Report Number: WUCSE-2005-40

2005-08-08

Efficient and Effective Schemes for Streaming Media Delivery

Cheng Huang

The rapid expansion of the Internet and the increasingly wide deployment of wireless networks provide opportunities to deliver streaming media content to users at anywhere, anytime. To ensure good user experience, it is important to battle adversary effects, such as delay, loss and jitter. In this thesis, we first study efficient loss recovery schemes, which require pure XOR operations. In particular, we propose a novel scheme capable of recovering up to 3 packet losses, and it has the lowest complexity among all known schemes. We also propose an efficient algorithm for array codes decoding, which achieves significant throughput gain... Read complete abstract on page 2.

Efficient and Effective Schemes for Streaming Media Delivery

Cheng Huang

Complete Abstract:

The rapid expansion of the Internet and the increasingly wide deployment of wireless networks provide opportunities to deliver streaming media content to users at anywhere, anytime. To ensure good user experience, it is important to battle adversary effects, such as delay, loss and jitter. In this thesis, we first study efficient loss recovery schemes, which require pure XOR operations. In particular, we propose a novel scheme capable of recovering up to 3 packet losses, and it has the lowest complexity among all known schemes. We also propose an efficient algorithm for array codes decoding, which achieves significant throughput gain and energy savings over conventional codes. We believe these schemes are applicable to streaming applications, especially in wireless environments. We then study quality adaptation schemes for client buffer management. Our control-theoretic approach results in an efficient online rate control algorithm with analytically tractable performance. Extensive experimental results show that three goals are achieved: fast startup, continuous playback in the face of severe congestion, and maximal quality and smoothness over the entire streaming session. The scheme is later extended to streaming with limited quality levels, which is then directly applicable to existing systems.

2005-40

Efficient and Effective Schemes for Streaming Media Delivery, Doctoral Dissertation, August 2005

Authors: Huang, Cheng

August 8, 2005

Abstract: The rapid expansion of the Internet and the increasingly wide deployment of wireless networks provide opportunities to deliver streaming media content to users at anywhere, anytime. To ensure good user experience, it is important to battle adversary effects, such as delay, loss and jitter.

In this thesis, we first study efficient loss recovery schemes, which require pure XOR operations. In particular, we propose a novel scheme capable of recovering up to 3 packet losses, and
it has the lowest complexity among all known schemes. We also propose an efficient algorithm for array codes decoding, which achieves significant throughput gain and energy savings over conventional codes. We believe these schemes are applicable to streaming applications, especially in wireless environments.

We then study quality adaptation schemes for client buffer management. Our control-theoretic approach results in an efficient online rate control algorithm with analytically tractable performance. Extensive experimental results show that three goals are achieved: fast startup, continuous playback in the face of severe congestion, and maximal quality and smoothness over the entire streaming session. The scheme is later extended to streaming with limited quality levels, which is then directly applicable to existing systems.

SEVER INSTITUTE OF TECHNOLOGY

DOCTOR OF SCIENCE DEGREE

DISSERTATION ACCEPTANCE

(To be the first page of each copy of the dissertation)

DATE: May 10, 2005

STUDENT'S NAME: Cheng Huang

This student's dissertation, entitled Efficient and Effective Schemes for Streaming Media Delivery has been examined by the undersigned committee of five faculty members and has received full approval for acceptance in partial fulfillment of the requirements for the degree Doctor of Science.

APPROVAL: ______________________ Chairman

____

____

____

____

WASHINGTON UNIVERSITY

SEVER INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

EFFICIENT AND EFFECTIVE SCHEMES FOR STREAMING MEDIA DELIVERY

by

Cheng Huang, M.Sc.

Prepared under the direction of Professor Lihao Xu

A dissertation presented to the Sever Institute of
Washington University in partial fulfillment
of the requirements for the degree of

Doctor of Science

August, 2005

Saint Louis, Missouri

WASHINGTON UNIVERSITY

SEVER INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

ABSTRACT

EFFICIENT AND EFFECTIVE SCHEMES FOR STREAMING MEDIA DELIVERY

by Cheng Huang

ADVISOR: Professor Lihao Xu

August, 2005

Saint Louis, Missouri

The rapid expansion of the Internet and the increasingly wide deployment of wireless networks provide opportunities to deliver streaming media content to users at anywhere, anytime. To ensure good user experience, it is important to battle adversary effects, such as delay, loss and jitter.

In this thesis, we first study efficient loss recovery schemes, which require pure XOR operations. In particular, we propose a novel scheme capable of recovering up to 3 packet losses, and it has the lowest complexity among all known schemes. We also propose an efficient algorithm for array codes decoding, which achieves significant throughput gain and energy savings over conventional codes. We believe these schemes are applicable to streaming applications, especially in wireless environments.

We then study quality adaptation schemes for client buffer management. Our control-theoretic approach results in an efficient online rate control algorithm with analytically tractable performance. Extensive experimental results show that three goals are achieved: fast startup, continuous playback in the face of severe congestion, and maximal quality and smoothness over the entire streaming session. The scheme is later extended to streaming with limited quality levels, which is then directly applicable to existing systems.

Dedication: To my family

Contents

List of Tables

List of Figures

Acknowledgments

I gratefully acknowledge the guidance of my advisor, Dr. Lihao Xu, and my mentors at Microsoft, Dr. Philip A. Chou and Mr. Anders E. Klemets. Most of this work is the result of my collaboration with them. I wish to thank Dr. Ramaprabhu Janakiraman for being a great collaborator and lab-mate. Parts of this work are the result of joint work with Dr. Janakiraman.

I am indebted to Dr. Jason E. Fritts, Dr. Chenyang Lu, and Dr. Joseph A. O'Sullivan for serving on my dissertation committee, and for insightful discussions and helpful advice. I also take this chance to thank Ms. Fuller, Ms. Grothe, Ms. Harbison, Ms. Matlock, and Ms. Sung – the administrative assistants in the department of Computer Science at Washington University – for making my (and countless others') graduate student life so much easier.

Cheng Huang

Washington University in Saint Louis
August 2005

Chapter 1

Introduction

1.1 Challenges

The rapid expansion of the Internet and the increasingly wide deployment of wireless networks provide opportunities to serve multimedia content to users anywhere, anytime. Compression and delivery are the two major components of streaming media applications. In this thesis, we will focus on delivery techniques for streaming media over today's best-effort networks, with the goal of providing a better overall user experience for clients.

The fundamental difficulties that distinguish streaming media delivery from traditional bulk data downloading (e.g., file transfer), stem from the real-time constraints introduced by the former. It is now well-understood that the performance of media streaming is greatly impacted by end-to-end transmission delays, packet loss, and congestion [1]. These adverse effects may happen due to one or more of the following reasons:

1. Packets may get corrupted in transit, especially in wireless networks.
2. Current networks operate under best-effort policies and seldom provide guaranteed *Quality-of-Service* (QoS). Competing traffic can contribute to queuing delays and packet drops in routers, severely impacting streaming media flows.
3. Signal interference and user mobility can cause fluctuations in reception quality on wireless networks.

Although error concealment techniques [2] can be used to mitigate the perceived consequences of these pathologies, these techniques have limited impact, and often the net effect is a degradation of the user experience. Therefore we must look beyond mere error concealment towards more effective strategies.

Streaming media delivery systems are usually modeled using a client-server architecture. A server sends out media data packets, which traverse intermediate inter-connected machines to arrive at a client. The client stores received packets in a buffer before consuming them, and optionally may also use packet arrival patterns to infer the state of the network. Buffering by the client is a very useful technique to overcome transient delay and jitter. It can also be used to buy more time in which to request retransmission of lost packets.

Since client buffering introduces extra delay before playback, it might not be suitable for some applications with very strong delay constraints (or strict low latency requirements), like streaming of live events, conferencing, surveillance, etc. In these applications, the lack of client buffering means that often retransmissions of lost packets will arrive too late to be useful. Similarly, in multicast or broadcast applications, different clients may experience different loss patterns, increasing the overhead of retransmission-based loss recovery.

For all these applications, one technique is *Forward Error Correction* (FEC), in which the server sends redundant *parity* information along with the original information. Note that in packet networks like the Internet, data integrity is verified through separate mechanisms like checksums, so that corrupted packets are silently discarded upon detection. Hence, on such networks FEC techniques are mainly used to recover from packet losses or *erasures*, not arbitrary corruptions or *errors*. FEC permits recovery from limited packet loss without the additional latency introduced by negative acknowledgments and retransmission of lost packets, but with the additional cost in network bandwidth incurred by transmitting parity information.

The efficacy of an FEC scheme depends on the extent of loss recovery possible for a given amount of redundant parity information, which in turn depends on *exactly how the parity information is constructed.* One technique is to construct parity packets as carefully chosen linear combinations of the original data packets, where the arithmetic operations on packet data are done over a *finite field*. A popular example is to construct the parity packets using *Reed-Solomon codes* [3]. Reed-Solomon codes belong to a class of codes called Maximum Distance Separable (MDS) codes [4], which provide optimal erasure recovery capability – loosely speaking, if such codes are used, each additional parity packet permits the recovery of one additional lost packet. The disadvantage of Reed-Solomon codes, and in general, codes defined over finite fields, is that on general-purpose hardware like PCs, arithmetic operations on finite fields tend to be slower than integer arithmetic or boolean logic. Thus FEC techniques based on these codes might not be suitable for applications involving *constrained clients* – clients with low-end processors or limited battery life, like small portable devices.

More computationally efficient schemes are desirable for these applications. Previous research has suggested the use of exclusive-OR (*XOR*) based codes instead of finite field-based codes to implement computationally efficient FEC schemes. The primary disadvantage of using such codes is that XOR-based codes with the above MDS property are known only for limited values of the code parameters (like one or two parity packets.) Conversely, XOR-based codes for generating more parity packets do not possess the MDS property. This motivates the need for new XOR-based MDS codes capable of efficient encoding/decoding, and also tolerating more packet erasures.

For applications with more relaxed delay constraints, such as streaming media on demand, client buffering can obviate the need for FEC techniques for loss recovery; simpler retransmission-based techniques may suffice. However, to ensure continuous playback under varying network conditions, it is important to maintain a large enough client buffer, thereby reducing the risk of buffer underflow.

The buffer size is regulated by the rate of data *inflow* from the network, and the rate of data *outflow* to the client's media decoder. The inflow rate is determined by external factors like transport protocols, flow and congestion control mechanisms, competing traffic, network signal strength, etc. The outflow rate is determined by the quality (or bit rate) of the content. Variation in the inflow rate impacts the ability of the client to maintain a desired buffer size at any given time. Thus, to achieve the latter goal, it is necessary to vary the outflow rate and consequently the quality of content in accordance with the inflow rate.

Indeed, various schemes [5–13] have been proposed to take advantage of the ability to adapt the coding rate of media data. Qualities are adjusted on the fly during a streaming session to manage the client buffer. However, these approaches either assume *a priori* knowledge of network variations, making them impractical, or deal with only a few choices of qualities, limiting their applicability to media content with finer adaptive capability. Therefore, the primary challenge is to develop quality adaptation schemes with arbitrary granularity, which maintain client buffer level and also achieve better quality, under severe network variations that are unknown *a priori*.

1.2 Contributions

Our contributions towards improving the delivery of streaming media are two-fold.

First, we study efficient FEC schemes for streaming media delivery using a special class of error correcting codes, called *array codes* [14]. Array codes in general arrange data in a two dimensional array and use only XOR operations for encoding and decoding. Our starting point is a well-known MDS array code, the EVENODD code [15], which is capable of recovering up to two packet losses in a single coded block.

Extending from the EVENODD code, we have designed a new MDS array code called **STAR**. The STAR code is an MDS array code capable of tolerating three packet losses in a single coded block. By exploiting the geometric property of the code, we have also developed an efficient decoding algorithm. Our analysis shows that the STAR decoding is much more computationally efficient than comparable codes [16–19], especially when the block length is small. This makes the STAR code especially attractive for streaming applications with strong delay constraints.

MDS array codes can achieve optimal loss recovery performance when each column in an array is treated as a packet of streaming media data. However, this limits the code block length to small values and is thus most applicable to strictly low latency streaming applications. When the delay constraint is relatively weak, schemes with larger block lengths tend to have better recovery performance for a fixed coding rate. Our second contribution in the design of FEC schemes is to propose **XEOD**, an efficient scheme for bit level decoding of array codes. Both theoretical analysis and simulation measurements show that the XEOD scheme has significant throughput gain (and energy savings for portable devices) compared to the Reed-Solomon code, while achieving comparable loss recovery – especially under bursty packet loss patterns prevalent in the Internet.

In addition to packet loss, congestion and jitter are also major issues in media streaming. Our primary contribution to overcome these issues is through new quality adaptation schemes. We have designed a client buffer management scheme called Optimal Rate Control (**ORC**), in which the problem of quality adaptation is formulated as a standard problem in linear quadratic optimal control with the objective of maintaining a target buffer size profile. This control-theoretic approach results in an efficient online rate control algorithm with analytically tractable performance. To our knowledge this is the first use of optimal control theory for client buffer management. Also, we explicitly take into consideration, using a leaky bucket model, the natural variation in content bit rate to achieve smooth variation in user-perceived quality. To our knowledge this is also the first use of a leaky bucket to model source coding rate constraints during client buffer management beyond the initial startup delay. Extensive experimental results show that three goals are achieved: fast startup, continuous playback under severe congestion, and satisfactory quality and smoothness over the entire streaming session. Also, our algorithm complements any transport protocol, and we show that it works effectively with both TCP and TFRC transport protocols.

We have also extended the ORC scheme to *Multi Bit Rate* (**MBR**) streaming, where the server is forced to choose from a limited set of rates. Compared to existing schemes in commercial systems, our scheme demonstrates more stability and effectiveness in overcoming severe network congestion.

1.3 Organizations

This thesis is organized as follows: in Chapter 2, we describe STAR, an MDS array code for triple erasure recovery, and illustrate our decoding algorithm based on the geometric construction of the STAR code. We describe the XEOD scheme in Chapter 3 and present its significant throughput benefit and energy savings for wireless streaming applications.

We describe ORC, an optimal coding rate control scheme, in Chapter 4 and its extension for MBR streaming in Chapter 5. We conclude in Chapter 6 with a summary of our contributions and an outline of future research directions.

Chapter 2

STAR: An Efficient Scheme for Triple Erasure Recovery

As a technique to battle data loss in streaming media delivery, FEC sends redundant *parity* packets along with the original information packets. When losses occur in the network, a recovery procedure is invoked to obtain the original media data. FEC permits recovery from limited packet losses without the additional latency introduced by negative acknowledgments and retransmission of lost packets, but with the additional cost in network bandwidth incurred by transmitting the parity packets.

The efficacy of an FEC scheme depends on the extent of loss recovery possible for a given amount of redundant parity packets, which in turn depends on exactly how the parity packets are constructed. One technique is to construct parity packets as carefully chosen linear combinations of the original data packets, where the arithmetic operations on packet data are done over a *finite field*. A popular example is to construct the parity packets using *Reed-Solomon codes* [3]. Reed-Solomon codes belong to a class of codes, which provide optimal erasure recovery capability – loosely speaking, if such codes are used, each additional parity packet permits the recovery of one additional lost packet. The disadvantage of the Reed-Solomon code, and in general, codes defined over finite fields, is that on general-purpose hardware like PCs, arithmetic operations on finite fields tend to be slower than integer arithmetic or boolean logic. Thus FEC techniques based on these codes might not be suitable for applications involving constrained clients – clients with low-end processors or limited battery life, like small portable devices.

More computationally efficient schemes are desirable for these applications. Previous research has suggested the use of XOR-based codes instead of finite field-based codes to implement computationally efficient FEC schemes. And in this work, we study using array codes as computationally efficient erasure recovery schemes. Array codes are a class of linear codes, where information and parity bits are placed in a two-dimensional array rather than a one-dimensional vector. An array code denoted by $m \times n$ corresponds to an array of m rows and n columns of *symbols*.

Symbols are defined over an Abelian group $G(q^t)$ with an addition operation $\oplus$. In particular, we are interested in the case of $q = 2$, e.g., each symbol has t binary bits and $\oplus$ is just simple bitwise exclusive-OR (*XOR*).

On the other hand, the array code can also be regarded as a one-dimensional code defined over the Abelian group $G((q^t)^m)$, by regarding each column as a single element. Then, the minimum distance d of the code can also be defined over $G((q^t)^m)$. Let N be the number of its codewords, then the dimension k of the array code can be defined as $k = log_{(q^t)^m} N$, as for usual one-dimensional codes. The array code can now be viewed as an $(n,\ k,\ d)$ code over $G((q^t)^m)$. In this chapter, we consider streaming media data packets of size mt bits each. Then, each packet maps to one column of the array code and a packet loss is correspondingly represented as a column erasure. If it satisfies the property that $d = n-k+1$, then the code achieves the Singleton bound [4] and is called an MDS array code. When the MDS array code is used as an FEC scheme, the original media data can always be recovered with up to $n - k$ packet losses. For simplicity, we will assume symbol size $t = 1$ in this thesis. However, all results hold for arbitrary t.

There exist MDS array codes. For instance, the simplest PARITY code can be regarded as a $(k + 1,\ k,\ 2)$ MDS array code over $G(q^m)$, if each column contains m symbols. The only parity column is generated as the XOR sum of all other k information (data) columns. It is clear that this scheme can recover from arbitrary *single* erasure. MDS array codes for *double* erasure recovery have been proposed, such as the EVENODD [15], the B-Code [20], the X-Code [21], the DH1 and DH2 [22], etc. All these schemes satisfy $n = k + 2$, $d = 3$ and can thus recover from arbitrary double erasures.

For *triple* erasures, MDS array codes have also been studied. In particular, Tau and Wang [16] propose the HDD1 and HDD2 schemes. ([17] is an unsuccessful attempt to handle multiple erasures, which we do not discuss in details here.) Both schemes claim satisfying $n = k + 3$ and $d = 4$. Their encodings are efficient, by requiring exactly or slightly more than 3 XORs per symbol (the total number of XORs normalized by the total number of information symbols). Note that 3 is the minimum number of XORs needed for triple parities, thus these schemes all have good encoding performance. However, the decodings of these schemes resort to techniques essentially similar to Gaussian eliminations to solve unknown elements in a set of linear equations and tend to require more than 9 XORs per symbol (on average). The gap between the encoding and decoding complexity is fairly significant for these schemes. Blaum *et al.* [18, 19] generalize the EVENODD and propose an MDS code for multiple erasures. The construction of the Blaum code conforms to a special structure, which is then exploited for an efficient decoding algorithm. As a result, the decoding complexity is reduced and asymptotically approaches 3 XORs per symbol now. However, when k is limited, the decoding complexity of the Blaum code deviates from its asymptotic bound fairly significantly. For example, the complexity is about 5 XORs per symbol when $k = 11$. The number is even bigger with a smaller k. On the other hand, streaming applications with strong delay

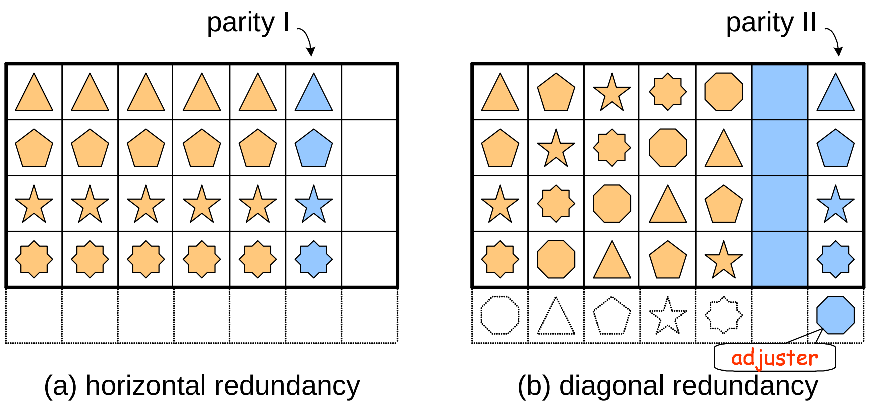


Figure 2.1: EVENODD Code Encoding

constraints tend to use small coding block lengths. Thus, the decoding complexity for small ks is critical to these applications, and it is desirable to seek schemes that perform well in this region.

In this chapter, we describe STAR, an efficient scheme for triple erasure recovery. The STAR code is an alternative extension of the EVENODD code and has the same encoding complexity as the Blaum code. Our key contribution is to exploit the geometric property of the code construction, which then leads to an efficient decoding algorithm. Our analysis shows that the decoding complexity remains at slightly more than 3 XORs per symbol, even for small ks. This makes the STAR code especially attractive for streaming applications with strong delay constraints.

2.1 EVENODD Code: Double Erasure Recovery

2.1.1 EVENODD Code and Encoding

We first briefly describe the EVENODD code [15], which was initially proposed to address disk failures in disk array systems. Data from multiple disks form a two dimensional array, with one disk corresponding to one column of the array. A disk failure is equivalent to a column erasure. The EVENODD code uses two parity columns together with p information columns (where p is a prime number). The code ensures that all information columns are fully recoverable when *any* two disks fail. In this sense, it is an optimal 2-erasure correcting code, i.e., it is an $(p+2,\ p,\ 3)$ MDS code. Besides this MDS property, the EVENODD code is computationally efficient in both encoding and decoding, since only XOR operations are involved.

