% of peers. The algorithm SRA_GREEDY considerably increase the satisfactions especially for limited seeding capacities, which is often the case in practice. Figure 4.6 also shows that for a very large seeding capacity such as 200 Mbps, which is nearly enough for fully satisfying all peers even with the FCFS method, the BitTorrent-like method still could not increase the satisfaction as expected. That is because this method followed a random peer matching, which caused inefficient utilization of peers resources.

Next, we evaluate the video quality delivered to peers. Figure 4.7 depicts the average Y-PSNR quality that peers with different download bandwidths could receive. The seed servers capacity is 25 Mbps in this figure. Without our algorithm, some higher quality levels could not be achieved at all (beyond 32 dB). The other quality levels would require peers to have a significantly larger bandwidth, e.g., beyond 2 Mbps for a quality of 32 dB whereas it is achieved by 1.2 Mbps by employing our algorithm. A quality increase of more

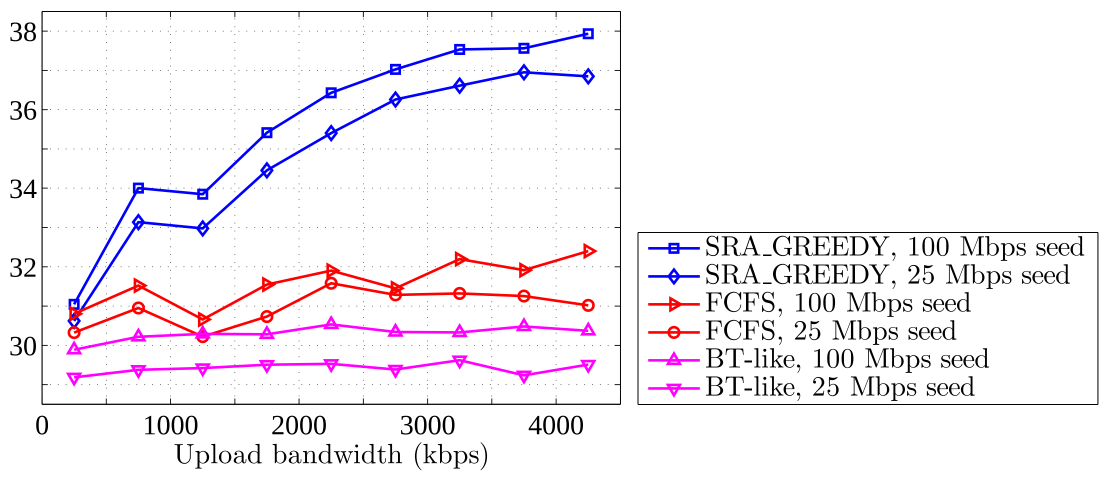


Figure 4.8: Incentive provided for peers to share upload bandwidth.

than 2 dB is obtained for most peers; the quality range of the considered scalable video is 13 dB in total.

We need to mention that with our algorithms a request is responded with a delay of up to the period between allocation algorithm runs. This period is typically a few seconds, but it is 10 seconds in our experiments; it needs to be this large so that our simulator, which is both in charge of allocation tasks as well as tasks of hundreds of peers, can simulate the network in reasonable time. We also mentioned that the computational cost of our algorithms, especially SRA_DP, is higher than simple algorithms such as FCFS and BT-like. Nevertheless, even our heavily-loaded simulator was able to perform these computations faster than real-time.

Our algorithms rely on knowing and utilizing the upload bandwidth of peers. This could be a weakness if peers are not cooperative. However, Figure 4.8 reveals how peers are encouraged to cooperate: more cooperation will bring them a significantly higher quality. This is because the utility function calculated in Eq. 4.17 assigns a higher score to cooperating peers (Section 4.4.3). For example, peers who shared 2 Mbps upload bandwidth received a video quality of 36 dB on average, which is 2 dB higher that the quality received by those contributing 1 Mbps, and 5 dB higher than those contributing 250 kbps. One might argue that this result is because peers with higher upload bandwidth also have higher download bandwidth, and naturally receive more video layers. We see, however, that for methods FCFS and BT-like this quality difference is marginal; the received quality is almost independent of the upload bandwidth. Thus, the higher quality achieved by our allocation algorithms is due peers' cooperation. With our algorithm compared to FCFS and BT-like algorithms, cooperating peers receive up to 7 dB higher quality, which is even more than half of the entire quality range of the video. This clearly shows the provided incentive for peers to cooperate as much as they can.