The encoding process considers a $(p-1)\times(p+2)$ array, where the first p columns are information columns and the last two parity columns. Symbol $a_{i,j}$ ($0 \le i \le p-2$, $0 \le j \le p+1$) represents symbol i in column j. A parity symbol in column p is computed as the XOR sum of all information symbols in the same row. And the computation of column $(p+1)$ takes the following steps. First, the array is augmented with an imaginary row $p-1$, where all symbols are assigned *zero* values (recall that symbols are defined over $G(2)$). The XOR sum of all information symbols along

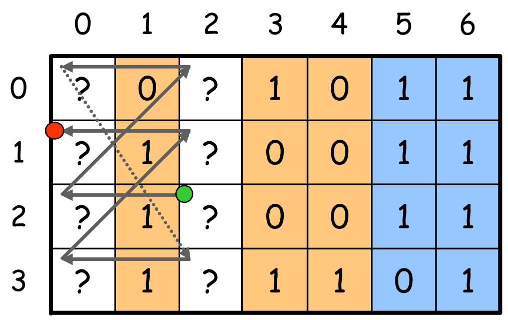


Figure 2.2: EVENODD Code Decoding

the same diagonal (indeed a diagonal with slope 1) is computed and assigned to their corresponding parity symbol, as marked by different shapes in Figure 2.1. Symbol $a_{p-1,p+1}$ now becomes non-zero and is called the EVENODD *adjuster*. To remove this symbol from the array, *adjuster complement* is performed, which adds (XOR addition) the adjuster to all symbols in column $p+1$. The encoding can be algebraically described as follows ($0 \leq i \leq p-2$):

$$a_{i,p} = \bigoplus_{j=0}^{p-1} a_{i,j}$$

$$a_{i,p+1} = S_1 \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle i-j \rangle_p, j} \right), \text{ where } S_1 = \bigoplus_{j=0}^{p-1} a_{\langle p-1-j \rangle_p, j}.$$

Here, S_1 is the EVENODD adjuster and $\langle x \rangle_p$ denotes $x \bmod p$. For more details, please refer to [15].

2.1.2 EVENODD Erasure Decoding

The EVENODD code is an optimal double erasure correcting code and any two column erasures in a coded block can be fully recovered. Regarding to the locations of the erasures, [15] divides decoding into four cases. Here, we only summarize the most common one, where neither of the erasures is a parity column. Note that the other three cases are special ones and can be dealt with easily. A decoder first computes horizontal and diagonal *syndromes* as the XOR sum of all available symbols along those directions. Then a *starting* point of decoding can be found, which is guaranteed to be the only erasure symbol in its diagonal. The decoder recovers this symbol and then moves horizontally to recover the symbol in the other erasure column. It then moves diagonally to the next erasure symbol and horizontally again. Upon completing this *Zig-Zag* process, all erasure symbols are fully recovered. In the example shown in Figure 2.2 ($p = 5$), the starting point is symbol $a_{2,2}$ and the decoder moves from $a_{2,2}$ to $a_{2,0}$, $a_{0,2}$, $a_{0,0} \cdots$ and finally completes at $a_{1,0}$.

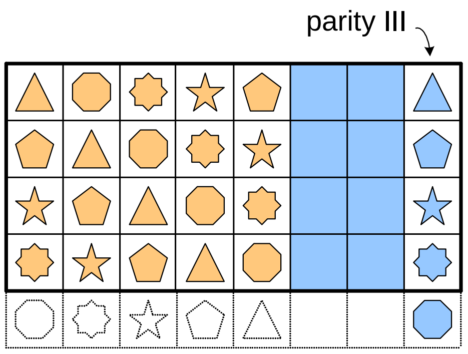


Figure 2.3: STAR Code Encoding

2.2 STAR Code Encoding: Geometric Description

As an extension of the EVENODD code, the STAR code consists of $p + 3$ total columns, where the first p columns contain information data and the last 3 columns contain parity data. It is a systematic code, similarly as the EVENODD code.

The STAR code encoding is also similar to the EVENODD code, where the parity columns p and $p + 1$ are computed from the horizontal and diagonal redundancy. And the parity column $p + 2$ is computed from another diagonal redundancy. This diagonal follows slope -1, as opposed to slope 1 when computing the parity column $p + 1$. For simplicity, we denote this as *anti-diagonal* redundancy. The procedure is depicted by Figure 2.3, where symbol $a_{p-1,p+2}$ in parity column $p + 2$ is also an *adjuster*, similar to the EVENODD code. And the adjuster is removed from the final code block by adjuster complement. Algebraically, the encoding of parity column $p + 2$ can be represented as $(0 \leq i \leq p - 2)$:

$$a_{i,p+2} = S_2 \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle i+j \rangle_p, j} \right), \text{ where } S_2 = \bigoplus_{j=0}^{p-1} a_{\langle j-1 \rangle_p, j}.$$

2.3 STAR Code Erasure Decoding

The essential part of the STAR code is the erasure decoding algorithm. As presented in this section, the decoding algorithm involves pure XOR operations, which allows efficient implementation and thus is suitable for computation/energy constrained applications. The MDS property of the STAR code, which guarantees the recovery from arbitrary triple erasures, is explained along with the description of the decoding algorithm. And a mathematical proof of this property will be given in a later section.

The STAR code decoding can be divided into two cases based on erasure patterns: 1) decoding without parity erasures, where all erasures are information columns; and 2) decoding with parity erasures, where at least one erasure is a parity column. The former case is the most common one and presents the essence of the decoding algorithm, thus it is the main focus of this section. Further, it can be divided into two subcases: symmetric and asymmetric, based on whether the erasure columns are evenly spaced. The latter case, on the other hand, handles several special situations and is consequently simpler.

2.3.1 Decoding without Parity Erasures: Asymmetric Case

We consider the recovery of triple information column erasures at position r, s and t ($0 \leq r,\ s,\ t \leq p-1$), among the total $p+3$ columns. Assume $r < s < t$ without loss of generality and let $u = s-r$ and $v = t-s$. Thus, the asymmetric case deals with erasure patterns satisfying $u \neq v$.

The decoding algorithm can be visualized with a concrete example, where $r = 0$, $s = 1$, $t = 3$ and $p = 5$, as shown in Figure 2.4(a). Empty columns denote erasures. And the decoding procedure consists of the following four steps:

Recover Adjusters and Calculate Syndromes

Given the definitions of the adjusters S_1 and S_2, it is easy to see that they can be computed as the XOR sums of all symbols in parity columns 5, 6 and 5, 7, respectively.

Then the adjusters are assigned to symbols $a_{4,6}$, $a_{4,7}$ and also applied through XOR additions to all of the rest parity symbols in columns 6, 7, which is to reverse the adjuster complement. The redundancy property of the coded block states that the XOR sum of all symbols along any parity direction (horizontal, diagonal and anti-diagonal) should equal to *zero*. Due to erasure columns, however, the XOR sum of rest symbols is non-zero and we denote it as the *syndrome* for this parity direction. To be specific, syndrome $\tilde{s}_{i,j}$ denotes the XOR sum of parity symbol $a_{i,j+p}$ and its corresponding non-erasure information symbols. For example, $\tilde{s}_{0,0} = a_{0,5} \oplus a_{0,2} \oplus a_{0,4}$ and $\tilde{s}_{0,1} = a_{0,6} \oplus a_{3,2} \oplus a_{1,4}$, etc. To satisfy the parity property, the XOR sum of all erasure information symbols along any redundancy direction needs to match the corresponding syndrome. For example, $\tilde{s}_{0,0} = a_{0,0} \oplus a_{0,1} \oplus a_{0,3}$ and $\tilde{s}_{0,1} = a_{0,0} \oplus a_{4,1} \oplus a_{2,3}$, etc.

In general, this step can be summarized as: 1) adjusters recovery ($j = 0,\ 1,\ 2$),

$$S_j = \bigoplus_{i=0}^{p-2} a_{i,p+j},$$

$S_1 = S_0 \oplus S_1$ and $S_2 = S_0 \oplus S_2$; 2) reversion of adjuster complement ($0 \leq i \leq p-2$),

$$a_{i,p+1} = a_{i,p+1} \oplus S_1,$$
$$a_{i,p+2} = a_{i,p+2} \oplus S_2;$$

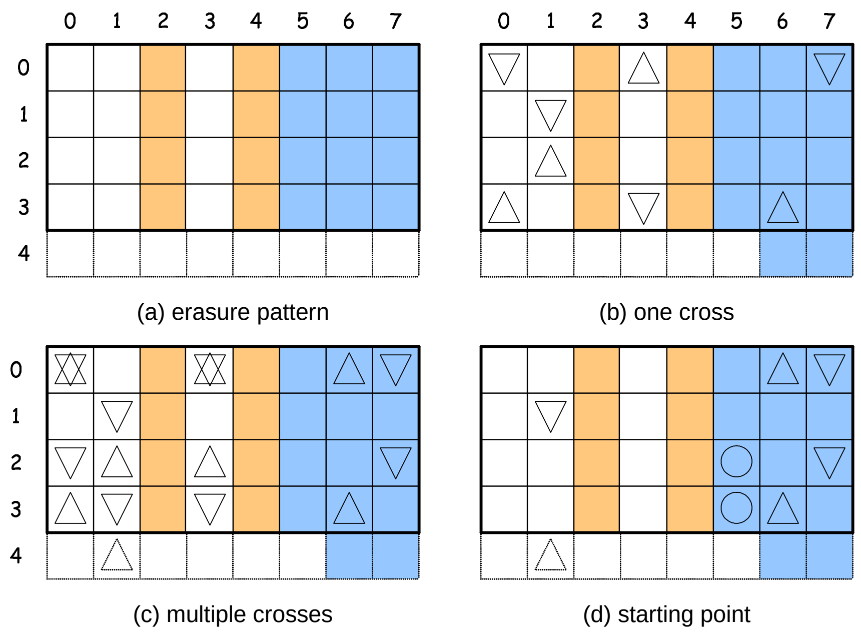


Figure 2.4: STAR Code Decoding

and 3) syndrome calculation,

$$\tilde{s}_{i,0} = a_{i,0} \oplus \left(\bigoplus_{j=0}^{p-1} a_{i,j} \right),$$

$$\tilde{s}_{i,1} = a_{i,1} \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle p+i-j \rangle_p, j} \right),$$

$$\tilde{s}_{i,2} = a_{i,2} \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle i+j \rangle_p, j} \right),$$

where $0 \leq i \leq p-1$ and $j \neq r$, s or t.

Find a Starting Point

Recall that finding a starting point is the key step of the EVENODD decoding, which seeks one particular diagonal with only one *unknown* symbol. This symbol can then be recovered from its corresponding syndrome, and it triggers the Zig-Zag decoding process until all unknown symbols are recovered. In the STAR decoding, however, it is *impossible* to find any parity direction (horizontal, diagonal or anti-diagonal) with only one unknown symbol. Therefore, the approach adopted in the EVENODD decoding does *not* directly apply here, and additional steps are needed to find a starting point.

For illustration purpose, we now assume all syndromes are represented by the shadowed symbols in the three parity columns, as shown in Figure 2.4(b). Based on the diagonal parity property, it is clear that $\tilde{s}_{3,1}$ equals to the XOR sum of three unknown symbols $a_{3,0}$, $a_{2,1}$ and $a_{0,3}$, as marked by "$\triangle$" signs in Figure 2.4(b). Similarly, $\tilde{s}_{0,2} = a_{0,0} \oplus a_{1,1} \oplus a_{3,3}$, which are all marked by "$\triangledown$" signs along an anti-diagonal. Imagine that all these marked symbols in the erasure information columns altogether form a *cross* pattern, whose XOR sum is computable ($\tilde{s}_{3,1} \oplus \tilde{s}_{0,2}$ in this case). And the *key* of this step is to choose multiple crosses, such that the following two conditions are satisfied: 1) each cross is v symbols offset from a previous one; and 2) the bottom row (after wrapping around) of the last cross *steps over* the top row of the first cross. In our particular example, two crosses are chosen. The second cross is $v = 2$ symbols offset from the first one and consists of erasure symbols $a_{0,0}$, $a_{4,1}$, $a_{2,3}$ (marked by "$\triangle$") and $a_{2,0}$, $a_{3,1}$, $a_{0,3}$ (marked by "$\triangledown$"), as shown in Figure 2.4(c). It is straightforward that the XOR sum of these two crosses equals to $\tilde{s}_{3,1} \oplus \tilde{s}_{0,2} \oplus \tilde{s}_{0,1} \oplus \tilde{s}_{2,2}$.

Notice, on the other hand, the calculation (XOR sum) of these two crosses includes symbols $a_{0,0}$ and $a_{0,3}$ twice. Their values are thus canceled out and do *not* affect the result. Also notice that the parities of unknown symbol sets ($a_{2,0}$, $a_{2,1}$ and $a_{2,3}$) and ($a_{3,0}$, $a_{3,1}$ and $a_{3,3}$) can be determined by horizontal syndromes $\tilde{s}_{2,0}$ and $\tilde{s}_{3,0}$, respectively. Thus, we can get

$$a_{1,1} \oplus a_{4,1} = \tilde{s}_{3,1} \oplus \tilde{s}_{0,2} \oplus \tilde{s}_{0,1} \oplus \tilde{s}_{2,2} \oplus \tilde{s}_{2,0} \oplus \tilde{s}_{3,0},$$

as all marked in Figure 2.4(d).

Repeating this process and starting the first cross at different rows, it is clear that we can obtain the XOR sum of any unknown symbol pair with a fixed distance 3 in column 1, i.e. $a_{0,1} \oplus a_{3,1}$, $a_{2,1} \oplus a_{0,1}$, etc.

From this example, we can see that the first condition of choosing crosses ensures the alignment of unknown symbols in the middle erasure column with those in the side erasure columns. Essentially, it groups unknown symbols together and replaces them with known syndromes. This is one way to cancel unknown symbols and results in a chain of crosses. The other way to cancel unknown symbols comes from the second condition, where unknown symbols in the header row of the cross chain are canceled with those in the tail row. This is indeed "gluing" the header of the first cross with the tail of the last one and turns the chain into a *ring*. It is conceivable that the number of crosses in the ring is completely determined by the erasure pattern (r, s and t) and the STAR code parameter p. And the following Lemma 1 shows the existence of such chain given arbitrary $u = s - r$, $v = t - s$ and p.

Lemma 1 *A ring satisfying both above conditions always exists and consists of l_d ($0 \leq l_d < p$) crosses, and l_d is determined by the following equation:*

$$\langle u + l_d v \rangle_p = 0, \tag{2.1}$$

where $0 \leq u,\ v < p$.

Proof. Since p is a prime number, integers modulo p define a finite field $GF(p)$. Let v^{-1} be the unique inverse of v in this field. Then, $l_d = (p-u)v^{-1}$ exits and is unique. ■

Given a ring, rows with 3 unknown symbols are substituted with horizontal syndromes (*substitution*), and symbols being included even times are simply removed (*simple cancellation*). For simplicity, we refer both cases as *cancellations*. Eventually, there are exactly two rows left with unknown symbols, which is confirmed by the following Lemma 2.

Lemma 2 *After cancellations, there are exactly two rows with unknown symbols in a ring. And the row numbers are* u *and* $p-u$*, as offsets from the top row of the first cross.*

Proof. To simplify the proof, we only examine the ring, whose first cross starts at row 0. Now the first cross contains two unknown symbols in column r and they are in rows 0 and $u+v$. We can represent them with a polynomial $(1+x^{u+v})$, where the power values (modulo p) of x correspond to row indices. Similarly, the unknown symbols in column s can be represented as $(x^u + x^v)$. Therefore, the first cross can be completely represented by $(1+x^{u+v}+x^u+x^v)$ and the ${l_1}^{th}$ cross by

$$(1+x^{u+v}+x^u+x^v)x^{l_1 v},$$

where $0 \leq l_1 < l_d$ and the coefficients of x are binary. Note that we don't explicitly consider unknown symbols in column t, which are reflected by polynomials representing column r. Using this representation, the cancellation of a polynomial term includes both cases of substitution and simple cancellation. And computing the XOR sum of all crosses can be equivalently represented by adding all corresponding polynomials together, as

$$\begin{aligned}
&\sum_{l_1=0}^{l_d-1}(1+x^{u+v}+x^u+x^v)x^{l_1 v} \\
=&(1+x^u)\sum_{l_1=0}^{l_d-1}(1+x^v)x^{l_1 v} \\
=&(1+x^u)(1+x^{p-u}) \\
=&x^u + x^{p-u},
\end{aligned} \tag{2.2}$$

where l_d is substituted using the result from Lemma 1. Thus, only two rows with unknown symbols are left after cancellations and the distance between them is $d = \langle p-2u \rangle_p$. ■

It is important to point out that unknown symbols in the remaining two rows are *not* necessarily in column s. For example, if $r=0$, $s=2$ and $t=3$, the remaining unknown symbols would be $a_{2,0}$, $a_{2,3}$, $a_{3,0}$ and $a_{3,3}$, which are indeed columns r and t. However, it is conceivable that we can easily get the XOR sum of corresponding unknown symbol pair in column s, since horizontal syndromes are available.

To summarize this step, we denote l_h to be the number of rows in a ring, which are canceled through substitution and define the set of corresponding row indices as $F_h = \{h_{l_2} \mid 0 \leq l_2 < l_h\}$. The set F_h is simply obtained by enumerating all crosses of the ring and then counting rows with 3 unknown symbols. Let $\tilde{a}_u$ denote the XOR sum of the unknown symbol pair $a_{0,s}$ and $a_{\langle p-2u\rangle_p,s}$, then the i^{th} pair has

$$\tilde{a}_{u+i} = \bigoplus_{l_1=0}^{l_d-1} \tilde{s}_{\langle -r+i\rangle_p,2} \bigoplus_{l_2=0}^{l_h-1} \tilde{s}_{\langle h_{l_2}+i\rangle_p,0}, \bigoplus_{l_1=0}^{l_d-1} \tilde{s}_{\langle t+i\rangle_p,1} \tag{2.3}$$

where $0 \leq i \leq p-1$.

Recover Middle Erasure Column

In the previous step, we have computed the XOR sum of arbitrary unknown symbol pair in column s with the fixed distance 3. Since symbol $a_{4,1}$ is an imaginary symbol with zero value, it is straightforward to recover symbol $a_{1,1}$. Next, symbol $a_{3,1}$ can be recovered since the XOR sum of the pair $a_{1,1}$ and $a_{3,1}$ is available. Consequently, symbols $a_{0,1}$ and $a_{2,1}$ are recovered. This process is shown to succeed with arbitrary parameters by the following Lemma 3.

Lemma 3 *Given the XOR sum of arbitrary symbol pair with a fixed distance d, all symbols in the column are recoverable if there is at least one symbol available.*

Proof. Since p is prime, $F = \{\langle di\rangle_p \mid 0 \leq i \leq p-1\}$ covers all integers in $[0, p)$. Therefore, a “tour” starting from row $p-1$ with the stride size d will visit all other rows exactly once before returning to it. As the symbol in row $p-1$ is always available (zero indeed) and the XOR sum of any pair with distance d is also known, all symbols can then be recovered along the tour. ■

To summarize, this step computes

$$\tilde{a}_{\langle (p-1)-di\rangle_p} = \tilde{a}_{\langle (p-1)-di\rangle_p} \oplus a_{\langle (p-1)-d(i-1)\rangle_p}, \tag{2.4}$$

where $0 \leq i \leq p-1$. Then, $a_{i,s} = \tilde{a}_i$ (where there are 2 unknown symbols left in the ring after cancellations) or $a_{i,s} = \tilde{a}_i \oplus \tilde{s}_{i,0}$ (where 4 unknown symbols are left) for all i’s. Thus far, column s is completely recovered.

Recover Side Erasure Columns

Now that column s is known, the first $p+2$ columns compose an EVENODD coded block with 2 erasures. It is conceivable that direct application of the EVENODD decoding can easily recover all remaining unknown symbols. Details are skipped in here.

2.3.2 Decoding without Parity Erasures: Symmetric Case

When the erasure pattern is symmetric ($u = v$), the decoding becomes much easier, where step 2 is greatly simplified while all other steps remain the same.

To illustrate the step of finding a starting point, we still resort to the previous example, although the erasure pattern is different now. Let's assume $r = 0$, $s = 1$ and $t = 2$. It is easy to see that only one cross is needed to construct a "ring" (still denoted as a ring, although not closed anymore). As in this example, a cross consists of unknown symbols $a_{0,0}$, $a_{0,2}$, $a_{2,0}$ and $a_{2,2}$, and $a_{1,1}$ is canceled because it is included twice. The XOR sum of the cross thus equals to $\tilde{s}_{2,1} \oplus \tilde{s}_{0,2}$. This is very similar to the situation in the previous case, where there are 4 unknown symbols in a ring after cancellations. Therefore, the rest of the decoding can followed the already described procedure and we don't repeat in here.

In summary the symmetric case can be decoded using the procedure for the asymmetric case, by simply setting $l_d = 1$, $l_h = 0$, $u = 0$ and $d = t - r$.

2.3.3 Decoding with Parity Erasures

In this part, we consider the situation when there are erasures in parity columns. The decoding is divided into the following 3 subcases.

Column $p + 2$ is an erasure.

In this subcase, parity column $p + 2$ is an erasure. Then, the rest $p + 2$ columns can be regarded as an EVENODD coded block with 2 or less erasures. Direct application of the EVENODD decoding can recover all unknown information symbols. Note that this case also takes care of all situations when erasures are less than 3.

Column $p + 1$ is an erasure, while $p + 2$ is not.

This subcase is almost the same as the previous case, except that now the "EVENODD" coded block consists of the first $p + 1$ columns and column $p + 2$. In fact, this coded block is no longer a normal EVENODD code, but rather a mirror reflection of one over the horizontal axis. Nevertheless, it can be decoded with slightly modification of the EVENODD decoding, which we simply leave to interested readers.

Column p is an erasure, while $p + 1$ and $p + 2$ are not.

Besides the above two, the only remaining subcase yet with parity erasures satisfy $0 \leq r < s \leq p-1$ and $t = p$, whose decoding is slightly different.

First, it is not possible to recover adjusters S_1 and S_2, as symbols in column p are unknown. However, $S_1 \oplus S_2$ is still computable, which simply equals to the XOR sum of all symbols in column

$p+1$ and $p+2$. This is easy to see by substituting the definitions of S_1 and S_2, where S_0 are added twice and canceled out. Then, it is not possible to reverse the adjuster complement. And the results from syndrome calculation are XOR sums of syndromes and their corresponding adjusters, rather than syndromes themselves. We use $\hat{s}_{i,j}$ to denote the results, which thus satisfy

$$\hat{s}_{i,j} = \tilde{s}_{i,j} \oplus S_j, \tag{2.5}$$

where $j = 1$ or 2 and $0 \leq i \leq p-1$. Note that $\hat{s}_{i,0} = \tilde{s}_{i,0}$ for all i's.

The next step is similar to the decoding of the symmetric case without parity erasures, as it is also true that only one cross is needed to construct a ring. Taking the cross starting with row 0 as an example, it consists of unknown symbols $a_{0,r}$. $a_{0,s}$, $a_{u,r}$ and $a_{u,s}$. Since the XOR sum of this cross equals to $\tilde{s}_{s,1} \oplus \tilde{s}_{\langle -r \rangle_p,2}$, we can easily get the following equation by substituting Eq. 2.5:

$$a_{0,r} \oplus a_{0,s} \oplus a_{u,r} \oplus a_{u,s} = \hat{s}_{s,1} \oplus \hat{s}_{\langle -r \rangle_p,2} \oplus S_1 \oplus S_2.$$

Therefore, the XOR sum of the cross is computable. Following the approach as used to recover middle erasure column in an early section, the XOR sum of two unknown symbols on any row can be recovered, which is still denoted as $\tilde{a}_i$ ($0 \leq i \leq p-1$). Then, parity column p can be recovered, as

$$a_{i,p} = \tilde{a}_i \oplus \tilde{s}_{i,0} = \tilde{a}_i \oplus \hat{s}_{i,0},$$

where $0 \leq i \leq p-1$.

After column p is recovered, the first $p+2$ columns can again be regarded as an EVENODD coded block with 2 erasures at column r and s. Therefore, the application of the EVENODD decoding can complete the recovery of all the remaining unknown symbols.

To summarize the procedure in this subcase, we have

$$S_1 \oplus S_2 = \left(\bigoplus_{i=0}^{p-2} a_{i,p+1}\right) \oplus \left(\bigoplus_{i=0}^{p-2} a_{i,p+2}\right),$$

and

$$\hat{s}_{i,0} = a_{i,0} \oplus \left(\bigoplus_{j=0}^{p-1} a_{i,j}\right),$$

$$\hat{s}_{i,1} = a_{i,1} \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle p+i-j \rangle_p,j}\right),$$

$$\hat{s}_{i,2} = a_{i,2} \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle i+j \rangle_p,j}\right),$$

where $0 \leq i \leq p-1$ and $j \neq r$ or s. Then,

$$\tilde{a}_i = \hat{s}_{\langle s+i\rangle_p,1} \oplus \hat{s}_{\langle -r+i\rangle_p,2} \oplus S_1 \oplus S_2,$$

where $0 \leq i \leq p-1$, and

$$\tilde{a}_{\langle (p-1)-ui\rangle_p} = \tilde{a}_{\langle (p-1)-ui\rangle_p} \oplus a_{\langle (p-1)-u(i-1)\rangle_p},$$

where $1 \leq i \leq p-1$. Finally, column p can be recovered as

$$a_{i,p} = \tilde{a}_i \oplus \hat{s}_{i,0},$$

for all i's. The rest is to use the EVENODD decoding to recover the remaining 2 columns, which is skipped in here.

Putting all the above cases together, we conclude this section with the following theorem:

Theorem 1 *The STAR code is completely recoverable from any triple column erasures.*

2.4 Algebraic Representation of the STAR Code

As described in [15], each column of an EVENODD coded block can be regarded algebraically as an element of a polynomial ring, which is defined with multiplication taken modulo $M_p(x) = (x^p - 1)/(x-1) = x^{p-1} + x^{p-2} + \cdots + x + 1$. For the ring element x, it is shown that its multiplicative order p. Using β to denote this element, then column j $(0 \leq j \leq p+1)$ in the coded block can be represented using the notation $a_j(\beta) = a_{p-2,j}\beta^{p-2} + \cdots + a_{1,j}\beta + a_{0,j}$, where $a_{i,j}$ $(0 \leq i \leq p-2)$ is the i^{th} symbol in the column. Note that the multiplicative inverse of β exists and can be denoted as β^{-1}. Applying same notations to the STAR code, we can then get its parity check matrix as:

$$H = \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 & 0 & 0 \\ 1 & \beta & \cdots & \beta^{p-1} & 0 & 1 & 0 \\ 1 & \beta^{-1} & \cdots & \beta^{-(p-1)} & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

It is straightforward to verify that any 3 columns in the check matrix are linearly independent. Therefore, the minimum distance of the STAR code is 4 (each column is regarded as a single element in the ring) and thus arbitrary triple (column) erasures are recoverable. This is an alternative way to show its MDS property.

2.5 Single Error Correction

The minimum distance 4 also implies that the STAR code can correct 1 column error and recover 1 column erasure simultaneously. Here, we again consider the most general case, where both the erasure and the error are information columns, denoted by j_d and j_e, respectively. Other patterns of erasure and error can be handled similarly and we leave them to interested readers.

Given a STAR coded block with erasure and error columns, we first calculate syndromes, as in the previous section.

$$\tilde{s}_{i,0} = a_{i,0} \oplus \left(\bigoplus_{j=0}^{p-1} a_{i,j} \right)$$

$$\tilde{s}_{i,1} = a_{i,1} \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle p+i-j \rangle_p, j} \right)$$

$$\tilde{s}_{i,2} = a_{i,2} \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle i+j \rangle_p, j} \right),$$

where $0 \le i \le p-1$ and $j \neq j_d$. Note that j_e is unknown before decoding and thus column j_e is included in the above calculations implicitly. Using algebraic notations, syndromes along same parity directions can be represented using a polynomial $\tilde{S}_j(\beta) = \sum_{i=0}^{p-1} \tilde{s}_{i,j}\beta^i$, where $j = 0$, 1 or 2. Let $a_{j_d}(\beta)$ and $e_{j_e}(\beta)$ denote polynomials corresponding to the original data of column j_d and the error data of column j_e, respectively. From the property of the check matrix, we have

$$\begin{aligned} \tilde{S}_0(\beta) + a_{j_d}(\beta) + e_{j_e}(\beta) &= 0 \\ \tilde{S}_1(\beta) + \beta^{j_d} a_{j_d}(\beta) + \beta^{j_e} e_{j_e}(\beta) &= 0 \\ \tilde{S}_2(\beta) + \beta^{-j_d} a_{j_d}(\beta) + \beta^{-j_e} e_{j_e}(\beta) &= 0. \end{aligned}$$

After simple cancellations, the above equations become

$$\begin{aligned} (\beta^{j_d} + \beta^{j_e}) e_{j_e}(\beta) &= \beta^{j_d}\tilde{S}_0(\beta) + \tilde{S}_1(\beta) \\ (\beta^{-j_d} + \beta^{-j_e}) e_{j_e}(\beta) &= \beta^{-j_d}\tilde{S}_0(\beta) + \tilde{S}_2(\beta) \end{aligned}$$

which can be further deduced to get

$$\beta^{j_d}\tilde{S}_0(\beta) + \tilde{S}_1(\beta) = \beta^{(j_d+j_e)}\left(\beta^{-j_d}\tilde{S}_0(\beta) + \tilde{S}_2(\beta)\right) \tag{2.7}$$

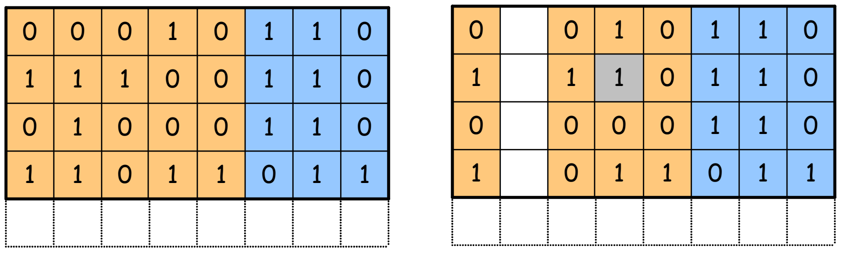

(a) original codeword block
(b) corrupted codeword block (f_d=1, f_e=3)

Figure 2.5: Single Error and Single Erasure Decoding

Note that the multiplication operation in the polynomial ring can be implemented efficiently using only shift and XOR operations, as presented in [23]. Here, we recap briefly with a simple example. Assume $p = 5$ and a column is $a = [0\ 1\ 1\ 1]^T$ with each entry representing a symbol. It is clear that this column can be represented algebraically as $a(\beta) = \beta + \beta^2 + \beta^3$. Straightforward calculation shows $\beta^2 a(\beta) = \beta^3 + \beta^4 + \beta^5 = \beta + \beta^2$, since $\beta^4 = 1 + \beta + \beta^2 + \beta^3$ and $\beta^5 = 1$. Or we can use the alternative approach as described in [23], which is more efficient when each symbol contains more than just one bit information. It first shifts all symbols in the column by the power of β, which is 2 in this example, and we now have $a = [1\ 0\ 0\ 1\ 1]^T$. Note that symbol $p - 1$ is not part of the original column and has an imaginary value of zero. But it participates in the shift operation, so the column has p entries now. The next step is to remove symbol $p - 1$ by *binary complement* (add symbol $p - 1$ to all other symbols). Then, we get $a = [0\ 1\ 1\ 0]^T$, which corresponds to the polynomial $\beta + \beta^2$ and matches the result from regular polynomial calculations.

With this technique, it is simple to compute terms $\beta^{j_d}\tilde{S}_0(\beta) + \tilde{S}_1(\beta)$ and $\beta^{-j_d}\tilde{S}_0(\beta) + \tilde{S}_2(\beta)$ in Eq. (2.7). Then the error correction of the STAR code boils down to finding j_e such that Eq. (2.7) is satisfied. Once j_e is known, the error column can be treated as another erasure and the invocation of the erasure decoding procedure can recover both column j_d and j_e.

This process is illustrated by an example. A corrupted STAR codeword block is shown in Figure 2.5(b), where there are single erasure (column $j_d = 1$) and single error (the shadowed symbol in column $j_e = 3$). We calculate syndromes and get the column representations of $\tilde{S}_0(\beta)$, $\tilde{S}_1(\beta)$ and $\tilde{S}_2(\beta)$ as $[0\ 0\ 1\ 1\ 0]^T$, $[1\ 1\ 0\ 0\ 1]^T$ and $[1\ 1\ 1\ 1\ 0]^T$, respectively. Then, $\beta\tilde{S}_0(\beta) + \tilde{S}_1(\beta)$ and $\beta^{-1}\tilde{S}_0(\beta) + \tilde{S}_2(\beta)$ can be calculated using aforementioned approach as $[1\ 1\ 0\ 1\ 0]^T$ and $[1\ 0\ 0\ 1\ 0]^T$. By *trial and error* method, it is easy to see that the first term becomes $[0\ 1\ 1\ 0\ 1]^T$ after shifting down one symbol and then $[1\ 0\ 0\ 1\ 0]^T$ after the binary complement, which now equals to the second term. Therefore,

$$\beta\big(\beta\tilde{S}_0(\beta) + \tilde{S}_1(\beta)\big) = \beta^{-1}\tilde{S}_0(\beta) + \tilde{S}_2(\beta)$$

Based on Eq. (2.7), we then have $j_d + j_e = 4$ and in turn $j_e = 3$. The next step is to treat both column 1 and 3 as erasures and invoke the erasure decoding to recover all symbols in these two columns.

2.6 Complexity Analysis

In this section, we analyze the complexity of the STAR code erasure decoding. The complexity is dominated by XOR operations, thus we can count the total number of XORs and use that as an indication of the complexity. Since decoding without parity erasures is the most complicated case, including both asymmetric and symmetric erasure patterns, we confine our analysis to this case.

2.6.1 Erasure Decoding Complexity

It is not difficult to see that the complexity can be analyzed individually for each of the 4 decoding steps. Note that a complete STAR code consists of p information columns and 3 parity columns. When there are only k ($k \leq p$) information columns, we can still use the same code by resorting to the *shortening* technique, which simply assigns zero value to all symbols in the last $p-k$ information columns. Therefore, in the analysis here, we assume the code block is a $(p-1) \times (k+3)$ array.

In step 1, the calculation of S_0 takes $(p-2)$ XOR operations and those of S_1 and S_2 take $(p-1)$ XORs each. The reversion of adjuster complement takes $2(p-1)$ XORs in total. Directly counting XORs of the syndrome calculations is fairly complicated and we can resort to the following alternative approach. First, it is easy to see that the syndrome calculations of any parity direction for a code block without erasures (a $(p-1) \times (p+3)$ array) take $(p-1)p$ XORs. Then, notice that any information column contributes $(p-1)$ XORs to the calculations. Therefore, for a code block with $(k-3)$ information columns (with triple erasures), the number of XORs becomes $(p-1)p - (p-k+3)(p-1) = (k-3)(p-1)$. In total, the XORs in this step is:

$$(p-2) + 2(p-1) + 2(p-1) + 3(k-3)(p-1) = (3k-4)(p-1) - 1. \tag{2.8}$$

In step 2, the computation of each ring takes $(2l_d + l_h - 1)$ XORs and there are $(p-1)$ rings to compute. Thus, the number of XORs is

$$(2l_d + l_h - 1)(p-1). \tag{2.9}$$

In step 3, it is easy to see that the number of XORs is

$$(p-1) - 1 = p - 2. \tag{2.10}$$

In step 4, the horizontal and the diagonal syndromes need to be updated with the recovered symbols of column s, which takes $2(p-1)$ XORs. Note that there is no need to update the anti-diagonal syndromes, because the decoding hereafter deals with only double erasures. The Zig-Zag decoding then takes $2(p-1)-1$ XORs. So the number of XORs in this step is

$$2(p-1) + 2(p-1) - 1 = 4(p-1) - 1. \tag{2.11}$$

Note that in step 2, the number of XORs is computed assuming the case where only 2 unknown symbols are left in a ring after cancellations. If the other case happens, where 4 unknown symbols are left, additional $(p-1)$ XOR operations are needed to recover column s. However, this case does *not* need to update the horizontal syndromes in step 4 and thus saves $(p-1)$ XORs in there. Therefore, it is just a matter of moving XOR operations from step 2 to step 4 and the total number remains the same for both cases.

In summary, the total number of XORs required to decode triple information column erasures can be obtained by putting Eq. (2.8), (2.9), (2.10) and (2.11) together, as:

$$\begin{aligned} & (3k-4)(p-1) - 1 + (2l_d + l_h - 1)(p-1) \\ & + (p-2) + 4(p-1) - 1 \\ = \; & (3k + 2l_d + l_h)(p-1) - 3 \qquad (2.12) \\ \approx \; & (3k + 2l_d + l_h)(p-1). \qquad (2.13) \end{aligned}$$

2.6.2 A Decoding Optimization

From Eq. (2.13), we can see that for fixed code parameters k and p, the decoding complexity depends on l_d and l_h, which are completely determined by actual erasure patterns (r, s and t). In Sec. 2.3, we present an algorithm to construct a ring of crosses, which will yield a starting point for successful decoding. Within the ring, all crosses are $v = t - s$ symbols offset from previous ones. And from Eq. (2.2), there are exactly two rows with unknown symbols left after cancellations. From the symmetric property of the ring construction, it is not difficult to show that using offset $u = s - r$ will also achieve the same purpose. And if using u as offset results in smaller l_d and l_h values (to be specific, smaller $2l_d + l_h$), then there is advantage to do so.

Moreover, we make the assumption $r < s < t$ during the description of the decoding algorithm. Although it helps to visualize the key procedure of finding a starting point, this assumption is unnecessary. Indeed, it is easy to verify that all proofs in Sec. 2.3 still hold without this assumption. And by swapping values among r, s and t, it might be possible to reduce the decoding complexity. For instance, in the previous example, $r = 0$, $s = 1$ and $t = 3$ results in $l_d = 2$ and $l_h = 2$. If letting $r = 1$, $s = 0$ and $t = 3$, then $u = -1$ and $v = 3$. The pattern of single cross is shown in Figure 2.6(a). And from Figure 2.6(b), it is clear that two crosses close a ring, which contains exactly two rows (row 1 and 4) with unknown symbols after cancellations. Thus, this choice also

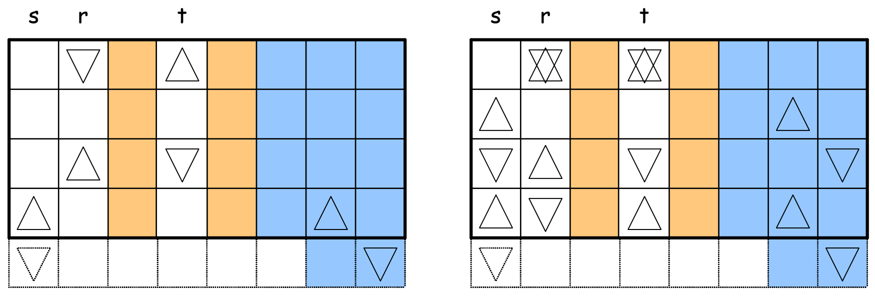


Figure 2.6: Optimization of STAR Decoding

yields $l_d = 2$ and $l_h = 2$. However, if letting $r = 0$, $s = 3$ and $t = 1$, we can get $u = s - r = 3$ and $v = t - s = -2$. It is easy to find out that unknown symbols in column s are canceled in every single cross. In fact, this is an equivalence of the symmetric case and in turn $l_d = 1$ and $l_h = 0$. Thus, the complexity is reduced by this choice. Note that for general u and v, the condition of symmetric now becomes $\langle u - v \rangle_p = 0$, instead of simply $u = v$.

Now let us revisit the ring construction algorithm described in Sec. 2.3. The key point there is to select multiple crosses such that the bottom row of the last cross "steps over" the top row of the first one, and there are exact two rows left with unknown symbols after cancellations. Further examination, however, reveals that it is possible to construct rings using alternative approaches. For instance, the crosses can be selected in such a way that *in the middle column* the bottom symbol of the last cross "steps over" the top symbol of the first one. Or perhaps there is even no need to construct closed rings and crosses might not have to be a fixed offset from previous ones. Indeed, if crosses can be selected arbitrarily while still ensures exact two rows left with unknown symbols after cancellations, the successful decoding can be guaranteed. Recall that single cross is represented by $C(x) = 1 + x^u + x^v + x^{u+v}$ and a cross with an offset of f symbols by $C(x)x^f$. Therefore, the construction of a ring is to determine a polynomial term $R(x)$, such that $C(x)R(x)$ results in exact two entries. For instance, the example in Sec. 2.3 has $R(x) = 1 + x^2$ and $C(x)R(x) = x + x^4$. Moreover, the following Theorem 2 shows that the decoding complexity is minimized if a $R(x)$ with minimum entries is adopted.

Theorem 2 *The decoding complexity is nondecreasing in terms of the number of crosses (l_d) in a ring.*

Proof. Whenever a new cross is included into the ring, two new non-horizontal syndromes (one diagonal and one anti-diagonal) need to be added to the XOR sum. With this new cross, at most four rows can be canceled (simple cancellation due to even times addition), among which two can be mapped with this cross and the other two with an earlier cross. Thus, each cross brings in two non-horizontal syndromes and takes away at most two horizontal syndromes. The complexity is nondecreasing in terms of the number of crosses. ■

Note that l_d is in fact the number of entries in $R(x)$. Now optimized ring constructions desire to find a $R(x)$ with minimum entries, which ensures that $C(x)R(x)$ has only two terms. An efficient approach to achieve this is to test all polynomials with two terms. If a polynomial is divisible by $C(x)$, then the quotient yields a valid $R(x)$. A $R(x)$ with minimum entries is then chosen to construct the ring. It is important to point out that there is no need to worry about common factors (always powers of x) between two terms in the polynomial, as it is not divisible by $C(x)$. Thus, the first entry of all polynomials can be fixed as 1, which means that only $p-1$ polynomials ($1+x^i$, $0 < i \leq p-1$) need to be examined. As stated in an earlier section, polynomials are essentially elements in the ring constructed with $M_p(x) = 1 + x + \cdots + x^{p-2} + x^{p-1}$. Based on the argument in [23], $(1+x^u)$ and $(1+x^v)$ are invertible in the ring. Thus, $C(x) = (1+x^u)(1+x^v)$ is also invertible, and it is straightforward to compute the inverse using Euclid's algorithm. For instance, $C(x) = 1 + x + x^2 + x^3$, as $u = 1$ and $v = 2$ in the previous example. The generator polynomial $M_p(x) = 1 + x + x^2 + x^3 + x^4$ as $p = 5$. Applying Euclid's algorithm, it is clear that

$$1(\mathbf{1+x+x^2+x^3+x^4}) + x(\mathbf{1+x+x^2+x^3}) = 1. \tag{2.14}$$

Thus, the inverse of $C(x)$ is $inv(C(x)) = x$. When examining the polynomial $1 + x^3$, we get $R(x) = inv(C(x))(1+x^3) = x + x^4$ or equivalently,

$$(1+x+x^2+x^3)(x+x^4) = 1 + x^3 \text{ m}od\ M_p(x). \tag{2.15}$$

It is desirable that $R(x)$ carries the entry of power 0, since the ring always contains the original cross. So we multiply x to both sides of Eq. (2.15), which now becomes

$$(1+x+x^2+x^3)(1+x^2) = x + x^4 \text{ m}od\ M_p(x).$$

Thus, we have $R(x) = 1 + x^2$ and the ring can be constructed using two crosses ($l_d = 2$) with an offset of two symbols. Once the ring is constructed, it is straightforward to get l_h.

It might seem contradictory to introduce ring operations to find the optimal l_d value and ring construction, as the whole purpose of the STAR code is to avoid computationally complex operations. However, it is important to point out that these operations (such as inversion using Euclid's algorithm) can be performed easily and do *not* require the construction of the complete ring. Furthermore, the optimization can be performed in advance (offline) and only XOR operations are required during (online) decoding procedures. This is elaborated in a later section.