4.6 Summary

In this chapter, we have considered streaming of scalable videos over P2P networks. In these networks, due to the significant asymmetry between peers download and upload bandwidths, a number of seed servers need to be deployed in the network for delivering high-quality video streams to peers. We focused on the problem of allocating these seeding resources to the requests of peers for different substreams, in order to maximize a system-wide utility function. We formulated this problem and showed that it is NP-complete. We then proposed two approximation algorithms for the problem and proved that they produce near-optimal results. The first algorithm allocates seed servers based on dynamic programming and is preferred for limited seeding capacities ($\leq$ 10 Mbps). The second algorithm is designed for larger capacities and follows a greedy approach. We evaluated the proposed algorithms by simulating a P2P streaming system. The results of our evaluations confirm that the utility obtained by the proposed algorithms is always beyond 90% of the optimal utility that can be gained from the system. The results show also that the proposed seed server allocation algorithms result in peers receiving more video layers, and thus an enhanced video quality (over 2 dB). Moreover, our algorithms encourage peers to cooperate, as they provide a significantly higher video quality for those peers that upload more.

Chapter 5

Conclusions and Future Work

In this chapter, we summarize the contributions of this thesis and outline possible directions for future research.

5.1 Conclusions

In this thesis, we have studied two research problems related to our goal of adopting and taking best advantage of scalable video streams in today's multimedia streaming systems. The first problem is ensuring the authenticity of scalable video streams delivered over open and insecure networks, such as the Internet. The second problem is efficient management of resources for serving scalable video streams in P2P streaming systems.

For the former problem, we first analyzed and compared the most important solutions proposed in the literature for the problem of authenticating multimedia streams. We carried out numeric analysis and simulations for the schemes to study their performance along different performance metrics, and derived conclusions on choosing the most suitable authentication scheme for a given streaming application. For nonscalable streams, we found that the scheme proposed in [44] imposes the least amount of communication overhead and achieves the best resilience to packet losses. On the other hand, the scheme proposed in [45] is very efficient in terms of computation cost, but it is designed for on-demand streaming and is not applicable to live streaming scenarios. Furthermore, based on our analysis, we proposed a new authentication scheme for on-demand streaming that combines the advantages of these two schemes [44,45], and is the best performing one for on-demand streaming applications. For scalable streams, we found that authentication schemes based on hash trees are more efficient in terms of communication overhead. However, they require that stream adaptation proxies involved in the delivery process be compatible with the authentication scheme, which may not be desirable. For end-to-end streaming scenarios, on the other hand, authentication schemes based on hash chaining are more efficient. Nevertheless, we found that current schemes for scalable video streams fail to support the full flexibility of recent scalable streams.

Then, we proposed a new authentication scheme for scalable video streams encoded using the three-dimensional scalability model. Our scheme enables verification of any possible substream extracted from the original stream, and is designed for end-to-end secure delivery scenarios, where any entity involved in the delivery process and possibly in adapting video streams does not have to be compatible with our scheme. Moreover, we proposed an additional algorithm for minimizing the communication overhead incurred by an authentication scheme, which could reduce the overhead by more than 50% in our experiments. This algorithm can also be used to minimize the overhead of other authentication schemes, which are designed for traditional scalable videos. We conducted a simulation study with real video traces to evaluate our authentication scheme, which shows that our scheme is robust against reasonable packet loss rates ($< 20\%$), incurs negligible computational cost, adds only a short (1–2 sec) delay, requires no significant buffering (< 2 MB) by receives, and has low communication overhead, particularly after applying the overhead reduction algorithm. We also implemented the proposed authentication scheme in a prototype called svcAuth. svcAuth is available as an open source Java library and can be employed by any multimedia streaming application as a software add-on. It provides an end-to-end authentication service and allows receivers not supporting svcAuth to still receive and decode the streams, since it is transparent.