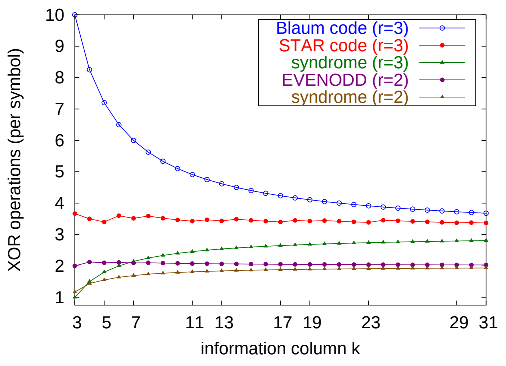


Figure 2.7: The Complexity Comparisons

2.7 Comparison with Existing Schemes

In this section, we compare the erasure decoding complexity of the STAR code to two other XOR-based codes, one proposed by Blaum *et al.* [18] (Blaum code hereafter) and the other by Blomer *et al.* [24].

The Blaum code is a generalization of the EVENODD code, whose horizontal and diagonal parities are now regarded as redundancies of slope 0 and 1, respectively. And the r^{th} parity column is generated using a redundancy of slope $r-1$. This construction is shown to maintain the MDS property for triple parity columns, given the code parameter p is a prime number. And the MDS property continues to hold for selected p values when the number of parities exceeds 3. To make the comparison meaningful, we focus on the triple parity case of the Blaum code. We compare the complexity of triple erasure decoding in terms of XOR operations between the Blaum code and the STAR code. Similar to all previous sections, we confine all three erasures to information columns.

The erasure decoding of the Blaum code adopts an algorithm described in [23], which provides a general technique to solve a set of linear equations in a polynomial ring. Due to special properties of the code, however, ring operations are *not* required during the decoding procedure, which can be performed with pure XOR and shift operations. The algorithm consists of 4 steps, whose complexities are summarized as follows: 1) syndrome calculation: $3(k-3)(p-1)-1$; 2) computation of $\hat{Q}(x;z)$: $\frac{1}{2}r(3r-3)p$; 3) computation of the right-hand value: $r((r-1)p+(p-1))$; and 4) extracting the erasure values: $r(r-1)(2(p-1))$. Here, $r=3$ is the number of erasures. Therefore, the total number of XORs is

$$\begin{aligned} & 3(k-3)(p-1)-1+9p+(9p-3)+12(p-1) \\ = \; & (3k+21)(p-1)+14 \qquad (2.16) \\ \approx \; & (3k+21)(p-1). \qquad (2.17) \end{aligned}$$

Comparison results with the STAR code are shown in Figure 2.7, where we can see that the

Table 2.1: Complexity of the RS Code (per 32 bits)

# of parities	finite field based impl.	XOR based impl.
r = 2	8 XORs + 8 muls	16 XORs
r = 3	12 XORs + 12 muls	24 XORs

complexity of the STAR decoding remains fairly constant and is just slightly above 3. Note that this complexity depends on actual erasure locations, thus the results reported here are average values over all possible erasure patterns. The complexity of the Blaum code, however, is rather high for small k values, although it *does* approach 3 asymptotically. As these recovery schemes are most likely to be applied with limited k values in streaming media applications, it is critical to achieve efficiency in this region and the STAR code is thus probably more favorable than the Blaum code. Figure 2.7 also includes the complexity of the EVENODD decoding as a reference, which is roughly constant and slightly above 2 XORs per symbol, Moreover, the complexity of syndrome calculations is depicted seperately for both the double and the triple erasure recoveries. It is clear that this part dominates the decoding complexity asymptotically.

The XOR-based code proposed in [24] uses Cauchy matrices to construct a Reed-Solomon (RS) code. It replaces generator matrix entries, information and parity symbols with binary representations. Then, the encoding and decoding can be performed with primarily XOR operations. To achieve maximum efficiency, it requires message length to be multiples of 32 bits. In that way, basic XOR unit is 32 bits, or single word, and can be performed by single operation. To compare with this scheme fairly, we require the symbol size of the STAR code to be multiples of 32 bits too. Then, the XOR-based decoding algorithm in [24] involves krL^2 XOR operations and r^2 operations in a finite field $GF(2^L)$, where k and r are the numbers of information symbols and erasures, respectively. Assume the code is constructed in the $GF(2^8)$ $(L = 8)$ and there are triple erasures $r = 3$. Also, ignore those r^2 finite field operations (due to the inversion of a decoding coefficients matrix), which tend to be small as the number of erasures is limited. Then, the normalized decoding complexity (by the total information length of kL words) is summarized in Table 2.1. Compared to Figure 2.7, where the STAR code decoding complexity is slightly more than 3 XORs per symbol (multiples of 32 bits now), it is clear that the STAR code is more efficient than the XOR-based RS code. The complexity of normal RS code implementation [25] is also listed in Table 2.1, which uses finite field operations intensively. It is clear that this implementation has even higher complexity than the XOR based scheme.

2.8 Implementation and Performance

The implementation of the STAR code encoding is straightforward, which simply follows the procedure described in Sec. 2.2. Thus, in this part, our main focus is on the erasure decoding procedure. As stated in Sec. 2.6, the decoding complexity is solely determined by l_d and l_h, given the number

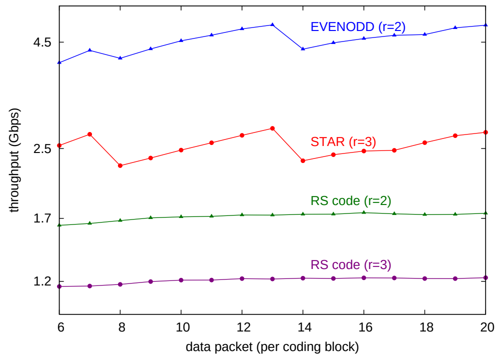


Figure 2.8: Throughput Performance

of information columns k and the code parameter p. As l_d and l_h vary according to actual erasure patterns, so does the decoding complexity. To achieve the maximum efficiency, we apply the optimization technique as described in the earlier section.

An erasure pattern is completely determined by the erasure columns r, s and t (again assume $r < s < t$), or further by the distances u and v between these columns, as the actual position of r does *not* affect l_d or l_h. Therefore, it is possible to setup a mapping from u and v to l_d and l_h. To be specific, given u and v, the mapping returns the positions of horizontal, diagonal and anti-diagonal syndromes, which would otherwise be obtained via ring constructions. The mapping can be implemented as a lookup table and the syndrome positions using bit vectors. Since the lookup table can be built in advance of actual decoding procedure, it essentially shifts complexity from online decoding to offline preprocess. Note that the table lookup operation is only needed once for every erasure pattern, thus there is no need to keep the table in memory (or cache). This is different from finite field based coding procedures, where intensive table lookups are used to replace complicated finite field operations. For example, RS code implementation might use an exponential and a logarithm table to perform multiplications and divisions. Furthermore, the number of entries in the lookup table is not large at all. For example, for code parameter $p = 31$, u and v are at most 30, which requires a table of at most $30 \times 30 = 900$ entries. The cost of keeping tables of this size is really negligible.

During the decoding procedure, u and v are calculated from the actual erasure pattern. Based on these values, the lookup table returns all syndrome positions, which essentially indicates the ring construction. The calculation of the ring is thus performed as the XOR sums of all the indicated syndromes. Then, the next ring is calculated by offsetting all syndromes with one symbol and the procedure continues until all rings are computed. And steps afterwords are to recover the middle column and then the sides columns, which are detailed in Sec. 2.3.

We implement the STAR code erasure decoding procedure and apply to streaming media applications. The throughput performance is measured and compared to a publicly available RS code implementation [26]. The results are shown in Figure 2.8, where the packet size is 528 bytes and the number of information packets in a code block varies from 6 to 20. These packet numbers are reasonable due to the delay constraints of the media data. And it is clear that the STAR code achieves about twice faster throughput than the RS code. Note that there are jigsaw effects in the throughputs of both the EVENODD and the STAR code. This happens mainly due to the shortening technique. When the number of data packets is not prime, the codes are constructed using the closest larger prime number. A larger prime number means each column (packet here) is divided into more pieces, which in turn incurs additional control overhead. As the number of information packets increases, the overhead is then amortized, reflected by the performance ramping up after each dip.

Chapter 3

Practical FEC Codes for Wireless Streaming

FEC techniques, based on error correcting codes, are widely used in streaming media applications to battle data loss and in turn to reduce or eliminate retransmission delays [27]. When used in multicast environments, it also helps to prevent negative feedbacks from overwhelming senders (*feedback implosion* [28]). In Chapter 2, we describe STAR, an efficient recovery scheme for triple erasures. It is shown that the STAR code requires pure XOR operations and also has the MDS property. With a recovery of up to triple packet losses, it is conceivable that the STAR code is ideal for streaming media applications with very strong delay constraints, where the number of data packets (in turn parity packets) is limited. In another word, this scheme is applied with limited coded block lengths.

In some streaming applications, however, the delay constraints are relatively weak, for instance, streaming of stored content can usually tolerate delay of several seconds. Larger delay implies that it might be possible to use larger coded block lengths to achieve better recovery performance. With a fixed coding rate (or redundancy), this would also mean increased number of parity packets in a single coded block. It is clear that the STAR code becomes insufficient, when the number of parity packets exceeds 3.

In this situation, it is certainly possible to resort to Reed-Solomon (RS) code [3]. The RS code can generate n total data packets from k information packets and tolerate up to $r = n - k$ arbitrary packet losses. It is clear that the RS code is a MDS code. And the parameters n and k of the code can be chosen very flexibly, which makes it applicable to this case. Indeed, the RS code is widely adopted in streaming media applications [28–35]. Moreover, as these applications increasingly include wireless links into the last mile of delivery, the RS code attracts even more attention due to high data loss characteristics of wireless networks [36–39]. While most attentions are focused on the optimal recovery performance of the RS code, there is one crucial aspect often neglected. The RS coding requires finite field operations, which are computationally complex. This could impose difficulties to constrained receivers, such as portable devices with limited computation capacity and power supply. In fact, our study in this chapter shows that the RS code can significantly increase the energy consumption of hand-held devices. In this sense, it might not always be a good choice as a FEC scheme.

On the other hand, there is a very efficient scheme relying on pure XOR operations: single parity code with *interleaving* (or simply, *PARITY code*). The PARITY code of length n and *interleaving degree* m can be described as an $m \times n$ array (with the last column dedicated to parity data). It is computationally efficient and energy saving, since all encoding and decoding operations are pure XORs. However, it does not in general have as good loss recovery performance as the RS code.

In this chapter, we seek to address the trade-off between loss recovery performance and computational cost for wireless data streaming. We study a practical FEC scheme using MDS array codes. Different from Chapter 2, we no longer map entire columns of an array to data packets, instead, each entry of the array is mapped to a single packet now. At this bit level, losses are not confined to limited columns and these array codes are no longer MDS. In this work, we focus on the suitability of the EVENODD code [15] for both random and bursty data loss recovery. An efficient and versatile decoding algorithm is proposed to enhance the EVENODD code. Compared to the PARITY code and the RS code, our analytical and simulation results show that the EVENODD code achieves good balance between loss recovery performance and savings in energy/computation. Our study suggests that the EVENODD code is suitable as a practical FEC scheme for wireless data streaming.

Note that there exist other XOR-based error correcting codes, which also have close to optimal data loss recovery capability, such as regular LDPC codes [40] fountain codes [41–43] (as used in the popular digital fountain approach [44]). But these codes usually require very large block lengths, which will violate delay constraints of streaming data applications. Therefore, it is not proper to consider them here.

3.1 EXtended EvenOdd Decoding (XEOD) Algorithm

3.1.1 EXtended EVENODD Decoding (XEOD) Scheme

The EVENODD code, its encoding and decoding are briefly described in Chapter 2. We denote the decoding algorithm there as the *Basic EVENODD Decoding* (BEOD). The BEOD is designed for an error (loss) model which is suitable for data storage devices, such as disks or tapes, where an entire column is considered to be an error or erasure as long as at least one of its symbols is an error or erasure. This *bursty* loss model is sensible for data storage applications. When the EVENODD code is used for a disk array, the BEOD can fully recover all the original data symbols when up to two disks fail. When the EVENODD code is applied to data streaming, however, it is unlikely that loss is constrained only in two columns, i.e., symbol loss can be *random* in addition to bursty. The

BEOD scheme doesn't provide a mechanism to deal with this situation. Moreover, when symbol loss occurs in one column, it doesn't imply that all symbols in the same column are lost. It is thus not computationally efficient to compute all the horizontal and diagonal syndromes if some of them are *not* actually used in decoding. However, there is no simple way to decide which syndrome is needed in the BEOD scheme.

To address all these issues, we propose an eXtended EVENODD Decoding (*XEOD*) algorithm for both random and bursty symbol losses. The EVENODD has been shown to be an LDPC (*Low Density Parity Check*) code and its probabilistic error correction performance based on its parity check matrix has been studied [40]. While taking advantage of the LDPC property of the EVENODD code as well, the XEOD algorithm is a *deterministic* one that corrects erasures (symbol losses) instead of errors (symbol corruptions) and thus is much more computationally efficient than other probabilistic decoding algorithms based on parity check matrix.

Now we described the XEOD: each codeword block is represented by a bipartite graph, with left nodes corresponding to the message symbols (in the first p columns) and right nodes corresponding to the check symbols (in the p^{th} and $(p+1)^{th}$ columns). A left node exists in the bipartite graph only if the corresponding message symbol is lost and a right node exists only if the check symbol is *not* lost. For simplicity, *message node* and *check node* are used to represent left node and right node throughout rest of this chapter. A link (edge) is setup between a message node and all its check nodes. The *degree* of a node represents the number of links connected to it. It is easy to see that the degree of a message node is no larger than 2 while the degree of a check node is less than or equal to $(p-1)$.

The XEOD starts decoding from a check node with degree 1 and moves to its only connected message node. Since this message node is the only missing one corresponding to the check node, it can be easily recovered and the link connected back to the check node is removed. If the message node has a second link, then the XEOD follows it to a new check and then removes the link. The same decoding cycle continues until reaching a message node with degree 0 or a check node with degree no equal to 1. Then the XEOD jumps to the next check node with degree 1 and repeats the entire procedure until no more such node exists.

Besides the above core loop procedure, the XEOD needs to compute the EVENODD adjuster, which is used to recover message symbols from the second check column symbols. The adjuster can be computed from all main diagonal message symbols, all check symbols or any second check column symbol with all its corresponding message symbols, as in [15]. This can be easily incorporated by augmenting the bipartite graph with an imaginary adjuster node, which connects to missing main diagonal message nodes. Experiments show that the adjuster can be calculated with very high probability when the symbol loss rate is relatively low. Thus, it is reasonable to assume the adjuster node always exists.

Algorithm 1 e**X**tended **E**VEN**O**DD **D**encoding (XEOD)

```
  procedure XEOD:
adjuster:
  if $adjuster\ node\ degree = 0$ then
    $calculate\ adjuster;\ goto$ decode;
  if $any\ check\ node\ degree = 0 \parallel all\ check\ nodes\ exist$ then
    $calculate\ adjuster;\ goto$ decode;
decode:
  for $all\ check\ nodes$ do
    while $check\ node\ degree = 1$ do
      if $adjuster\ node\ needed\ but\ not\ exist$ then
        $break$;
      $recover\ connected\ message\ node\ M$;
      $remove\ the\ link\ between\ M\ and\ the\ check\ node$;
      if $degree\ of\ M = 0$ then
        $break$;
      $check\ node \leftarrow the\ other\ check\ node\ connected\ to\ M$

  return $decoding\ complete$;
```

3.1.2 Correctness of XEOD

The complete decoding procedure is in Algorithm 1 and its correctness is stated in the following theorem:

Theorem 3 *There is no more message node recoverable when Algorithm 1 terminates.*

Proof. First, we prove that Algorithm 1 will terminate after finite number of iterations. This is because in each iteration, the decoder moves to a new check node and/or removes one link after recovering a message node. Since both check nodes and links in the graph are finite, the algorithm will terminate after finite iterations.

We then prove the theorem by contradiction. Suppose there exists a recoverable message node M when Algorithm 1 terminates. Then at least one check node (denoted by C) connected with M has degree 1. From the *while* loop in Algorithm 1, the only possible reason that the XEOD does *not* follow C's link to recover M is the adjuster node is needed but does *not* exist. This contradicts with the assumption that M is recoverable. ■

3.1.3 Complexity of XEOD

The complexity of the XEOD includes three parts: 1) constructing the bipartite graph; 2) computing the adjuster and 3) decoding itself. The bipartite graph construction takes $O(1)$ operation for each message node and $O(p^2)$ in total in the worst case when all message symbols are lost. Computing the adjuster has the worse case complexity of $O(p)$ when all check nodes are visited until the last

Table 3.1: BEOD computation analysis

# of XORs	CASE I $0 \leq i \leq p-1$, $j = p$	CASE II $0 \leq i \leq p-1$, $j = p+1$	CASE III $1 \leq i, j \leq p-1$	CASE IV $i = p$, $j = p+1$
calculate adjuster	p	0	$2p$	0
recover message	qp^2	qp^2	$2p^2 + p + 4pq$	0
# of occurrences	p	p	$\binom{p}{2}$	1

Table 3.2: XEOD computation analysis

# of XORs	CASE I $0 \leq i \leq p-1$, $j = p$		CASE II $0 \leq i \leq p-1$, $j = p+1$		CASE III $1 \leq i, j \leq p-1$		CASE IV $i = p$, $j = p+1$
	$i = 0$	$i \neq 0$	$i = 0$	$i \neq 0$	$i = 0$	$i \neq 0$	$i = p$
calculate adjuster	p	$2q(1-q)^{p-1} + [1 - q(1-q)^{p-1}]p$	p	$2q(1-q)^{p-1} + [1 - q(1-q)^{p-1}]p$	$(1-q)p + 2qp$	$(1-q)^2 p + 2[1 - (1-q)^2]p$	p
recover message	qp^2		qp^2		$2qp^2$		0
# of occurrences	1	$p-1$	1	$p-1$	$p-1$	$\binom{p-1}{2}$	1

one can be used for calculation. In the decoding procedure, each check node can at most be visited p times and involve XOR of p symbols, which yields $O(p^2)$ total complexity. Hence, the XEOD has the worst complexity $O(p^2)$, which is linear in terms of message symbol number. Note that it is easy to verify this complexity is reachable when all symbols in any two message columns are lost.

It is not difficult to see that the BEOD has complexity $O(p^2)$, which recovers all $2(p-1)$ symbols in any two columns with p XOR operations for each symbol. Thus, the BEOD has the same complexity as the worst case XEOD. However, if all symbols are not lost in those two columns (quite common in data streaming), the BEOD wastes computation by calculating unneeded horizontal and diagonal syndromes, as we mentioned earlier. In this part, we quantitatively analyze the computation advantage of the XEOD in avoiding these unneeded operations.

For comparison purpose, we confine symbol losses in two columns (i, $j \in$ [0, $p+1$]), such that both the BEOD and the XEOD can fully recover all losses. Assume a random loss rate q. Note that, for the XOR-based decoding schemes, computation analysis can be simplified by counting the number of XOR operations, which is the only dominating factor in the decoding process. To further simplify analysis, we don't distinguish $(p-1)$ and p, as p is large enough.

For the BEOD, we can categorize the analysis into four cases and the result is summarized in Table 3.1, where row 2 shows the number of XOR operations needed in calculating the EVENODD adjuster, row 3 the number of XOR operations in recovering message symbols and row 4 the number of occurrences in total $\binom{p+2}{2}$ cases. The expect value of the number of XOR operations can be calculated as:

$$E_{BEOD}(\#\ of\ XORs) = \frac{p^4 + (4q + \frac{1}{2})p^3 - (2q + \frac{1}{2})p^2}{\binom{p+2}{2}}$$

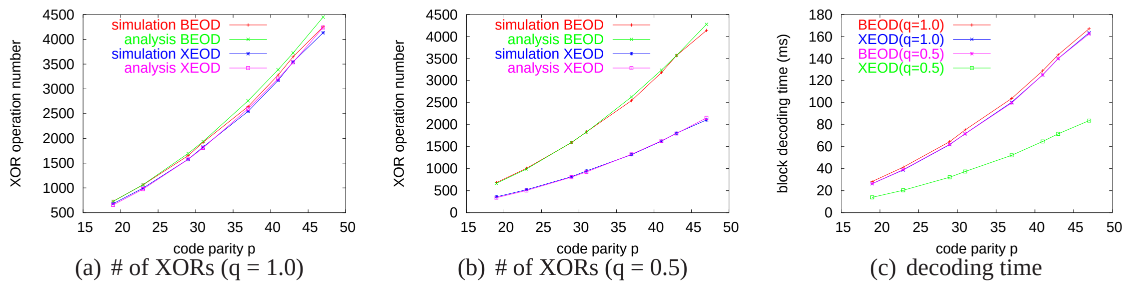


Figure 3.1: Computation Advantage of the XEOD

For the XEOD, the analysis is similar and summarized in Table 3.2, using the same notation as in Table 3.1. The expect value of the number of XOR operations can be calculated as:

$$E_{XEOD}(\#\, of\ XORs) = \frac{qp^4 + (\frac{1}{2} + 2q - \frac{1}{2}q^2)p^3 + (\frac{3}{2} - 2q + \frac{3}{2}q^2)p^2}{\binom{p+2}{2}} + \frac{(1-q-q^2)p + 2qp(1-q)^{p-1}(p-1)}{\binom{p+2}{2}}$$

The analysis results are also verified by simulation, where the number of XOR operations are counted in a real decoding implementation. In the simulation, each symbol in a codeword block corresponds to a data packet of 500 bytes, which is a proper choice for normal data streaming applications. The analysis and simulation results conform well with each other, as shown in Figure 3.1(a) and 3.1(b).

Figure 3.1(a) shows a special case where all the symbols in the column i and j are lost, which is the original case BEOD designed to deal with. The XEOD performs about the same number of XOR operations as the BEOD in this case. When only half of the symbols are lost, Figure 3.1(b) shows the saving of the XEOD in terms of XOR operations. This is understandable since the XEOD does *not* calculate unneeded horizontal and diagonal syndromes as the BEOD. The advantage of the XEOD is further verified by actual time measured in decoding simulation, as shown in Figure 3.1(c).

In summary, the XEOD is no worse than the BEOD in computation load under any circumstance and outperforms the BEOD in most cases. And the computation advantage of the XEOD becomes more prominent, as the symbol loss rate decreases. Moreover, the XEOD can handle more general loss scenarios and thus achieves higher loss recovery capability than the BEOD.

3.2 Energy Consumption and Throughput

3.2.1 Energy Consumption

Energy consumption is a big concern for wireless terminals. In this section, we study the feasibility of using various FEC schemes in wireless data streaming from this perspective. In particularly, we compare the decoding energy consumption of the proposed XEOD with the PARITY code and the RS code.

We choose a 30-sec MP3 file as experiment data and use the energy consumption of its decoding as a baseline. Both the MP3 file (*large.mp3*) and the MP3 decoder (*madplay*) are from a representative embedded benchmark suite (MiBench [45]). Since it is difficult to measure energy consumption directly, we resorts to Sim-Panalyzer [46] for simulation. We choose Intel StrongARM SA1100 (200 MHz) as a target microprocessor, which is used in many Compaq iPAQ Pocket PCs. Various decoder implementations are then compiled to generate StrongARM binary executables. The Sim-Panalyzer simulates the execution of them and collects instruction level power consumption data. Finally, the energy consumption is computed from total power consumption, microprocessor frequency and total clock cycles. For the RS code, we use Rizzo's implementation [26].

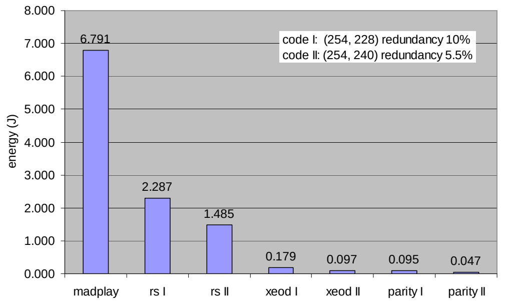


Figure 3.2: Energy Consumption

Figure 3.2 compares the energy consumption of the MP3 decoder and various FEC decoders. For each FEC scheme, two sets of code parameters $\big($I (254, 228) and II (254, 240)$\big)$ are examined as representatives. Data loss is always set equal to redundancy to explore maximum recovery capability. (Shorten technique is also used to achieve proper parameter for the XEOD and the PARITY code, as in a later section.) We can see that if the RS code is used in streaming, it will significantly increase total energy consumption (about $1/3$ over pure MP3 decoding). On the contrary, the additional energy consumed using the PARITY code or the XEOD would be almost negligible. Therefore, from energy perspective, the cost of using the RS code in wireless data streaming is high. Note that for the same decoder, code I consumes roughly twice as much energy as code II, which is reasonable because it needs to recover about twice as much data. Also note that it is not easy to directly measure the energy consumed only by FEC decoders in the simulation. Instead, we obtain the results by subtracting two measurements: 1) the energy consumption of data retrieval plus FEC decoding and 2) the energy consumption of data retrieval only.

3.2.2 Decoding Throughput

Besides data loss recovery, a complete wireless data streaming application usually includes many other functionalities, such as video/audio decoding, digital rights management, etc. To identify the performance bottleneck, it is desirable to compare the throughput of the FEC schemes to other possible components.

Here, we consider the following components: 1) media codec, choosing an open source MPEG-I decoder originated from UC Berkeley (SMPEG [47]) and 2) security component, choosing AES-128 and RC4 decoders (the fastest block cipher and stream ciphers [48]) from Wei Dai's Crypto++5.1 [49]. We use a 12-minute MPEG-I clip of the *Terminator2* at rate 1.5 Mbps as experiment data and perform throughput measurements on a P3 733 MHZ machine running Linux Redhat 7.3. Figure 3.3 shows that all FEC schemes have higher decoding throughputs than the MPEG-I decoder and thus will *not* be a bottleneck in a typical streaming application. Note that in this experiment, we use code parameters (360, 324) for all FEC schemes. Compared to parameters (254, 240) or (254, 228) as in the previous subsection, this configuration causes more difficulty for the RS code. The RS code now has to operate in a much larger finite field $GF(2^{16})$, as opposed to $GF(2^8)$. This certainly contributes to significant throughput difference between the RS code and the other two codes. It is desirable to repeat the same experiments on a real handheld device to verify our conclusion. We defer this to future work, though.

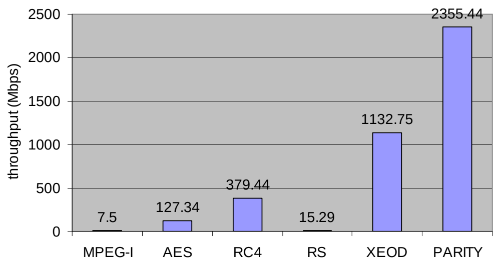


Figure 3.3: Decoding Throughput Comparison

3.3 Data Recovery (I): Random Symbol Loss

To study the loss recovery capability of the XEOD, this section compares it with the PARITY code and the RS code. Here, the random loss model is that each symbol has an equal and independent loss probability q.

3.3.1 Performance Analysis

For the PARITY code and the XEOD, we can analyze the decoding procedure by viewing it as a discrete random process and apply the approach discussed in [41], which we summarize as follows:

A codeword block is represented as a bipartite graph, with each link connecting a message node on one half plane to its check node on the other half. Links adjacent to a node of degree i are denoted as *links of degree* i. Let λ_i be the fraction of links of degree i corresponding to message nodes and ρ_i the fraction corresponding to check nodes. Then define two polynomials $\lambda(x) = \sum_i \lambda_i x^{i-1}$ and $\rho(x) = \sum_i \rho_i x^{i-1}$. The fraction of unrecoverable message nodes is:

$$r(x) = q(1-q)\lambda(q + (1-q)x) \times [x - 1 + \rho(1 - q\lambda(q + (1-q)x))]$$

where x is the smallest value satisfies:

$$\rho(1 - q\lambda(q + (1-q)x)) > 1 - x, \quad x \in (0, 1]$$

For a PARITY code with p message symbols, its $\lambda(x)$ and $\rho(x)$ can be calculated as follows: each message node participates parity calculation just once, thus $\lambda_1 = 1$ and $\lambda_i = 0$ for all $i \neq 1$. Since p is the width of message block ($p+1$ is the width of codeword block), and every check node has p links, $\rho_p = 1$ and $\rho_i = 0$ for all $i \neq p$. Therefore, $\lambda(x) = 1$ and $\rho(x) = x^{p-1}$.

For the XEOD, it is a little bit more complex to compute the polynomials. First, the EVENODD adjuster is assumed to be always available, as in early sections. Then, check nodes are categorized into three types based on their degrees and check equations, as shown in Figure 3.4. Therefore, we get the following polynomials:

$$\lambda(x) = x$$
$$\rho(x) = \frac{p-2}{2(p-1)}x^{p-3} + \frac{p}{2(p-1)}x^{p-2}$$

Note that the EVENODD code is *shortened* by not using the p^{th} column to achieve the same coding rate as the PARITY code. Code shortening will be discussed with more details in a later section.

With these two polynomials defined for the PARITY code and the XEOD, it is straightforward to calculate the largest feasible value of x and then compute the fraction of unrecoverable message nodes.

An (n, k) RS code can *not* decode at all if less than k symbols are received. Thus, a *systematic* code is always desirable so that at least received message symbols are still useful even if decoding fails. We use systematic RS codes in our analysis. Let m_1 be the number of lost message symbols and m_2 the number of lost check symbols, then the joint loss probability $P(m_1, m_2)$ in this case is:

$$P(m_1, m_2) = \binom{k}{m_1} q^{m_1}(1-q)^{k-m_1} \binom{n-k}{m_2} q^{m_2}(1-q)^{n-k-m_2}$$

Also define *normalized unrecoverable ratio* (denoted by r) as the performance metric, which represents the ratio between the number of unrecoverable message symbols and the total number of loss

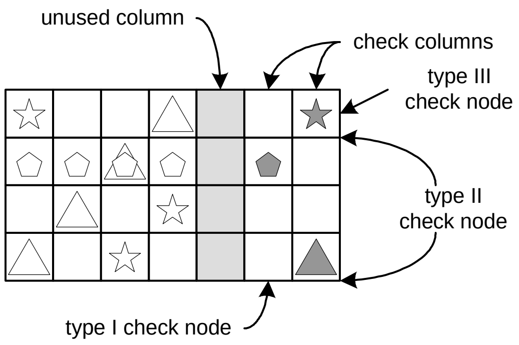


Figure 3.4: Random Loss Analysis of the XEOD

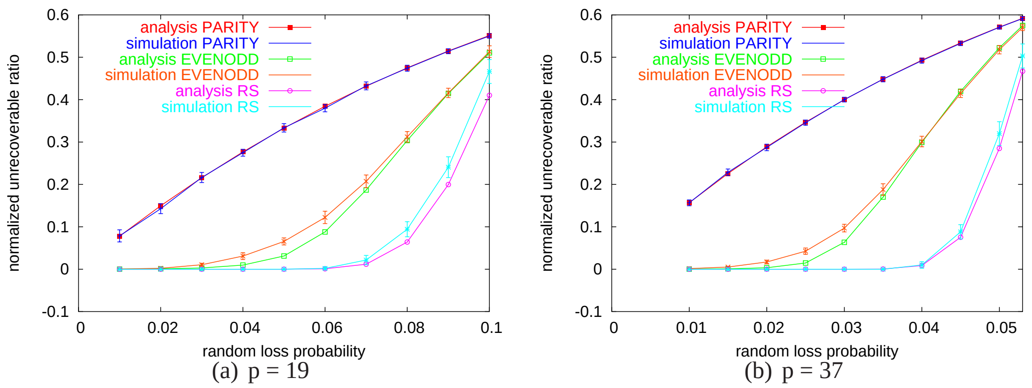


Figure 3.5: Random Loss Analysis

symbols in a codeword block. We can then compute r for the RS codes:

$$r = \sum_{m_1, m_2} \frac{m_1}{m_1 + m_2} \times P(m_1, m_2), \quad m_1 + m_2 > n - k$$

3.3.2 Performance Results

To compare the performance fairly, same block length and coding rate are chosen for all three codes. The PARITY code has height $2(p-1)$ and width $\frac{(p+1)}{2}$. The shortened EVENODD code has height $(p-1)$ and width $(p+1)$. For the RS code, only the block length matters, which is $(p-1)(p+1)$ here. Therefore, the coding rate is $\frac{p-1}{p+1}$, the same for all the three codes.

Figure 3.5 shows both analysis and simulation results of the normalized unrecoverable ratio for all three codes with respective to various random loss probability. The XEOD always outperforms the PARITY code. This is because each node in the EVENODD code participates in the calculation of two check nodes, which results in higher recovery chance. Also notice that the RS code has better performance than the XEOD, which is expected because the RS code is a MDS code at the symbol level, while the EVENODD code is MDS only at the column level.

Note that there is a small gap between the analysis and simulation results for the XEOD. This is due to the assumption that the EVENODD adjuster is always available to simplify analysis. This assumption makes the EVENODD code a little bit stronger and thus results in a slightly lower unrecoverable ratio in the analysis.

3.4 Data Recovery (II): Bursty Symbol Loss

3.4.1 A Realistic Data Loss Model: Bursty Loss

In the previous section, we use random data loss as a transmission model, which simplifies analysis. In wireless networks, however, data losses often occur in bursty manner. Hence, a more realistic loss model needs to take into account the dependency between packet losses. A two state Gilbert model [50] can represent the burstiness reasonably well. With this model, the network is either in a GOOD (G) state representing a packet reaches destination, or in a BAD (B) state representing a packet loss. Network state changes from state B to G with probability β and remains in state B with probably $(1-\beta)$. Similarly, state remains in state G with probability $(1-\alpha)$ and changes to B with probability α. It is easy to verify that the *stationary loss rate* is $\pi_B = \frac{\alpha}{\alpha+\beta}$ and the average length of consecutive BAD states, i.e., the *average burst length*, is $\mu_B = \frac{1}{\beta}$. The value of α and β can be derived by measuring π_B and μ_B in a real network environment.

3.4.2 Loss Recovery Performance of the FECs

It is not difficult to analyze the unrecoverable ratio of the RS code for a bursty loss model. We use a recursive approach here.

Let $P_{s_0,s_n}(k,n)$ be the probability of k symbol losses out of n total symbols, beginning from state s_0 and ending at state s_n. Therefore, we can get following recursive equations when the initial state is G:

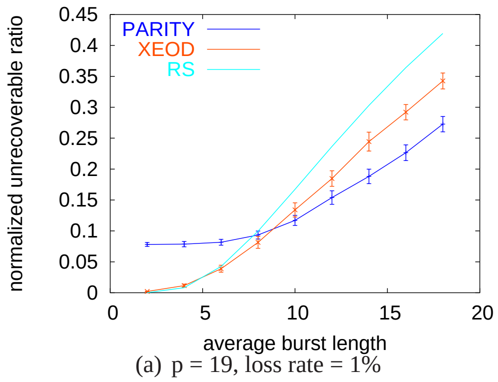
$$\begin{aligned}
P_{s_0=G,s_n=G}(k,n) =& P_{s_0=G,s_{n-1}=G}(k,n-1)\times(1-\alpha) + P_{s_0=G,s_{n-1}=B}(k,n-1)\times\beta \\
P_{s_0=G,s_n=B}(k,n) =& P_{s_0=G,s_{n-1}=G}(k-1,n-1)\times\alpha + P_{s_0=G,s_{n-1}=B}(k-1,n-1)\times(1-\beta)
\end{aligned}$$

with $P_{s_0=G,s_k=G}(k,k)=0$ and $P_{s_0=G,s_k=B}(k,k)=\alpha\times(1-\beta)^{k-1}$.

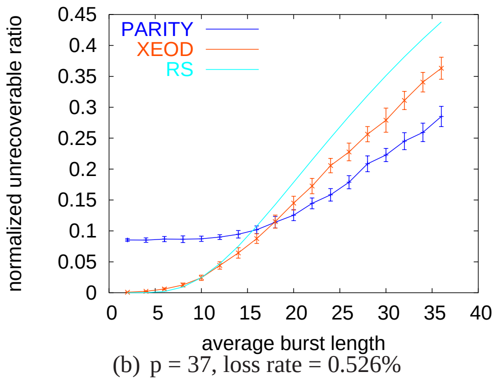
And also when the initial state is B:

$$\begin{aligned}
P_{s_0=B,s_n=G}(k,n) =& P_{s_0=B,s_{n-1}=G}(k,n-1)\times(1-\alpha) + P_{s_0=B,s_{n-1}=B}(k,n-1)\times\beta \\
P_{s_0=B,s_n=B}(k,n) =& P_{s_0=B,s_{n-1}=G}(k-1,n-1)\times\alpha + P_{s_0=B,s_{n-1}=B}(k-1,n-1)\times(1-\beta)
\end{aligned}$$

with $P_{s_0=B,s_k=G}(k,k)=0$ and $P_{s_0=B,s_k=B}(k,k)=(1-\beta)^k$.



(a) p = 19, loss rate = 1%



(b) p = 37, loss rate = 0.526%

Figure 3.6: Burst Loss Analysis

Thus, the probability $P(k,n)$ of k symbol losses out of total n symbols is:

$$\begin{aligned} P(k,n) = (1-\pi_B) \times (P_{s_0=G,s_n=G}(k,n) + P_{s_0=G,s_n=B}(k,n)) \\ + \pi_B \times (P_{s_0=B,s_n=G}(k,n) + P_{s_0=B,s_n=B}(k,n)) \end{aligned}$$

It is, however, much more difficult to get closed-form representation of the unrecoverable ratio of the PARITY code or the XEOD for a bursty loss model, if possible at all. This is mainly because the recovery capability depends heavily on actual loss pattern in each codeword block, which is extremely difficult to count. Therefore, we use simulation to measure the burst loss recovery capability of the PARITY code and the XEOD. For each simulation, we let the first 10,000 states of the Gilbert model pass to ensure our experiments always start from a steady state. Then, 1,000,000 codeword blocks are generated and decoded. The unrecoverable ratio is calculated as the average over all codeword blocks. Figure 3.6 shows the simulation results of the PARITY code and the XEOD, which are also compared with the analysis results of the RS code. Loss rate for each case is simulated to be $\frac{1}{10}$ of the redundancy.

It is worth pointing out that the performance of the RS code is worse than the XEOD under various burst patterns when loss rate is relatively low. The explanation is that in a bursty network, symbol losses tend to group closer together with longer gaps between groups than in random loss situation. Whenever there are more than $(n-k)$ symbol losses in a codeword block of a (n,k) RS code, the decoder fails to solve necessary linear equations due to too many unknowns. Thus none of these lost packets can be recovered. In contrast, each subset of symbols can do their own decoding in the PARITY code and the XEOD. This provides higher chance to recover symbol losses in some circumstances. Hence, the RS code, although optimal for random data loss, is not necessarily the best choice even not considering energy consumption and decoding throughput.

Also, the performance of the PARITY code is not always poor in bursty losses. When the average burst length goes beyond a *cross point* with the XEOD, the PARITY code actually yields

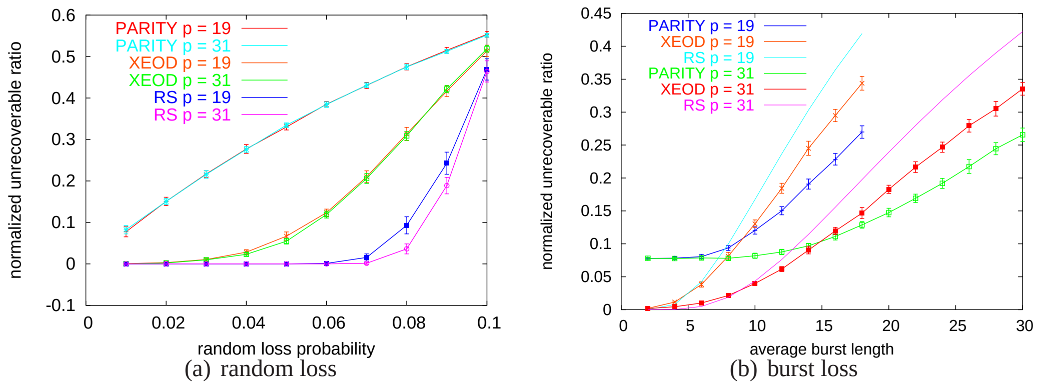


Figure 3.7: Effect of Shortening

better loss recovery performance. This justifies that the PARITY code is still an effective approach for loss recovery.

3.5 Effects of Parameters on the XEOD

This section discusses the effects of parameter p on the XEOD. Fair comparisons among codes constructed with different p values are achieved by *shortening* codes with larger ps such that all the codes have the same coding rate. Shortening is a common practice to adjust the rate of a code without changing its loss recovery capability, by setting certain information symbols to zero [4]. For a code with k information and n total columns, a simple shortening example is to set the entire last column to zero, and the coding rate decreases from $\frac{k}{n}$ to $\frac{k-1}{n-1}$. Shortened codes can have the same coding rate but different block length.

Figure 3.7(a) shows the random loss recovery results of the PARITY code, the XEOD and the RS code. It is clear that shortening has only marginal effect on codes' recovery capability if data happens randomly. Figure 3.7(b) compares the burst loss recovery of the three codes. For each type of code, its loss recovery capability increases as p increases (block length also increases). For the same p, the relative loss recovery capability remains the same among the PARITY code, the XEOD and the RS code, i.e., the XEOD outperforms the RS code and is better than the PARITY code for short bursts. Notice that larger p yields better loss recovery performance, but at the cost of larger codeword block length, which in general demands more buffer space usage and longer decoding delay. Hence, a general rule to decide p value is to push p to the maximum value limited by recovery buffer and delay constraints.

Chapter 4

ORC: Optimal Coding Rate Control for Scalable Streaming

Perhaps the major technical problem in streaming media on demand over the Internet is the need to adapt to changing network conditions. As competing communication processes begin and end, the available bandwidth, packet loss and packet delay all fluctuate. Network outages lasting many seconds can and do occur. Resource reservation and quality of service support can help, but even they cannot guarantee that network resources will be stable. If the network path contains a wireless link, for example, its capacity may be occasionally reduced by interference. Thus it is necessary for commercial-grade streaming media systems to be robust to hostile network conditions. Moreover, such robustness cannot be achieved solely by aggressive (nonreactive) transmission. Even constant bit rate transmission with retransmissions for every packet loss cannot achieve a throughput higher than the channel capacity. Some degree of adaptivity to the network is therefore required.

End users expect that a good streaming media system will exhibit the following behavior: content played back on demand will start with low delay; once started, it will play back continuously (without stalling) unless interrupted by the user; and it will play back with the highest possible quality given the average communication bandwidth available. To meet these expectations in the face of changing network conditions, buffering of the content at the client before decoding and playback is required.

Buffering at the client serves several distinct but simultaneous purposes. First, it allows the client to compensate for short-term variations in packet transmission delay (i.e., “jitter”). Second, it gives the client time to perform packet loss recovery if needed. Third, it allows the client to continue playing back the content during lapses in network bandwidth. And finally, it allows the content to be coded with variable bit rate, which can dramatically improve overall quality.[1] By controlling the size of the client buffer over time it is possible for the client to meet the above

[1]Note that even so-called constant bit rate (CBR) coded content is actually coded with variable bit rate within the constraints of a decoding buffer of a given size. The larger the decoding buffer size, the better the quality. The required decoding buffering is part of the larger client buffer.

mentioned user expectations. If the buffer is initially small, it allows a low startup delay. If the buffer never underflows, it allows continuous playback. If the buffer is eventually large, it allows eventual robustness as well as high, nearly constant quality. Thus, client buffer management is a key element affecting the performance of streaming media systems.