For the second problem, we formulated the problem of allocating the finite resources of seed servers to peers' requests for different video layers. We proved the NP-completeness of this problem, and proposed two approximation algorithms to solve it, which complement each other for a full spectrum of seeding capacities: the first algorithm allocates seed servers based on dynamic programming and is preferred for limited seeding capacities (≤ 10 Mbps), while the second algorithm is designed for larger capacities and follows a greedy approach. The results of evaluating our algorithms in a simulated P2P streaming system confirm that the utility obtained by the algorithms is always beyond 90% of the optimal utility that can be gained from the system. The proposed seed server allocation algorithms also result in peers receiving more video layers, and thus an enhanced video quality (over 2 dB). Moreover, our algorithms provide incentives for peers to cooperate in uploading, as the algorithms provide a significantly higher video quality for those peers that upload more.

5.2 Future Work

There are several open problems related to the topics discussed in this thesis. We summarize a number of them in this section and highlight some directions for future research.

Efficient support for limited-capability receivers. The proposed authentication scheme enables a receiver to verify all video packets, and minimizes the computation and communication overhead that need to be paid for achieving this goal. Although this is a desirable feature for many applications, we can further reduce the computation and/or communication overhead for limited-capability receivers by relaxing the security requirements for less security-critical applications, while still having a reasonably safe streaming session. For example, in a P2P streaming system, we can capitalize on the unique features of the P2P environment such as the large number of potential senders. Given many senders, a receiver can receive chunks of video data and authentication information from multiple intelligently-chosen senders and compare them. This will enable a limited-capability receiver to skip downloading some parts of the authentication information and avoid performing one signature verification for every few GoPs, while ensuring the authenticity with a high probability and detecting any attack in a short time. Note that in order for an attack on the video content to be meaningful, it needs to modify a continuous set of several video frames.

Enhancements to the seed resource allocation solutions. The proposed algorithms for allocation of seed servers are designed for being employed by a tracker that controls the seed servers. A desirable enhancement to this method is to extend these algorithms to be run on the seed servers themselves in a distributed manner, ensuring that the outcome is close to the tracker-based solution. In addition, in the proposed algorithms, the amount of computations on the tracker for each round of allocation is in proportion to the amount of requests. An efficient mechanism for regulating the rate of each peer's requests to the tracker, and accordingly, for avoiding any risk of a denial-of-service attack, can be of high interest.

Distribution of data availability information for scalable videos. The tracker-based approach for keeping track of peers' data availability information, which we assumed in our P2P streaming model, will require us to deploy a sufficient number of trackers in case of serving a large scale network of millions of concurrent users. On the other hand, arbitrary tracker-less methods for distributing this information, such as through *gossip* messages periodically generated and forwarded by peers, as in most of today's live P2P streaming systems (with nonscalable videos) [4], may not be efficient. This is because the data demanded by peers is no longer the same among all peers: in an on-demand streaming setting with scalable streams, each peer has a window of buffered data, e.g., the last 30 minutes it has played back, and possibly a different number of layers from each video segment in the buffer. Accordingly, finding a potential sender for a data segment, which happens to be available only at a small fraction of peers, could be very inefficient without the help of a tracker. Thus, a hybrid method for efficiently distributing data availability information by peers and trackers, while incurring a small load on trackers, can be very useful. This includes algorithms for determining when and how to send an update report to trackers and when to refer to them, an appropriate gossiping protocol, and algorithms for compressing a buffer map before reporting it to other peers. Note that a buffer map may consist of a large number of video segments, each with a different number of layers in different scalability dimensions.

Capacity provisioning for P2P streaming systems with scalable videos. The seed server allocation algorithms proposed in this thesis help us decide how to best utilize a given amount of seeding capacity. On the other hand, it can be very useful if we can estimate beforehand that with a given seeding capacity, what the expected throughput of the P2P streaming system will be, e.g., the average video quality served in the network, and the number of peers that can be served. Accordingly, we can determine the amount of seeding capacity that we need to provision for the network in order to achieve a desired level of video quality. To answer these questions, an analytical model is needed for forecasting the behavior of a given P2P streaming system. For example, as inputs to the analysis, we provide the characteristics of peers and video streams, the seeding capacity, and the employed seed allocation method, and we obtain the expected video quality that peers will receive.

Taking best advantage of the three-dimensional scalability model. The three-dimensional scalability model provides a high degree of flexibility for deciding what substream to receive using a given limited bandwidth. However, we need to carefully determine which layers in these dimensions we should download in order to achieve the highest perceptual quality, while meeting the bandwidth constraint. A similar problem, which is for traditional stream transcoders to choose among different adaptation possibilities, is studied by several previous works, e.g., [86, 87]. This is often done either by a subjective study to find a global solution for multi-dimensional stream adaptation, or by expensive retrieval of advanced information from the video content, such as the amount of motion or spatial details—note that deciding the best adaptations according to these information is still a subjective issue. Interestingly, H.264/SVC allows each client on its own to adapt the substream it receives: a client can request an ordered list of temporal, spatial, and quality layers according to their importance for the user, where the preferences of the user are known by the client software. This ability should be properly taken advantage of to best allocate a user's limited download bandwidth for receiving various layers, according to the bitrate and the importance of each combination of layers. An appropriate solution to this problem can be of interest for other streaming challenges as well, such as requesting H.264/SVC streams from multiple heterogeneous receivers or unequal loss protection of video layers. Moreover, previous subjective studies are limited to a small number of subjects, whereas having each user in a multimedia streaming system decide its own preference will result in a rich data set, and the possibility for an accurate universal quality metric.

Receiving scalable video streams from heterogeneous and dynamic senders. Suppose a P2P streaming system with scalable videos, where each peer has a list of potential senders. Each sender has a set of video layers available and an estimated throughput. The peer needs to employ a scheduling algorithm for downloading data chunks, which should maximize the received video quality and provide a smooth playback. This is an important problem in heterogeneous P2P streaming systems with scalable videos, and a simple form of it has been studied before for layered videos and for MPEG-4 Fine-Grained Scalable (FGS) videos in [18, 81]. In these works, it is assumed that the throughput of each sender is given as a certain number and that it does not change frequently. A peer continuously runs the proposed algorithms for requesting video pieces, and keeps measuring and updating the throughput of each sender. In a more practical scenario, however, a very dynamic sender with highly variable throughput should be differentiated from one with rather steady throughput: the former, even if having a higher mean value than the latter, is less preferred for being chosen to provide lower layers. That is because there is a chance that we need to throw away some higher video layers that we have already downloaded, since we could not receive their lower layers by their deadline, i.e., their playback time. Thus, an algorithm for requesting video data from heterogeneous senders, which takes into account these dynamics and gets the highest video quality while minimizing the chance of such cases can be very useful.

Bibliography

[1] The Insight Research Corporation. Streaming media, IPTV, and broadband transport: Telecommunications carriers and entertainment services 2006-2011, April 2006. `http://www.insight-corp.com/execsummaries/iptv06execsum.pdf`.

[2] RNCOS Industry Research Solutions. Global IPTV market analysis (2006-2010). Research Report, August 2006. http://www.rncos.com/Report/IM063.htm.

[3] Global Industry Analysts Inc. Video conferencing – A global strategic business report, May 2008. `http://www.strategyr.com/MCP-1062.asp`.

[4] J. Liu, S. Rao, B. Li, and H. Zhang. Opportunities and challenges of peer-to-peer internet video broadcast. *Proceedings of the IEEE*, 96(1):11–24, January 2008.

[5] B. Li and J. Liu. Multirate video multicast over the Internet: An overview. *IEEE Network*, 17(1):24–29, February 2003.

[6] M. Wien, H. Schwarz, and T. Oelbaum. Performance analysis of SVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1194–1203, September 2007.

[7] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, July 2003.

[8] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the scalable video coding extension of the H.264/AVC standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1103–1120, September 2007.

[9] T. Schierl, T. Stockhammer, and T. Wiegand. Mobile video transmission using scalable video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1204–1217, September 2007.