The size of the client buffer can be expressed as the number of seconds of content in the buffer, called the buffer *duration*. The buffer duration tends to increase as content enters the buffer and tends to decrease as content leaves the buffer. Content leaves the buffer when it is played out, at a rate of ν seconds of content per second of real time, where ν is the *playback speed* (typically 1 for normal playback, but possibly more than 1 for high speed playback or less than 1 for low speed playback). Content enters the buffer when it arrives at the client over the network, at a rate of r_a/r_c seconds of content per second of real time, where r_a is the *arrival rate*, or average number of bits that arrive at the client per second of real time, and r_c is the *coding rate*, or average number of bits needed to encode one second of content. Thus the buffer duration can be increased by increasing r_a, decreasing r_c, and/or decreasing ν (and vice versa for decreasing the buffer duration). Although the buffer duration can be momentarily controlled by changing r_a (cf. "Fast Start" in Windows Media 9 [51]) or changing ν (cf. "Adaptive Media Playout (AMP)" in [52]), these quantities are generally not possible to control freely for long periods of time. The arrival rate r_a on average is determined by the network capacity, while the playback speed ν on average is determined by user preference. Thus if the network capacity drops dramatically for a sustained period, reducing the coding rate r_c is the only appropriate way to prevent a *rebuffering event* in which playback stops ($\nu = 0$) while the buffer refills.

Thus, adaptivity to changing network conditions requires not only a buffer, but also some means to adjust the coding rate r_c of the content. This can be done by stream switching in combination with multi bit rate (MBR) coding or coarse grained or fine grained scalable coding. Today's commercial streaming media systems [51, 53] rely on MBR coding as well as *thinning*, which is a form of coarse grained scalability.[2] Future commercial systems may support fine grained scalability (FGS) as well.[3] FGS coding offers great flexibility in adapting to variable network conditions, and can demonstrably improve quality under such conditions.

In this chapter we focus on the problem of *coding rate control*, that is, dynamically adjusting the coding rate of the content to control the buffer duration. Outside the scope of this chapter is the problem of transmission rate control. The *transmission rate* r_x is the rate at which the sender application injects bits into the transport layer and is equal to the arrival rate r_a on average if the

[2]In MBR coding, semantically identical content is encoded into alternative bit streams at different coding rates and stored in the same media file at the server, allowing the content to be streamed at different levels of quality corresponding to the coding rates r_c, possibly using bit stream switching [54]. In coarse grained scalable coding (such as MPEG-2/4 temporal or SNR scalability [55]) the content is encoded into several substreams or *layers*, so that the coding rate r_c can be changed in large deltas by adding or dropping (at possibly restricted times) one layer of content at a time. Thinning is a special case of coarse grained scalability in which dependent video frames (P and B frames) are dropped before independent video frames (I frames), which are in turn are dropped before audio frames.

[3]Fine grained scalable coding (such as 3D SPIHT [56], MPEG-4 FGS [57], or EAC [58]) allows the coding rate r_c to change at any time in deltas sometimes as small as one byte per presentation.

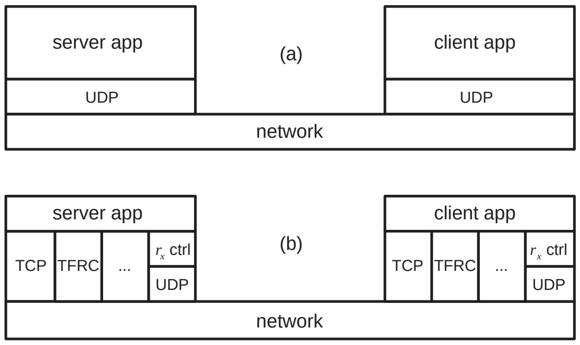


Figure 4.1: (a) Traditional streaming media architecture. (b) Proposed streaming media architecture with congestion control factored out.

transport is lossless. By *transmission rate control* we mean congestion control as well as any other mechanisms affecting the transmission rate such as bursting, tracking the transmission rate to the available bandwidth, and so on. Thus we control the buffer duration by adjusting the coding rate r_c at which bits leave the buffer, while letting the the arrival rate r_a at which bits enter the buffer be determined by other means.

In the streaming media literature, with few exceptions (e.g., [59, 60] and the works based thereon; also [11]), there has been little attention paid to the the distinction between the coding rate r_c and the arrival rate r_a or the transmission rate r_x. Indeed, in typical streaming media systems (e.g., [51]), after an initial buffering period (in which $\nu = 0$ and possibly $r_x/r_c > 1$), r_x/r_c is locked to ν. A difficulty with locking the transmission rate to the coding rate via the playout speed is that it essentially removes any means of controlling the client buffer duration after the initial buffering period.[4] A further difficulty is that the transmission rate, if it is locked to the coding rate, will typically be incompatible with transports that use standard congestion control, such as TCP and TFRC [61].

By decoupling the coding and transmission rates, it is possible to continually control the client buffer duration. This allows the buffer to grow over time, for example, providing a low startup delay, asymptotically high robustness, and eventual constant quality. Furthermore, decoupling the coding and transmission rates makes possible an architecture in which the transport and congestion control protocol may be factored out of the streaming problem, if desired. Figure 4.1(a) illustrates the traditional architecture in which congestion control is integrated into the streaming media application running on top of UDP. Figure 4.1(b) illustrates the proposed architecture in which congestion control is factored out of the streaming media application, allowing standard transport mechanisms (such as TCP and TFRC) to be used, as well as custom transport solutions using custom transmission rate control over UDP [62–65].

[4]However, congestion, as evidenced by a drop in r_a and hence a drop in the buffer duration, can still be alleviated by reducing r_x and r_c by the same factor.

In addition to factoring the problem of network adaptation into transmission rate control and coding rate control, the novelty of our approach lies in the following two aspects. First, we formulate the problem of coding rate control as a standard problem in linear quadratic optimal control, in which the client buffer duration is controlled as closely as possible to a target level while keeping the coding rate (and hence the quality) as constant as possible. To our knowledge this is the first use of optimal control theory for client buffer management. Second, we explicitly take into consideration, using a leaky bucket model, the natural variation in the instantaneous coding rate that occurs for a given average coding rate. We incorporate the leaky bucket model into the control loop so that the changes in buffer duration due to natural variation in the instantaneous coding rate are not mistaken for changes in buffer duration due to network congestion. To our knowledge this is also the first use of a leaky bucket to model source coding rate constraints during client buffer management beyond the initial startup delay.[5]

4.1 Problem Formulation

4.1.1 Temporal Coordinate Systems

It will pay to distinguish between the temporal coordinate systems, or clocks, used to express time. In this chapter, *media time* refers to the clock running on the device used to capture and timestamp the original content, while *client time* refers to the clock running on the client used to play back the content. We assume that media time is real time (i.e., one second of media time elapses in one second of real time) at the time of media capture, while client time is real time at the time of media playback. We use the symbol τ to express media time and the symbol t to express client time, with subscripts and other arguments to indicate corresponding events. For example, we use $\tau_d(0), \tau_d(1), \tau_d(2), \ldots$ to express the playback deadlines of frames $0, 1, 2, \ldots$ in media time, while we use $t_d(0), t_d(1), t_d(2), \ldots$ to express the playback deadlines of frames $0, 1, 2, \ldots$ at the client. Content may be played back at a rate ν times real time. Thus the conversion from media time to client time can be expressed

$$t = t_0 + \frac{\tau - \tau_0}{\nu}, \tag{4.1}$$

where t_0 and τ_0 represent the time of a common initial event, such as the playback of frame 0 (or the playback of the first frame after a seek or rebuffering event) in media and client coordinate systems, respectively.

4.1.2 Leaky Bucket Model

For the moment we revert to a scenario in which both the encoder and the decoder run in real time over an isochronous communication channel. In this case, to match the instantaneous coding rate to

[5]Ribas, Chou, and Regunathan use a leaky bucket to model source coding rate constraints to reduce initial startup delay [66], while Hsu, Ortega and Reibman use a leaky bucket to model transmission rate contraints [5].

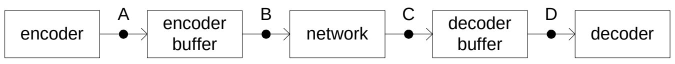


Figure 4.2: Communication pipeline.

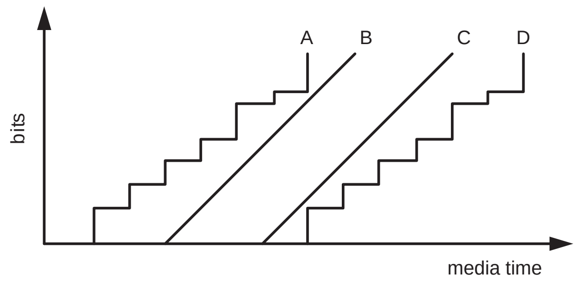


Figure 4.3: Schedules at which bits in the coded bit stream pass the points A, B, C, and D in the communication pipeline.

the instantaneous channel rate, an *encoder buffer* is required between the encoder and the channel and a *decoder buffer* is required between the channel and the decoder, as illustrated in Figure 4.2. A *schedule* is the sequence of times at which successive bits in the coded bit stream pass a given point in the communication pipeline. Figure 4.3 illustrates the schedules of bits passing the points A, B, C, and D in Figure 4.2. Schedule A is the schedule at which captured frames are instantaneously encoded and put into the encoder buffer. This schedule is a staircase in which the nth step rises by $b(n)$ bits at time $\tau(n)$, where $\tau(n)$ is the time at which frame n is encoded, and $b(n)$ is the number of bits in the resulting encoding. Schedules B and C are the schedules at which bits respectively enter and leave the communication channel. The slope of these schedules is R bits per second, where R is the communication rate of the channel. Schedule D is the schedule at which frames are removed from the decoder buffer and instantaneously decoded for presentation. Note that Schedule D is simply a shift of Schedule A. Note also that Schedule B is a lower bound to Schedule A, while Schedule C is an upper bound to Schedule D. Indeed, the gap between Schedules A and B represents, at any point in time, the size in bits of the encoder buffer, while the gap between Schedules C and D likewise represents the size of the decoder buffer. The encoder and decoder buffer sizes are complementary. Thus the coding schedule (either A or D) can be contained within a *buffer tube*, as illustrated in Figure 4.4, having slope R, height B, and initial offset F^d from the top of the tube (or equivalently initial offset $F^e = B - F^d$ from the bottom of the tube). It can be seen that $D = F^d/R$ is the *startup delay* between the time that the first bit arrives at the receiver and the first frame is decoded. Thus it is of interest to minimize F^d for a given R.

A *leaky bucket* is a metaphor for the encoder buffer. The encoder dumps $b(n)$ bits into the leaky bucket at time $\tau(n)$, and the bits leak out at rate R. In general it is possible for the leak

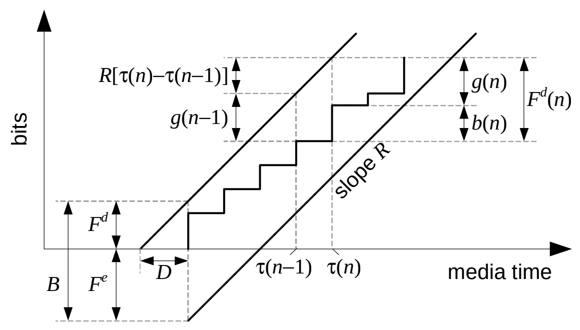


Figure 4.4: Buffer tube containing a coding schedule.

rate R to be high enough so that the bucket occasionally empties. Thus the encoder buffer fullness $F^e(n)$ immediately before frame n is added to the bucket and the encoder buffer fullness $B^e(n)$ immediately after frame n is added to the bucket evolve from an initial encoder buffer fullness $F^e(0) = F^e$ according to the dynamical system

$$B^e(n) = F^e(n) + b(n), \tag{4.2}$$

$$F^e(n+1) = \max\{0, B^e(n) - R/f(n)\}, \tag{4.3}$$

where

$$f(n) = \frac{1}{\tau(n+1) - \tau(n)} \tag{4.4}$$

is the instantaneous frame rate, for $n = 0, 1, 2, \ldots$. If R is sufficiently low, then the bucket will never run dry (underflow), but if R is too low the bucket will eventually overflow. We take the largest R such that the buffer will never run dry to be the average coding rate r_c of the bit stream. This is made more precise in the following two paragraphs.

A leaky bucket with size B, rate R, and initial fullness F^e is said to *contain* a stream having a schedule characterized by the steps $\{(b(n), \tau(n))\}$ if $B^e(n) \leq B$ for all n. We define the minimum bucket size needed to contain the stream given leak rate R and initial fullness F^e as

$$B^e_{\min}(R, F^e) = \min_n B^e(n), \tag{4.5}$$

while we define the corresponding initial decoder buffer fullness as

$$F^d_{\min}(R, F^e) = B^e_{\min}(R, F^e) - F^e. \tag{4.6}$$

We denote the minimum of each of these over F^e as

$$B^e_{\min}(R) = \min_{F^e} B^e_{\min}(R, F^e), \tag{4.7}$$

$$F^d_{\min}(R) = \min_{F^e} F^d_{\min}(R, F^e). \tag{4.8}$$

It is shown in [66, Proposition 2] that remarkably, these are each minimized by the same value of F^e, which is hence equal to

$$F^e_{\min}(R) = B^e_{\min}(R) - F^d_{\min}(R). \tag{4.9}$$

Thus given a bit stream with schedule $\{(b(n), \tau(n))\}$, for each bit rate R there is a unique leaky bucket that contains the stream and that has the minimum buffer size B as well as the minimum startup delay $D = F^d/R$. These parameters can be computed with the above equations.

For sufficiently low leak rates R, the leaky bucket does not underflow, when beginning with initial fullness $F^e = F^e_{\min}(R)$. We may use the maximum such rate R as the average coding rate r_c of a bit stream with coding schedule $\{(b(n), \tau(n))\}$.

Leak rates R greater than r_c will also be used in this chapter. It is shown in [66] that both $B^e_{\min}(R)$ and $F^d_{\min}(R)$ are decreasing, piecewise linear, and convex in R. Hence if the transmission rate R is greater than the average coding rate r_c, the startup delay $D = F^d_{\min}(R)/R$ can be reduced compared to $D = F^d_{\min}(r_c)/R$. This fact will be used in Section 4.3.1.

A leaky bucket with leak rate $R = r_c$, size $B = B^e_{\min}(r_c)$ and initial decoder buffer fullness $F^d = F^d_{\min}(r_c)$ thus corresponds to a straight buffer tube bounding the coding schedule as in Figure 4.4. Each stream in the media file has a coding schedule; thus each stream corresponds to a straight buffer tube with slope equal to the average coding rate r_c of the stream. The size B of the buffer tube and its offset F^e (or F^d) relative to the coding schedule can be either computed by the above formula for a variable bit rate (VBR) stream (such as a constant-quality substream of a scalable stream), or obtained from the size B and initial state F^e of the actual encoder buffer used to encode the stream if it is a constant bit rate (CBR) stream.

In the sequel we will need to consider the gap $g(n)$ at frame n between the buffer tube *upper bound* and the coding schedule, as depicted in Figure 4.4. Note that the decoder buffer fullness $F^d(n) = B - F^e(n)$ can also be expressed

$$F^d(n) = b(n) + g(n) = g(n-1) + \frac{r_c(n)}{f(n)}, \tag{4.10}$$

where $r_c(n)$ is the coding rate of the buffer tube, now taking into account that different frames may lie in different buffer tubes with different coding rates as coding rate control is applied and streams are switched.

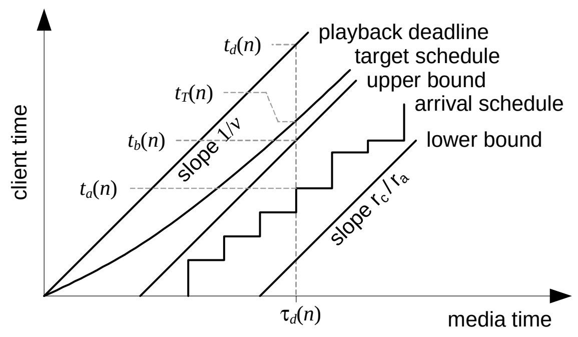


Figure 4.5: Arrival schedule and its upper bound in client time.
The upper bound is controlled to the target schedule, which is increasingly in advance of the playback deadline to provide greater robustness over time.

4.1.3 Rate Control Model

Assume for the moment that bits arrive at the client at a constant rate r_a. Then frame n (having size $b(n)$) arrives at the client $b(n)/r_a$ seconds after frame $n-1$. Indeed, the index of a bit is proportional to its arrival time. Dividing the vertical scale of the schedules in Figure 4.4 by r_a, we obtain the schedules in terms of client time, rather than bits, as shown in Figure 4.5. The coding schedule divided by r_a becomes the *arrival schedule*, which provides for each n the time $t_a(n)$ of arrival of frame n at the client. The buffer tube upper bound (in bits) divided by r_a becomes the buffer tube upper bound (in time), which provides for each n the time $t_b(n)$ by which frame n is guaranteed to arrive. In the same plot we show the *playback deadline,* which is the time $t_d(n)$ at which frame n is scheduled to be played (after instantaneous decoding). Thus the gap between a frame's arrival time and its playback deadline is the client buffer duration at the time of the frame arrival. This must be non-negative to allow continuous playback.

In reality the arrival rate is not constant. If $t_a(n-1)$ and $t_a(n)$ are the arrival times of frames n and $n-1$ respectively, then we may define

$$r_a(n) = \frac{b(n)}{t_a(n) - t_a(n-1)} \tag{4.11}$$

to be the *instantaneous arrival rate* at frame n. In practice we estimate the average arrival rate at frame n by a moving average $\tilde{r}_a(n)$ of previous values of $r_a(n)$, as detailed in Section 4.3.3. Hence using (4.11) we may express the arrival time of frame n in terms of the arrival time of frame $n-1$ as

$$\begin{aligned} t_a(n) &= t_a(n-1) + \frac{b(n)}{r_a(n)} && (4.12) \\ &= t_a(n-1) + \frac{b(n)}{\tilde{r}_a(n)} + v(n), && (4.13) \end{aligned}$$

where the $v(n)$ term is an error term that captures the effect of using the slowly moving average $\tilde{r}_a(n)$ instead of the instantaneous arrival rate $r_a(n)$. From (4.10), however, we have

$$b(n) = \frac{r_c(n)}{f(n)} + g(n-1) - g(n), \tag{4.14}$$

whence (substituting (4.14) into (4.13)) we have

$$t_a(n) = t_a(n-1) + \frac{r_c(n)}{f(n)\tilde{r}_a(n)} + \frac{g(n-1)}{\tilde{r}_a(n)} - \frac{g(n)}{\tilde{r}_a(n)} + v(n). \tag{4.15}$$

Now defining the buffer tube upper bound (in time) of frame n as

$$t_b(n) = t_a(n) + \frac{g(n)}{\tilde{r}_a(n)}, \tag{4.16}$$

so that

$$t_b(n) - t_b(n-1) = t_a(n) - t_a(n-1) + \frac{g(n)}{\tilde{r}_a(n)} - \frac{g(n-1)}{\tilde{r}_a(n-1)}, \tag{4.17}$$

we obtain the following update equation:

$$t_b(n) = t_b(n-1) + \frac{r_c(n)}{f(n)\tilde{r}_a(n)} + w(n-1), \tag{4.18}$$

where

$$w(n-1) = \frac{g(n-1)}{\tilde{r}_a(n)} - \frac{g(n-1)}{\tilde{r}_a(n-1)} + v(n) \tag{4.19}$$

is again an error term that captures variations around a locally constant arrival rate.

Using (4.16), the client can compute $t_b(n-1)$ from the measured arrival time $t_a(n-1)$, the estimated arrival rate $\tilde{r}_a(n-1)$, and $g(n-1)$ (which can be transmitted to the client along with the data in frame $n-1$ or computed at the client as described in Section 4.4.5). Then using (4.18), the client can control the coding rate $r_c(n)$ so that $t_b(n)$ reaches a desired value, assuming the frame rate and arrival rate remain roughly constant. From this perspective, (4.18) can be regarded as the state transition equation of a feedback control system and it is thus possible to use a control-theoretic approach to regulate the coding rate.

4.1.4 Control Objective

With the state transition equation defined in (4.18), uninterrupted playback can be achieved by regulating the coding rate so that the client buffer does not underflow. To introduce a margin of safety that increases over time, we introduce a *target schedule*, illustrated in Figure 4.5, whose distance from the playback deadline grows slowly over time. By regulating the coding rate, we attempt to control the buffer tube upper bound so that it tracks the target schedule. If the buffer tube upper bound is close to the target schedule, then the arrival times of all frames will certainly be earlier than their playback deadlines and thus uninterrupted playback will be ensured. Note that controlling the actual arrival times (rather than their upper bounds) to the target would result in an approximately constant number of bits per frame, which would in turn result in very poor quality overall. By taking the leaky bucket model into account, we are able to establish a control that allows the instantaneous coding rate to fluctuate naturally according to the encoding complexity of the content, within previously established bounds for a given average coding rate.

Although controlling the upper bound to the target schedule is our primary goal, we also wish to minimize quality variations due to large or frequent changes to the coding rate. This can be achieved by introducing into the cost function a penalty for relative coding rate differences.

Letting $t_T(n)$ denote the target for frame n, we use the following cost function to reflect both of our concerns:

$$I = \sum_{n=0}^{N} \left(\big(t_b(n) - t_T(n)\big)^2 + \sigma \left(\frac{r_c(n+1) - r_c(n)}{\tilde{r}_a(n)} \right)^2 \right), \tag{4.20}$$

where the first term penalizes the deviation of the buffer tube upper bound from the target schedule and the second term penalizes the relative coding rate difference between successive frames. N is the control window size and σ is a Lagrange multiplier or weighting parameter to balance the two terms.

4.2 Optimal Control Solution

Before presenting the optimal control solution, we first describe the design rational of the target schedule.

4.2.1 Target Schedule Design

Figure 4.6 shows an illustrative target schedule. The gap between the playback deadline and the target schedule is the desired client buffer duration (in client time). If the gap is small at the beginning of streaming, then it allows a small startup delay, while if the gap grows slowly over time, it gradually increases the receiver's ability to counter jitter, delays, and throughput changes.