[10] O. Hillestad, A. Perkis, V. Genc, S. Murphy, and J. Murphy. Adaptive H.264/MPEG-4 SVC video over IEEE 802.16 broadband wireless networks. In *Proc. of Packet Video Workshop*, pages 26–35, Lausanne, Switzerland, November 2007.

[11] Institute of Electronics and Telecommunications Rennes (IETR-INSA). Open SVC Decoder, 2009. `http://sourceforge.net/projects/opensvcdecoder/`.

[12] Stretch Inc. Stretch H.264/SVC CODEC, 2008. `http://www.stretchinc.com/`.

[13] Y. Wu and R. Deng. Scalable authentication of MPEG-4 streams. *IEEE Transactions on Multimedia*, 8:152–161, February 2006.

[14] R. Kaced and J. Moissinac. Multimedia content authentication for proxy-side adaptation. In *Proc. of International Conference on Digital Telecommunications (ICDT'06)*, Cote d'Azur, France, August 2006.

[15] H. Yu. Scalable streaming media authentication. In *Proc. of IEEE International Conference on Communications(ICC'04)*, volume 4, pages 1912–1916, Paris, France, June 2004.

[16] R. Rejaie and A. Ortega. PALS: peer-to-peer adaptive layered streaming. In *Proc. of International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'03)*, pages 153–161, Monterey, CA, June 2003.

[17] X. Lan, N. Zheng, J. Xue, X. Wu, and B. Gao. A peer-to-peer architecture for efficient live scalable media streaming on Internet. In *Proc. of ACM Multimedia Conference (MM'07)*, pages 783–786, Augsburg, Germany, September 2007.

[18] M. Hefeeda and C. Hsu. Rate-distortion optimized streaming of fine-grained scalable video sequences. *ACM Transactions on Multimedia Computing, Communications and Applications*, 4(1):2:1–2:28, January 2008.

[19] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava. On peer-to-peer media streaming. In *Proc. of IEEE International Conference on Distributed Computing Systems (ICDCS'02)*, pages 363–371, Vienna, Austria, July 2002.

[20] M. Hefeeda and K. Mokhtarian. Analysis of authentication schemes for nonscalable video streams. In *Proc. of Packet Video Workshop (PV'09)*, pages 1–10, Seattle, WA, May 2009.

[21] M. Hefeeda and K. Mokhtarian. Authentication schemes for multimedia streams: Quantitative analysis and comparison. *ACM Transactions on Multimedia Computing, Communications and Applications*, 2009. Accepted to appear.

[22] K. Mokhtarian and M. Hefeeda. End-to-end secure delivery of scalable video streams. In *Proc. of International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'09)*, pages 79–84, Williamsburg, VA, June 2009.

[23] K. Mokhtarian and M. Hefeeda. Efficient allocation of seed servers in peer-to-peer streaming systems with scalable videos. In *Proc. of IEEE International Workshop on Quality of Service (IWQoS'09)*, Charleston, SC, July 2009.

[24] Y. Challal, H. Bettahar, and A. Bouabdallah. A taxonomy of multicast data origin authentication: Issues and solutions. *IEEE Communications Surveys and Tutorials*, 6(3):34–57, July 2004.

[25] A. Menezes, S. Vanston, and P. Van Oorschot. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, FL, 1st edition, 1996.

[26] National Institute of Standards and Technology (NIST). Federal Information Processing Standards (FIPS) publication 180: Secure Hash Standard, May 1993.

[27] R. Rivest. RFC1321; the MD5 message-digest algorithm. IETF, April 1992.

[28] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, Februrary 1978.

[29] National Institure of Standards and Technology (NIST). Federal Information Processing Standards (FIPS) publication 186: Digital Signature Standard (DSS), May 1994.

[30] R. Deng, D. Ma, W. Shao, and Y. Wu. Scalable trusted online dissemination of JPEG2000 images. *Multimedia Systems*, 11(1):60–67, November 2005.

[31] R. Grosbois, P. Gerbelot, and T. Ebrahimi. Authentication and access control in the JPEG2000 compressed domain. In *Proc. of SPIE Applications of Digital Image Processing XXIV*, volume 4472, pages 95–104, San Dieog, CA, December 2001.

[32] C. Peng, R. Deng, Y. Wu, and W. Shao. A flexible and scalable authentication scheme for JPEG2000 image codestreams. In *Proc. of ACM Multimedia Conference (MM'03)*, pages 433–441, Berkeley, CA, November 2003.

[33] B. Zhu, M. Swanson, and S. Li. Encryption and authentication for scalable multimedia: Current state of the art and challenges. In *Proc. of SPIE Internet Multimedia Management Systems V*, volume 5601, pages 157–170, Philadelphia, PA, October 2004.

[34] G. Sullivan and T. Wiegand. Video compression–from concepts to the H.264/AVC standard. *Proceedings of the IEEE*, 93(1):18–31, January 2005.

[35] R. Gennaro and P. Rohatgi. How to sign digital streams. In *Proc. of Advances in Cryptology (CRYPTO'97)*, volume 1294 of *LNCS*, pages 180–197, Santa Barbara, CA, August 1997. Springer-Verlag.

[36] P. Golle and N. Modadugu. Authenticating streamed data in the presence of random packet loss. In *Proc. of Network and Distributed System Security Symposium (NDSS'01)*, pages 13–22, San Diego, CA, February 2001.

[37] Z. Zhang, Q. Sun, and W. Wong. A proposal of butterfly-graph based stream authentication over lossy networks. In *Proc. of IEEE International Conference on Multimedia and Expo (ICME'05)*, pages 784–787, Amsterdam, The Netherlands, July 2005.

[38] Z. Zhishou, Q. Apostolopoulos, J.and Sun, S. Wee, and W. Wong. Stream authentication based on generalized butterfly graph. In *Proc. of IEEE International Conference on Image Processing (ICIP'07)*, volume 6, pages 121–124, San Antonio, TX, September 2007.

[39] C. Wong and S. Lam. Digital signatures for flows and multicasts. *IEEE/ACM Transactions on Networking*, 7(4):502–513, August 1999.

[40] R. Merkle. A certified digital signature. In *Proc. of Advances in Cryptology (CRYPTO'89)*, volume 435 of *LNCS*, pages 218–238, Santa Barbara, CA, August 1989. Springer-Verlag.

[41] J. Park, E. Chong, and H. Siegel. Efficient multicast stream authentication using erasure codes. *ACM Transactions on Information and System Security*, 6(2):258–285, May 2003.

[42] M. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348, April 1989.

[43] Y. Park and Yookun Cho. The eSAIDA stream authentication scheme. In *Proc. of International Conference on Computational Science and Its Applications (ICCSA'04)*, volume 3046 of *LNCS*, pages 799–807, Assisi, Italy, May 2004.

[44] A. Pannetrat and R. Molva. Efficient multicast packet authentication. In *Proc. of Network and Distributed System Security Symposium (NDSS'03)*, San Diego, CA, February 2003.

[45] A. Habib, D. Xu, M. Atallah, B. Bhargava, and J. Chuang. A tree-based forward digest protocol to verify data integrity in distributed media streaming. *IEEE Transactions on Knowledge and Data Engineering*, 17(7):1010–1014, July 2005.

[46] A. Perrig, R. Canetti, J. Tygar, and D. Song. Efficient authentication and signing of multicast streams over lossy channels. In *Proc. of IEEE Symposium on Security and Privacy (S&P'00)*, pages 56–73, Berkeley, CA, May 2000.

[47] A. Perrig, R. Canetti, D. Song, and J. Tygar. Efficient and secure source authentication for multicast. In *Proc. of Network and Distributed Systems Security Symposium (NDSS'01)*, pages 35–46, San Diego, CA, February 2001.

[48] C. Liang, A. Li, and X. Niu. Video authentication and tamper detection based on cloud model. In *Proc. of Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIHMSP'07)*, volume 1, pages 225–228, Splendor Kaohsiung, Taiwan, November 2007.