The slope of the target schedule relates the average coding rate to the average arrival rate. Let $t_T(n)$ be the target for frame n. As illustrated in Figure 4.6, the slope of the target schedule at frame n is

$$s(n) = \frac{t_T(n+1) - t_T(n)}{\tau(n+1) - \tau(n)}. \tag{4.21}$$

If the upper bound $t_b(n)$ aligns perfectly with the target schedule (i.e., $t_b(n) = t_T(n)$) and the arrival rate r_a is constant (i.e., the $w(n-1)$ term vanishes), we get from (4.18)

$$s(n) = \frac{t_b(n+1) - t_b(n)}{\tau(n+1) - \tau(n)} = \frac{r_c(n)}{r_a}. \tag{4.22}$$

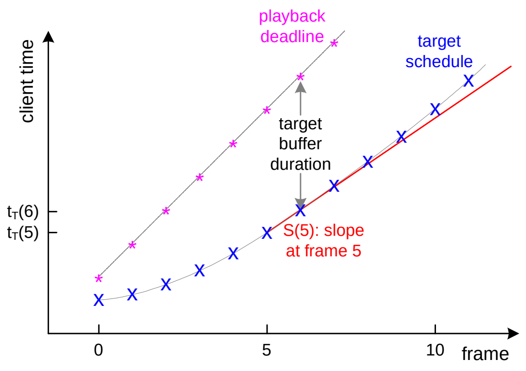


Figure 4.6: Target schedule design.

Thus initially, when the slope is low, i.e., less than $1/\nu$, r_a/r_c is greater than ν and more than ν seconds of content are received per second of client time, causing the client buffer (which is playing out only ν seconds of content per second of client time) to grow. Over time, as the slope approaches $1/\nu$, r_a/r_c approaches ν and the buffer remains relatively constant (except for changes due to variations in the instantaneous coding rate), since content is received and played back at the same speed ν. We next present two target schedule functions that illustrate the general design idea.

Logarithmic Target Schedule

One way to choose the target schedule t_T is to have the client buffer duration grow logarithmically over time. Specifically, if t_d is the playback deadline, then for each t_d greater than some start time t_{d0},

$$t_T = t_d - \frac{b}{a}\ln(a(t_d - t_{d0}) + 1). \tag{4.23}$$

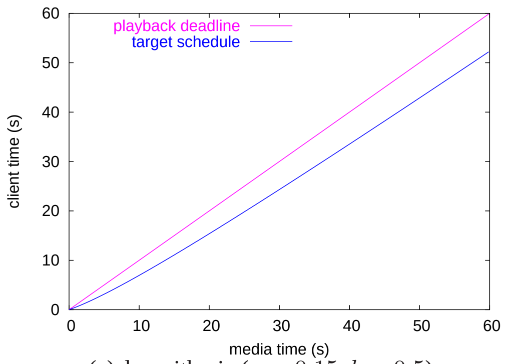
Since by (4.1), $t_d = t_{d0} + (\tau_d - \tau_{d0})/\nu$, we have

$$s = \frac{dt_T}{d\tau_d} = \frac{dt_T}{dt_d}\frac{dt_d}{d\tau_d} = \frac{1}{\nu} - \frac{b}{a(\tau_d - \tau_{d0}) + \nu}, \tag{4.24}$$

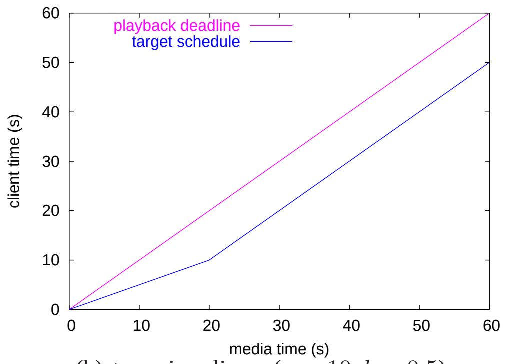
and hence the initial slope at frame 0 (when $t_d = t_{d0}$) is $s(0) = (1-b)/\nu$. Setting $b = 0.5$ implies that initially $r_c/r_a = 0.5/\nu$, causing the client buffer to grow initially at two times real time. Further setting $a = 0.15$ implies that the client buffer duration will be 7.68 seconds after 1 minute, 15.04 seconds after 10 minutes, and 22.68 seconds after 100 minutes, regardless of ν.

Two-piece Linear Target Schedule

Another way to choose the target schedule t_T is to have the client buffer duration grow linearly at rate b seconds of media time per second of client time until the buffer duration reaches a seconds of media time, after which it remains constant. Specifically, for each t_d greater than some start time



(a) logarithmic ($a = 0.15$, $b = 0.5$)



(b) two-piece linear ($a = 10$, $b = 0.5$)

Figure 4.7: Target schedules.

t_{d0},

$$t_T = \begin{cases} t_d - b(t_d - t_{d0}) & t_d \le t_{d0} + a/b \\ t_d - a & t_d \ge t_{d0} + a/b \end{cases} . \tag{4.25}$$

The initial slope is again $s(0) = (1-b)/\nu$. Setting $b = 0.5$ implies that initially $r_c/r_a = 0.5/\nu$, causing the client buffer to grow initially at two times real time. Further setting $a = 10$ implies that the client buffer duration will reach 10 seconds of media time after 20 seconds of client time, regardless of ν.

Figure 4.7 shows the above two target schedules. As one can see, if a client buffer duration of 10 seconds is considered to be a safe level against jitter, delay and network fluctuations, then the two-piece linear target schedule reaches the safe level in 20 seconds, much faster than the logarithmic target schedule. On the other hand, the slope of the two-piece linear target schedule remains lower for longer (hence the coding rate and quality are lower for longer) and furthermore experiences an abrupt change at 20 seconds when its slope changes from $0.5/\nu$ to $1/\nu$. Consequently, the coding rate will not change as smoothly as with the logarithmic target schedule, although it will not be as abrupt as the schedule itself because of the smoothness objective in the controller design. Hence, we investigate the effect of both target schedules.

4.2.2 Optimal Controller Design

Recall from (4.18) the fundamental state transition equation, which describes the evolution of the buffer tube upper bound $t_b(n)$ in terms of the coding rate $r_c(n)$:

$$t_b(n+1) = t_b(n) + \frac{r_c(n+1)}{f\tilde{r}_a} + w(n). \tag{4.26}$$

Here we now assume that the frame rate f and the average arrival rate $\tilde{r}_a$ are relatively constant. Deviations from this assumption are captured by $w(n)$.

We wish to control the upper bound by adjusting the coding rate. As each frame arrives at the client, a feedback loop can send a message to the server to adjust the coding rate. Note, however, that by the time frame n arrives completely at the client, frame $n+1$ has already started streaming from the server. Thus the coding rate $r_c(n+1)$ for frame $n+1$ must already be determined by time $t_a(n)$. Indeed, at time $t_a(n)$, frame $n+2$ is the earliest frame for which the controller can determine the coding rate. Hence at time $t_a(n)$, the controller's job must be to choose $r_c(n+2)$. We must explicitly account for this one-frame delay in our feedback loop.

For simplicity, we linearize the target schedule around the time that frame n arrives. The linearization is equivalent to using a line tangent to the original target schedule at a particular point as an approximate target schedule. Thus we have

$$t_T(n+1) - 2t_T(n) + t_T(n-1) = 0. \tag{4.27}$$

Rather than directly control the evolution of the upper bound, which grows without bound, for the purposes of stability we use an error space formulation. By defining the error

$$e(n) = t_b(n) - t_T(n), \tag{4.28}$$

we obtain

$$\begin{aligned}
&e(n+1) - e(n) \\
&= (t_b(n+1) - t_T(n+1)) - (t_b(n) - t_T(n)) && (4.29)\\
&= (t_b(n+1) - t_b(n)) - (t_T(n+1) - t_T(n)) && (4.30)\\
&= \frac{r_c(n+1)}{f\tilde{r}_a} - (t_T(n+1) - t_T(n)) + w(n), && (4.31)
\end{aligned}$$

from which we obtain in turn

$$\begin{aligned}
&(e(n+1) - e(n)) - (e(n) - e(n-1)) \\
&\quad = \quad [r_c(n+1) - r_c(n)]/f\tilde{r}_a \\
&\qquad\quad -(t_T(n+1) - 2t_T(n) + t_T(n-1)) \\
&\qquad\quad +(w(n) - w(n-1)) && (4.32)\\
&\quad = \quad \frac{r_c(n+1) - r_c(n)}{f\tilde{r}_a} + (w(n) - w(n-1)). && (4.33)
\end{aligned}$$

We next define the control input

$$u(n) = \frac{r_c(n+2) - \hat{r}_c(n+1)}{\tilde{r}_a}, \tag{4.34}$$

where $\hat{r}_c(n+1)$ is a possibly quantized version of $r_c(n+1)$ (as defined in Section 4.3.4) and we define the disturbance

$$d(n) = \frac{\hat{r}_c(n) - r_c(n)}{f\tilde{r}_a} + w(n) - w(n-1). \tag{4.35}$$

Then (4.33) can be rewritten

$$e(n+1) = 2e(n) - e(n-1) + \frac{u(n-1)}{f} + d(n). \tag{4.36}$$

Therefore, defining the error vector

$$\mathbf{e}(n) = \begin{bmatrix} e(n) \\ e(n-1) \\ u(n-1) \end{bmatrix} = \begin{bmatrix} t_b(n) \\ t_b(n-1) \\ \frac{r_c(n+1)}{\tilde{r}_a} \end{bmatrix} - \begin{bmatrix} t_T(n) \\ t_T(n-1) \\ \frac{\hat{r}_c(n)}{\tilde{r}_a} \end{bmatrix}, \tag{4.37}$$

the error space representation of the system can be expressed

$$\mathbf{e}(n+1) = \begin{bmatrix} 2 & -1 & \frac{1}{f} \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{e}(n) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(n) + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} d(n), \tag{4.38}$$

or $\mathbf{e}(n+1) = \Phi\mathbf{e}(n) + \Gamma u(n) + \Gamma_d d(n)$ for appropriate matrices Φ, Γ and Γ_d.

Assuming the disturbance $d(n)$ is a pure white noise, and assuming *perfect state measurement* (i.e., we can measure all components of $\mathbf{e}(n)$ without using an estimator), the disturbance $d(n)$ does *not* affect the controller design. Thus we can use a linear controller represented by

$$u(n) = -G\mathbf{e}(n), \tag{4.39}$$

where G is a *feedback gain*. By the time frame n is completely received, all elements of $\mathbf{e}(n)$ are available at the client and $u(n)$ can thus be computed. The ideal coding rate for frame $n+2$ can then be computed as

$$r_c(n+2) = \hat{r}_c(n+1) - G\mathbf{e}(n)\tilde{r}_a. \tag{4.40}$$

Finding the optimal linear controller amounts to finding the feedback gain G^* that minimizes the quadratic cost function defined in Section 4.1.4. Before continuing with the design, we first check the system *controllability matrix* $\mathcal{C}$,

$$\mathcal{C} = \begin{bmatrix} \Gamma & \Phi\Gamma & \Phi^2\Gamma \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{f} & \frac{2}{f} \\ 0 & 0 & \frac{1}{f} \\ 1 & 0 & 0 \end{bmatrix}, \tag{4.41}$$

which has full rank for any frame rate f. Thus, the system is *completely controllable* and the state $\mathbf{e}(n)$ can be regulated to any desirable value. Now recall that the cost function defined in

Section 4.1.4 is

$$I = \sum_{n=0}^{N} \Big\{ \Big(t_b(n) - t_T(n) \Big)^2 + \sigma \Big(\frac{r_c(n+1) - r_c(n)}{\tilde{r}_a} \Big)^2 \Big\} \tag{4.42}$$

$$= \sum_{n=0}^{N} \Big\{ \mathbf{e}(n)^T Q \mathbf{e}(n) + u(n-1)^T R u(n-1) \Big\}, \tag{4.43}$$

where $Q = C^T C$ (with $C = [1 \;\; 0 \;\; 0]$) and $R = \sigma$. Then, the original control problem of tracking the target schedule while smoothing the coding rate fluctuations (i.e., minimizing the cost function I) is converted to a standard regulator problem in the error space. Letting $N \to \infty$, the infinite horizon optimal control problem can be solved by applying the results in [67, Section 3.3] to obtain an optimal regulator in two steps: 1) solving, to get S, the *discrete algebraic Riccati equation*

$$S = \Phi^T \{ S - S\Gamma[\Gamma^T S \Gamma + R]^{-1} \Gamma S \} \Phi + Q, \tag{4.44}$$

and 2) computing the optimal feedback gain

$$G^* = [\Gamma^T S \Gamma + R]^{-1} \Gamma^T S \Phi. \tag{4.45}$$

The existence and uniqueness of S (and in turn of G^*) is guaranteed when Q is nonnegative definite and R is positive definite, which is straightforward to verify in our case.

4.2.3 Frame Rate

In the previous section, we assumed that the frame rate is constant. This assumption is reasonable when streaming a single medium, such as video without audio.[6] However, usually video and audio are streamed together, and their merged coding schedule may have no fixed frame rate. Even if there is a fixed frame rate f, we may wish to operate the controller at a rate lower than f, to reduce the feedback rate, for example.

To address these issues, in practice we use the notion of a *virtual frame rate*. We choose a virtual frame rate f, for example $f = 1$ frame per second (fps); we partition media time into intervals of size $1/f$; and we model all of the (audio and video) frames arriving within each interval as a *virtual frame* whose decoding and playback deadline is the end of the interval.

This approach has several advantages. First, it allows us to design offline a universal feedback gain, which is independent of the actual frame rate of the stream or streams. Second, it allows us to reduce the rate of feedback from the client to the server. And finally, since the interval between virtual frames is typically safely larger than a round trip time (RTT), a one-frame delay in the error space model (as described in the previous section) is sufficient to model the feedback delay.

[6]Variable frame rate video is usually achieved by skipping frames, which we can accommodate by setting $b(n) = 0$.

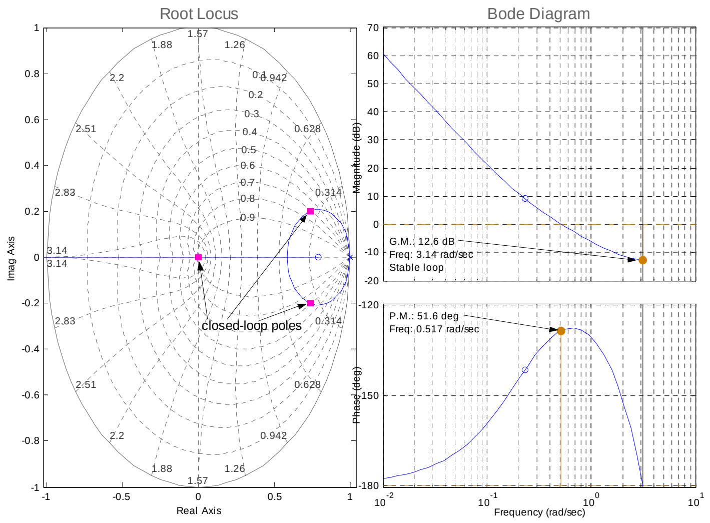


Figure 4.8: Root locus and Bode diagram.

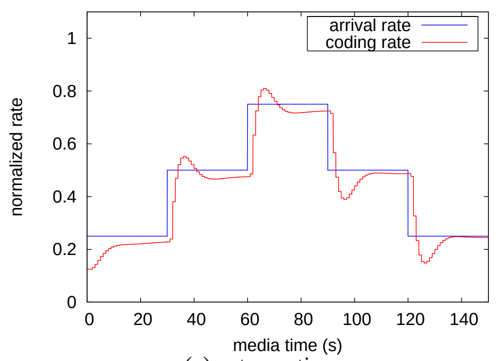
Otherwise we would have to model the feedback delay with approximately RTT/f additional state variables to represent the network delay using a shift register of length RTT/f.

In the sequel we therefore use a virtual frame rate $f = 1$ fps, and we refer to this simply as the frame rate.

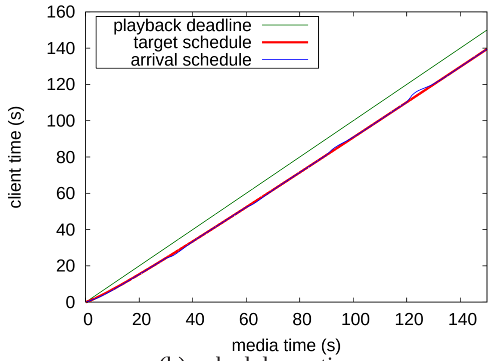
4.2.4 Stability and Robustness

To compute the optimal regulator, it is necessary to choose a value for σ in (4.20) or (4.42)-(4.43). This can be done by following the following four steps: 1) pick a σ value to balance $\mathbf{e}(n)$ and $u(n)$; 2) compute the optimal feedback gain; 3) plot the closed-loop root locus (to check stability) and bode diagram (to check robustness); and 4) perform time domain simulations to verify transient response. Several iterations may be needed to determine a suitable σ value.

Following the above steps in this chapter we select $\sigma = 50$. The corresponding optimal feedback control gain is then $G^* = [0.6307 \ \ -0.5225 \ 0.5225]$, for which the closed-loop system has poles at $0.7387 + 0.1999i$, $0.7387 - 0.1999i$ and 0, which are all inside the unit circle. Therefore, the closed-loop system is asymptotically stable. Figure 4.8 shows the closed-loop root locus and the bode diagram with the optimal feedback. We can again verify the stability of the closed-loop system since all poles are inside the unit circle. Also, the system has a *gain margin* (GM) of 12.60 dB and a *phase margin* (PM) of 51.59 degrees. The GM and PM are usually good indicators of system robustness. In our case, the PM is much larger than 30 degrees, which is often judged as the lowest adequate value [68, Section 6.4]. And this PM is close to 60 degrees, the best PM an optimal controller could achieve if continuous time feedback control was allowed. Therefore, the system achieves good robustness. Finally, Figure 4.9 provides the time response simulation results, which



(a) rate vs. time



(b) schedule vs. time

Figure 4.9: Time response simulation.

show good tracking properties with a fairly stable coding rate (more simulation results are available in [69]).

4.3 Practical Issues with Streaming

4.3.1 Fast Startup

As discussed in previous sections, the startup delay is the length of the period from the time that content first begins to arrive at the client to the time that playback begins. During this period, content accumulates in the receiver buffer to counter packet jitter, retransmission delay, variations in network bandwidth, and variations in instantaneous coding rate. It is conceivable that a longer startup delay would increase the chances of being able to maintain continuous playback in a dynamic network environment. On the other hand, users expect the startup delay to be as small as possible. Thus, it is desirable to investigate techniques that can reduce the startup delay while retaining robustness. One possible approach is to transmit the content at a faster than normal rate at the beginning of streaming. This *bursting* technique will certainly build up the buffer duration in a small amount of time. It, however, puts extra pressure on the network by demanding a higher than normal initial bandwidth, which may not even be available.

In this chapter, we use an alternative *fast startup* technique, which takes advantage of the properties of adaptive media. As discussed in previous sections, by choosing an initial coding rate r_c equal to half the arrival rate r_a (divided if necessary by the playback speed ν), the client buffer duration can grow at two times real time during playback. Growing the client buffer during playback enables the startup delay to be low, because playback can begin while the buffer duration is still low. Beginning playback while the buffer duration is low is not particularly risky over the short term, because the probability of deep congestion occuring in any short interval is low. However, the probability of deep congestion occuring in a long interval is high, so it is important for the

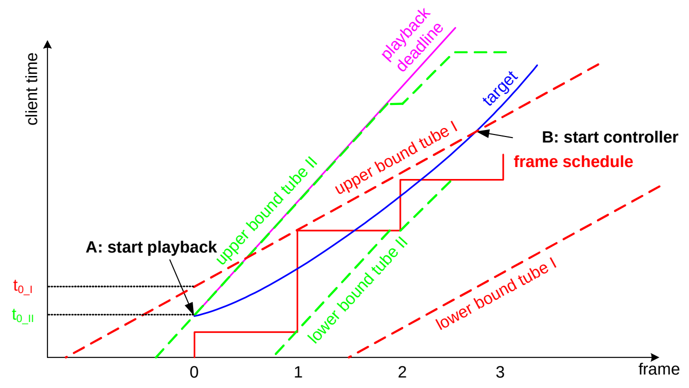


Figure 4.10: Leaky buckets (buffer tubes) for various transmission rates.

buffer duration to be high over the long term. Without the ability to grow the buffer duration during playback, startup would have to be delayed until the buffer duration was sufficiently high to guarantee continuous playback over the long term.

Moreover, if the transmission rate is twice the coding rate, the startup delay can be further reduced by taking advantage of properties of the leaky bucket model [66]. As detailed in Section 4.1.2, the startup delay for a given bit stream is $D = F^d_{\min}(R)/R$ when the stream is transmitted at rate R. This is ordinarily equal to $F^d_{\min}(r_c)/r_c$ when transmitting the stream at its coding rate. However, when transmitting the stream at a rate $r_a > r_c$ ($r_c = 0.5r_a/\nu$), then the startup delay drops to $F^d_{\min}(r_a)/r_a$. Thus the startup delay D decreases both because the numerator decreases and because the denominator increases.

Figure 4.10 illustrates the decrease in the initial decoder buffer fullness $F^d_{\min}(R)$ as R changes from r_c to r_a. In particular, it depicts the coding schedule for a given bit stream, as well as upper and lower bounds, denoted Tube I and Tube II, corresponding to two leaky buckets with leak rates r_c and r_a respectively, both containing the coding schedule. Tube II is smaller than Tube I, since the minimum size $B_{\min}(R)$ of a leaky bucket containing a given stream is decreasing in the leak rate R [66]. Likewise, the initial decoder buffer fullness $F_{\min}(R)$ is decreasing in R [66]. Hence the playback deadline for frame 0 can begin as early as client time $t_{0\_II} = F^d_{\min}(r_a)/r_a$, instead of $t_{0\_I} = F^d_{\min}(r_c)/r_a$. From there, the playback deadline advances at $1/\nu$ seconds of client time per second of media time.