[49] A. Sun, D. He, Z. Zhang, and Q. Tian. A secure and robust approach to scalable video authentication. In *Proc. of IEEE International Conference on Multimedia and Expo (ICME'03)*, volume 2, pages 209–212, Baltimore, MD, July 2003.

[50] Q. Sun, S. Chang, M. Kurato, and M. Suto. A new semi-fragile image authentication framework combining ECC and PKI infrastructure. In *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS'02)*, volume 2, pages 440–443, Phoenix-Scottsdale, AZ, May 2002.

[51] C. Lin and S. Chang. Semi-fragile watermarking for authenticating JPEG visual content. In *Proc. of SPIE Security and Watermarking of Multimedia Content II*, volume 3971, pages 140–151, SanJose, CA, January 2000.

[52] P. Atrey, W. Yan, and M. Kankanhalli. A scalable signature scheme for video authentication. *Multimedia Tools and Applications*, 34:107–135, July 2007.

[53] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, November 1979.

[54] W. Wang and H. Farid. Exposing digital forgeries in video by detecting double MPEG compression. In *Proc. of ACM Multimedia and Security Workshop*, pages 37–47, Geneva, Switzerland, September 2006.

[55] S. Wenger, M. Hannuksela, T. Stockhammer, M. Westerlund, and D. Singer. RFC 3984; RTP payload format for H.264 video. IETF, February 2005.

[56] P. Argyroudis, R. Verma, H. Tewari, and D. O'Mahony. Performance analysis of cryptographic protocols on handheld devices. In *Proc. of IEEE International Symposium on Network Computing and Applications (NCA'04)*, pages 169–174, Cambridge, MA, September 2004.

[57] S. Tillich and J. Groschdl. A survey of public-key cryptography on J2ME-enabled mobile devices. In *Proc. of International Symposium on Computer and Information Sciences (ISCIS'04)*, volume 3280 of *LNCS*, pages 935–944, Antalya, Turkey, October 2004.

[58] V. Klima. Finding MD5 collisions - a toy for a notebook. Cryptology ePrint Archive: Report 2005/075, March 2005.

[59] V. Klima. Tunnels in hash functions: MD5 collisions within a minute. Cryptology ePrint Archive: Report 2006/105, April 2006.

[60] M. Yajnik, S. Moon, J. Kurose, and D. Towsley. Measurement and modeling of the temporal dependence in packet loss. In *Proc. of IEEE INFOCOM'99*, volume 1, pages 345–352, New York, NY, March 1999.

[61] D. Skraparlis. Design of an efficient authentication method for modern image and video. *IEEE Transactions on Consumer Electronics*, 49(2):417–426, May 2003.

[62] T. Li, H. Zhu, and Y. Wu. Multi-source stream authentication framework in case of composite MPEG-4 stream. In *Proc. of International Conference on Information and*

Communications Security (ICICS'05), volume 3783 of *LNCS*, pages 389–401, Beijing, China, December 2005. Springer.

[63] R. Kaced and J. Moissinac. SEMAFOR: A framework for authentication of adaptive multimedia content and delivery for heterogeneous networks. In *Proc. of International Conference on Internet Surveillance and Protection (ICISP'06)*, page 28, Cote d'Azur, France, August 2006.

[64] T. Suzuki, Z. Ramzan, H. Fujimoto, C. Gentry, T. Nakayama, and R. Jain. A system for end-to-end authentication of adaptive multimedia content. In *Proc. of IFIP Conference on Communications and Multimedia Security (IFIP CMS'04)*, volume 175 of *LNCS*, pages 237–249, Windermere, The Lake District, United Kingdom, September 2004. Springer.

[65] C. Gentry, A. Hevia, R. Jain, T. Kawahara, and Z. Ramzan. End-to-end security in the presence of intelligent data adapting proxies: The case of authenticating transcoded streaming media. *IEEE Journal on Selected Areas in Communications*, 23(2):464–473, February 2005.

[66] J. Apostolopoulos. Architectural principles for secure streaming & secure adaptation in the developing scalable video coding (SVC) standard. In *IEEE International Conference on Image Processing (ICIP'06)*, pages 729–732, Atlanta, GA, October 2006.

[67] S. Park and S. Shin. Combined scheme of encryption and watermarking in H.264/Scalable Video Coding (SVC). In *New Directions in Intelligent Interactive Multimedia*, volume 142 of *Studies in Computational Intelligence*, pages 351–361. Springer, September 2008.

[68] R. Iqbal, S. Shirmohammadi, A. El-Saddik, and J. Zhao. Compressed-domain video processing for adaptation, encryption, and authentication. *IEEE Multimedia*, 15(2):38–50, April 2008.

[69] I. Amonou, N. Cammas, S. Kervadec, and S. Pateux. Optimized rate-distortion extraction with quality layers in the scalable extension of H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 17:1186–1193, September 2007.

[70] S. Wenger. H.264/AVC over IP. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):645–656, July 2003.

[71] Y. Wang, L. Chau, and K. Yap. GOP-based unequal error protection for scalable video over packet erasure channel. In *Proc. of IEEE Symposium on Broadband Multimedia Systems and Broadcasting (BMSB'08)*, pages 1–4, Las Vegas, NV, April 2008.

[72] D. Bakker, D. Cromboom, T. Dams, A. Munteanu, and J. Barbarien. Priority-based error protection for the scalable extension of H.264/SVC. In *In Proc. of SPIE Conference*

on Optical and Digital Image Processing, volume 7000 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 7000H–1 – 7000H–11, Strasbourg, France, April 2008.

[73] Y. Zhang and N. Duffield. On the constancy of Internet path properties. In *Proc. of ACM SIGCOMM Workshop on Internet Measurement*, San Francisco, CA, November 2001.

[74] T. Wiegand, G. Sullivan, J. Reichel, H. Schwarz, and M. Wien. Joint draft 11 of SVC amendment. Joint Video Team (JVT), Doc. JVT-X201, July 2007.

[75] PPLive. `http://www.pplive.com/en/index.html`.

[76] SopCast. `http://www.sopcast.org/`.

[77] TVAnts. `http://www.tvants.com/`.

[78] X. Zhang, J. Liu, B. Li, and T. Yum. CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming. In *Proc. of IEEE INFOCOM'05*, pages 2102–2111, Miami, FL, March 2005.

[79] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross. A measurement study of a large-scale P2P IPTV system. *IEEE Transactions on Multimedia*, 9(8):1672–1687, December 2007.

[80] Y. Liu, Y. Guo, and C. Liang. A survey on peer-to-peer video streaming systems. *Peer-to-Peer Networking and Applications*, 1(1):18–28, March 2008.

[81] Y. Cui and K. Nahrstedt. Layered peer-to-peer streaming. In *Proc. of International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'03)*, pages 162–171, Monterey, CA, June 2003.

[82] M. Zink, O. Kunzel, J. Schmitt, and R. Steinmetz. Subjective impression of variations in layer encoded videos. In *Proc. of IEEE International Workshop on Quality of Service (IWQoS'03)*, pages 137–154, Berkeley, CA, June 2003.

[83] X. Xiao, Y. Shi, and Y. Gao. On optimal scheduling for layered video streaming in heterogeneous peer-to-peer networks. In *Proc. of ACM Multimedia Conference (MM'08)*, pages 785–788, Vancouver, BC, Canada, October 2008.

[84] R. Rajendran and D. Rubenstein. Optimizing the quality of scalable video streams on P2P networks. *Computer Networks*, 50(15):2641–2658, October 2006.

[85] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems.* Springer, 2004.

[86] Y. Wang, J. Kim, S. Chang, and H. Kim. Utility-based video adaptation for universal multimedia access (UMA) and content-based utility function prediction for real-time video transcoding. *IEEE Transactions on Multimedia*, 9(2):213–220, February 2007.

[87] M. Barzilay, J. Taal, and R. Lagendijk. Subjective quality analysis of bit rate exchange between temporal and SNR scalability in the MPEG4 SVC extension. In *IEEE International Conference on Image Processing (ICIP'07)*, volume 2, pages 285–288, San Antonio, TX, September 2007.