4.3.2 Controller Initialization

As illustrated in Figure 4.10, the target schedule starts at the same time as the playback deadline and grows according to a predefined function. The controller attempts to control the upper bound of Tube I to the target schedule. Initially the upper bound of Tube I is above the target schedule (and is indeed above the playback deadline, though we know that this is safe). Hence, when the playback starts, the controller would try to close the gap by decreasing the coding rate. This, however, would not be desirable because the current coding rate is already lower than the arrival rate to allow the client buffer to grow. Further reduction of the coding rate would not be proper. To avoid this effect, we initialize the controller when the upper bound of Tube I exceeds the target schedule i.e., at point B in Figure 4.10. Point B can be found analytically, but in practice there is no need to explicitly solve for it. The controller can be initialized as soon as the upper bound of Tube I exceeds the target.

4.3.3 Exponential Averaging of the Arrival Rate

From the performance studies of the controller, using the average arrival rate from a low pass filter (instead of the instantaneous arrival rate) helps to reduce coding rate oscillations. This section details our exponential averaging algorithm for the arrival rate.

Let $\tilde{r}_a(k)$ and $r(k)$ be the average arrival rate and the instantaneous arrival rate, respectively, when packet k is received. Note that unlike the controlling operation, the rate averaging operation may be performed after the arrival of every *packet*, rather than after the arrival of every *frame*. Hence we use the discrete packet index k rather than the frame index n. Instead of using the widely adopted exponentially weighted moving average (EWMA)

$$\tilde{r}_a(k) = \beta(k)\tilde{r}_a(k-1) + (1-\beta(k))r_a(k) \tag{4.46}$$

with constant $\beta(k) = \beta$, we perform the exponential averaging more carefully. In our algorithm, the factor $\beta(k)$ is not constant, but varies according to the packets' interarrival gaps. Our algorithm has several advantages over the EWMA algorithm with constant $\beta(k)$. First, the estimate of the average arrival rate $\tilde{r}_a(k)$ goes to zero naturally as the gap since the last packet goes to infinity, rather than being bounded below by $\beta\tilde{r}_a(k-1)$. Second, the estimate of the average arrival rate $\tilde{r}_a(k)$ does not go to infinity as the gap since the last packet goes to zero. This is especially important, since packets often arrive in bursts, causing extremely high instantanous arrival rates. And finally, the estimate of the average arrival rate $\tilde{r}_a(k)$ does not over-weight the initial condition, as if it represented the infinite past. This is especially important in the early stages of estimation.

As in (4.11), we define the instantaneous arrival rate after packet k as

$$r_a(k) = \frac{b(k)}{t_a(k) - t_a(k-1)}, \tag{4.47}$$

where here $b(k)$ denotes the size of packet k and $t_a(k)$ denotes the arrival time of packet k. We extend the discrete time function $r_a(k)$ to the piecewise constant continuous time function $r_a(t)$ by

$$r_a(t) = r_a(k) \quad \text{for all } t \in (t_a(k-1), t_a(k)], \tag{4.48}$$

as illustrated in Figure 4.11. Then we filter the function $r_a(t)$ by the exponential impulse response

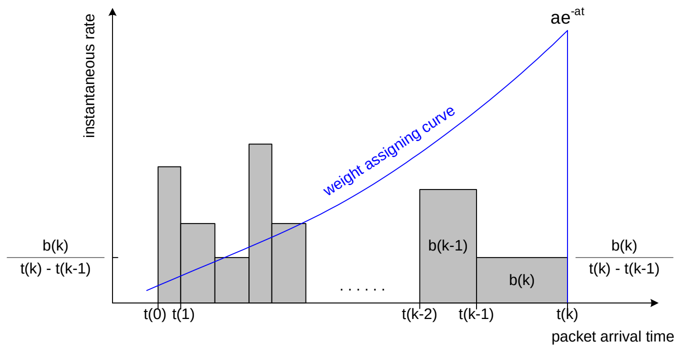


Figure 4.11: Exponential averaging.

$\alpha e^{-\alpha t}$, $t \geq 0$, for some time constant $1/\alpha$:

$$\tilde{r}_a(k) = \frac{\int_{t(0)}^{t(k)} r_a(t')\alpha e^{-\alpha(t(k)-t')}dt'}{\int_{t(0)}^{t(k)} \alpha e^{-\alpha(t(k)-t')}dt'}. \tag{4.49}$$

(Here and in the remainder of this subsection we suppress the subscript from the arrival time $t_a(k)$.) Noting that $\int_t^\infty \alpha e^{-\alpha t'}dt' = e^{-\alpha t}$, the denominator integral can be expressed $1 - e^{-\alpha(t(k)-t(0))}$. Now, we split the range of the numerator integral into ranges $(t(0), t(k-1)]$ and $(t(k-1), t(k)]$ to obtain a recursive expression for $\tilde{r}_a(k)$ in terms of $\tilde{r}_a(k-1)$ and $r_a(k)$,

$$\begin{aligned}
&\tilde{r}_a(k) \\
&= \frac{1-e^{-\alpha[t(k-1)-t(0)]}}{1-e^{-\alpha[t(k)-t(0)]}} e^{-\alpha[t(k)-t(k-1)]}\tilde{r}_a(k-1) \\
&\quad + \frac{1-e^{-\alpha[t(k)-t(k-1)]}}{1-e^{-\alpha[t(k)-t(0)]}} r_a(k) \qquad (4.50) \\
&= \beta(k)\tilde{r}_a(k-1) + (1-\beta(k))r_a(k), \qquad (4.51)
\end{aligned}$$

where

$$\beta(k) = \frac{e^{-\alpha[t(k)-t(k-1)]} - e^{-\alpha[t(k)-t(0)]}}{1-e^{-\alpha[t(k)-t(0)]}}. \tag{4.52}$$

Note that $\beta(k)$ is numerically stable as k goes to infinity. However, as the gap $\delta = t(k) - t(k-1)$ goes to zero, $1 - \beta(k)$ goes to zero while $r_a(k)$ goes to infinity. Their product, however, is well

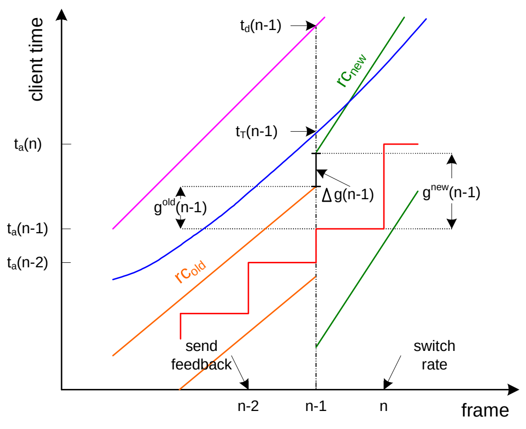


Figure 4.12: Buffer tube change and control target adjustment.

behaved. Indeed,

$$\tilde{r}_a(k) = \frac{1-e^{-\alpha[t(k-1)-t(0)]}}{1-e^{-\alpha[\delta+t(k-1)-t(0)]}} e^{-\alpha\delta}\tilde{r}_a(k-1) + \frac{1-e^{-\alpha\delta}}{1-e^{-\alpha[t(k)-t(0)]}}\frac{b(k)}{\delta} \tag{4.53}$$

$$\rightarrow \tilde{r}_a(k-1) + \frac{\alpha b(k)}{1-e^{-\alpha[t(k)-t(0)]}} \tag{4.54}$$

as $\delta \rightarrow 0$, using l'Hôpital's rule. Thus (4.54) is the update rule in the case when $t(k) = t(k-1)$.

4.3.4 Choosing a Stream Given a Coding Rate

When the client requests a coding rate $r_c(n)$, the server complies by choosing a stream (or substream of a scalable stream) having coding rate $\hat{r}_c(n)$ approximately equal to $r_c(n)$. There are several reasons that $\hat{r}_c(n)$ may differ from $r_c(n)$. The first reason is that there are only a finite number of streams (or substreams) in the media file, even if fine grain scalable coding is used. Thus there may be no stream in the media file with average coding rate exactly equal to $r_c(n)$. The second reason is that, even if there is a stream in the media file with average coding rate exactly equal to $r_c(n)$, the buffer tube for the stream may be too large to allow switching to the stream without risk of client buffer underflow. In fact, whenever the stream switches, there is generally a discontinuity in the upper bound, which may be either positive or negative. A positive shift in the upper bound is illustrated in Figure 4.12, which, if large, could cause the client buffer to underflow either immediately or eventually.

Thus the server must choose a stream that causes the upper bound to shift up no more than some amount $\Delta^{\max}g(n-1)$ supplied to it by the client. The client supplies $\Delta^{\max}g(n-1)$ to the server in its feedback along with $r_c(n)$, shortly after client time $t_a(n-2)$ (after frame $n-1$ has already begun streaming). Upon receiving the feedback, the server selects a stream with coding rate $\hat{r}_c(n)$ as high as possible such that $\hat{r}_c(n) \leq r_c(n)$ and, if $\hat{r}_c(n) > \hat{r}_c(n-1)$ (i.e., if it is a switch up in rate), then $g^{new}(n-1) - g^{old}(n-1) \leq \Delta^{\max} g(n-1)$, where $g^{new}(n-1)$ and $g^{old}(n-1)$ are illustrated in Figure 4.12. The constraint given by $\Delta^{\max} g(n-1)$ is not applied if it is a switch down in rate.

The client chooses $\Delta^{\max} g(n-1)$ to limit (its prediction of) what the upper bound would be at time $t_a(n-1)$ if the new coding rate were in effect, namely,

$$\begin{aligned} & t_b^{new}(n-1) \\ \approx \quad & t_b(n-2) + \frac{\hat{r}_c(n-1)}{f\tilde{r}_a} + \frac{\Delta g(n-1)}{\tilde{r}_a} && (4.55) \\ \leq \quad & t_T(n-1) + p[t_d(n-1) - t_T(n-1)]. && (4.56) \end{aligned}$$

That is, the client chooses $\Delta^{\max} g(n-1)$ to limit $t_b^{new}(n-1)$ so that it would be no more than fraction p of the way from the target $t_T(n-1)$ to the playback deadline $t_d(n-1)$. In our experiments, we choose $p = 1/3$.

4.3.5 Control Target Adjustment

When a frame with a new average coding rate $\hat{r}_c(n)$ arrives at the client at time $t_a(n)$, there is a shift in the upper bound. Real scalable stream data (cf. Figure 4.14) shows that this shift can be on the order of seconds and hence, rather than being negligible, can be confusing to the controller. If the shift is upward, for example, the controller will immediately try to reduce the coding rate $r_c(n+2)$. If the shift is downward, on the other hand, the controller will immediately try to increase the coding rate $r_c(n+2)$. Either way is probably not good; the intention is that $\hat{r}_c(n)$ will be maintained unless there is a disturbance in the arrival rate. Our solution is to introduce a simultaneous shift in the control target schedule equal to $\Delta g(n-1)/\tilde{r}_a$, where $\Delta g(n-1) = g^{new}(n-1) - g^{old}(n-1)$ is the actual shift in the upper bound (in bits) at frame $n-1$ computed at the server, as illustrated in Figure 4.12. The server can send this value to the client along with frame n. If there is no stream change, this value is simply zero.

If the control target schedule is adjusted whenever the coding rate changes, it will no longer follow the designed target schedule. We refer to the adjusted target schedule as the *control target* schedule to distinguish it from the *designed target* schedule (or simply the *target schedule*).

The control target schedule, of course, must have a tendency to approach the designed target schedule. The basic idea is to decrease the slope of the control target schedule when it is above the designed target schedule and to increase the slope when it is below.

For the logarithmic target schedule $t_T = t_d - \frac{b}{a}\ln(at_d + 1)$ (where $t_d = t_{d0} + (\tau_d - \tau_{d0})/\nu$), according to (4.24) the slope at media time τ_d is

$$s = \frac{dt_T}{d\tau_d} = \frac{1}{\nu} - \frac{b}{a(\tau_d - \tau_{d0}) + \nu}. \tag{4.57}$$

If we define d as the distance between the playback deadline and the target schedule, namely

$$d = \frac{b}{a}\ln\left(a\left(\frac{\tau_d - \tau_{d0}}{\nu}\right) + 1\right), \tag{4.58}$$

then the slope may be expressed as a function of d,

$$s = \frac{1}{\nu} - \frac{b}{\nu e^{(a/b)d}}. \tag{4.59}$$

Hence whenever d is the distance between the playback deadline and the control target, we set the slope of the control target to s in (4.59). Specifically, if $t_{\hat{T}}(n)$ is the control target at frame n after the shift, then we reset $t_{\hat{T}}(n-1)$ to be $T_{\hat{T}}(n) - s/f$. We then use $t_{\hat{T}}(n)$ and $t_{\hat{T}}(n-1)$ in place of $t_T(n)$ and $t_T(n-1)$ to compute the error vector $\mathbf{e}(n)$ in (4.37). The resulting error vector is then used to compute the ideal coding rate in (4.40).

For the two-piece linear target schedule, the slope is easy to compute by using a predefined time period over which the control target schedule is expected to return to the target schedule. The slope of the control target schedule can then be computed from the distance d and the period. We set the period to 50 seconds in our experiments.

4.4 Implementation Details

This section highlights implementation details on both the sender and the receiver side.

4.4.1 Generation of Virtual Streams

In our implementation, a fine grained scalable (FGS) stream comprises a set of data units, each tagged by a Lagrange multiplier λ representing the per-bit decrease in distortion if the data unit is received by the client. If the λ for the data unit is above a threshold, then the data unit is included in a virtual stream corresponding to that threshold. Each threshold corresponds to an overall number of bits and hence an average coding rate for the virtual stream. In our experiments, we generate $N = 50$ virtual streams. A threshold is chosen for each stream such that the resulting streams have coding rates that are uniformly spaced in the log domain between lower and upper bounds.

During streaming, when the server reads a data unit from the media file, it includes the data unit in the virtual stream currently being transmitted if its Lagrange multiplier λ is above the threshold for the stream.

4.4.2 Leaky Bucket Computations at the Sender

For each virtual stream, leaky bucket parameters $(R, B_{\min}(R), F^d_{\min}(R))$ are precomputed off line for $R = R_{avg}$ and $R = R_{\max}$, where $R_{avg} = r_c$ is the average coding rate of the stream, and $R_{max} = 2r_c$. These leaky bucket parameters are sent to the client in a preamble.

In addition, during streaming the server performs on-line leaky bucket simulations for each stream. Specifically, whenever the server reads a data unit from the media file, it determines the virtual streams to which the data unit belongs, using the Lagrange multiplier of the data unit and the list of thresholds for each stream. The sender then updates, for the determined streams, the states of those leaky buckets having leak rates equal to an average coding rate R_{avg}, using (4.2) and (4.3). Once all the data units in a frame are read from the media file, the sender computes $g(n) = B_{\min}(R_{avg}) - B^e(n)$ for each of the virtual streams. On a stream switch (i.e., $\hat{r}_c(n) \neq \hat{r}_c(n-1)$), the gap $g^{new}(n)$ for the new stream is transmitted to the client along with $\Delta g(n-1) = g^{new}(n-1) - g^{old}(n-1)$ as described below. It is easy to see that the cost of updating the leaky bucket states is quite low. However, it is also possible to precompute these values and store them with each data unit in the media file.

4.4.3 Initial Coding Rate Selection

At the beginning of a streaming session, the sender needs to have some knowledge of the available network bandwidth so that it can choose an initial coding rate (usually half of the bandwidth). The bandwidth estimate can be drawn from proactive measurements, using approaches such as packet pair [70], path chirp [71], etc., or reactive approximations based on history values. The exact form of the initial bandwidth estimation is beyond the scope of this work.

4.4.4 Coding Rate Switching

The rate control feedback from the client contains the frame number at which feedback is generated (e.g., $n-2$ in the previous section) and the maximum allowable shift of the upper bound in bits (e.g., $\Delta^{max} g(n-1)$ in the previous section). If the sender finds a suitable coding rate and makes a switch at frame n, it will transmit three values to the client along with the frame: the new coding rate $\hat{r}_c^{new}(n)$, the current gap to the upper bound $g^{new}(n)$, and the shift $\Delta g(n-1) = g^{new}(n-1) - g^{old}(n-1)$. With this information, the client can properly adjust its control target schedule as well as its upper bound. Note that coding rate switching always happens at the beginning of a new frame, never inside a frame.

4.4.5 Optimal Rate Control at the Client

Whenever a new coding rate starts, the client receives the value $g(n)$ along with the new frame. The values of $g(n)$ for successive frames can be then inferred by the client itself based on the coding

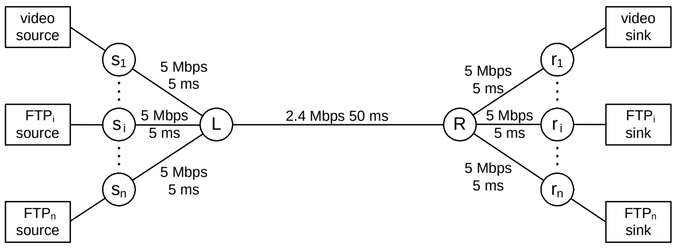


Figure 4.13: ns-2 Simulation network setup.

rate $\hat{r}_c(n)$ and the frame size $b(n)$. The client records the arrival frame time $t_a(n)$, calculates the buffer tube upper bound $t_b(n)$ and then computes the deviation $e(n)$. If there is a coding rate switch, it will also compute the buffer tube shift and adjust the control target schedule accordingly. Then $e(n)$ is feed to the optimal rate controller, which then outputs a desired new coding rate. The latest new coding rate is fed back to the sender whenever there is a feedback opportunity, which could be generated at regular intervals or on-demand.

4.5 Performance Evaluation

In this section, we evaluate the performance of the optimal rate control system when streaming a fine grained scalable (FGS) video stream.

The test video is a 3-minute clip, which we obtain by six repetitions of the concatenation of the three MPEG standard test sequences *Akiyo*, *Stefan*, and *Foreman* in that order. The test video is downsampled to QCIF, 10 fps, for a total of 1800 underlying QCIF frames.[7] The test video is coded using a variant of MPEG-4 FGS [57], with a 10-second I-frame distance and no B frames. Using rate-distortion optimization, from the FGS stream we extract 50 substreams whose average coding rates are uniformly spaced in the log domain between log 50 kbps and log 1000 Kbps.

Using the popular network simulator ns-2 [72], we set up a simple network environment as shown in Figure 4.13. Video traffic is streamed from node s_1 to node r_1 while competing FTP cross traffic (FTP$_i$) is transmitted node s_i to node r_i ($2 \leq i \leq n$). By adjusting the number of FTP flows and their beginning/ending times, we can create both constant and variable available bandwidth scenarios for the streaming session, as specified in Table 4.1. Experiments are carried out using both TCP and TFRC [61] as alternative transport layer protocols. Note the TFRC protocol yields similar results as the TCP protocol, which are thus not reported here (refer to [69] for more details).

4.5.1 Startup Delay

Figure 4.14 shows the startup delay as a function of the transmission/arrival rate r_a, for two streams,

[7]The original Akiyo and Stefan test sequences are 300 frames, which we downsample to 100 frames each. The original Stefan test sequence is 400 frames, from which we extract the first 300 frames before downsampling to 100 frames.

Table 4.1: Bandwidth Available to the Streaming Session

	client time	# of FTPs	fair share BW
Constant Bandwidth	0–180 s	5	400 Kbps
Variable Bandwidth	0–30 s	2	800 Kbps
	30–60 s	5	400 Kbps
	60–90 s	11	200 Kbps
	90–130 s	5	400 Kbps
	130–180 s	2	800 Kbps

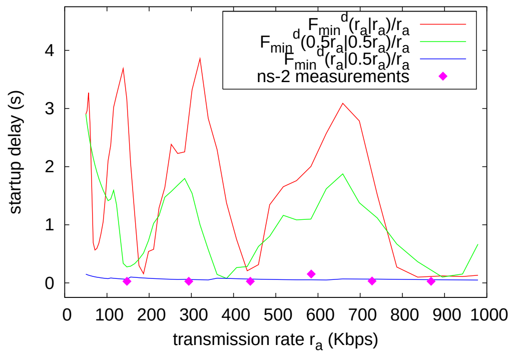
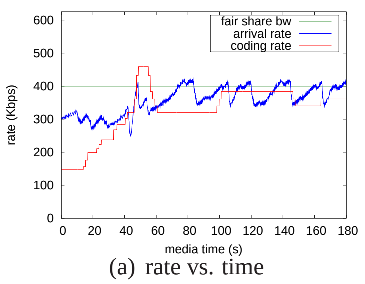


Figure 4.14: Startup delay vs. transmission rate.

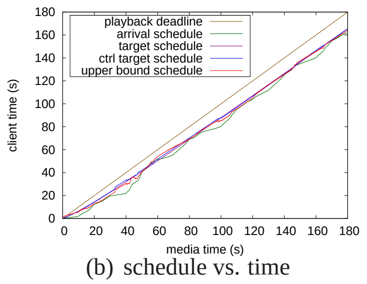
one at average coding rate $r_c = r_a$, and another at $r_c = 0.5r_a$. Specifically, for the virtual stream with average coding rate r_c, let $F^d_{\min}(R|r_c)$ denote the minimum initial decoder buffer size computed for a leaky bucket with leak rate R. (We know that for a fixed r_c, this function decreases in R). The top curve in the figure shows the startup delay $F^d_{\min}(r_a|r_a)/r_a$, when the coding rate is chosen to match the transmission rate. The middle curve shows the startup delay $F^d_{\min}(0.5r_a|0.5r_a)/r_a$, when the coding rate is chosen to be half of the transmission rate, but the initial decoder buffer fullness is based on the coding rate. And the bottom curve shows the startup delay $F^d_{\min}(r_a|0.5r_a)/r_a$, when the coding rate is chosen to be half of the transmission rate, and the initial decoder buffer fullness is based on the transmission rate, thus further reducing the startup delay. The three curves in the figure are calculated using leaky bucket simulations with the virtual streams' coding schedules, but we notice that the bottom curve matches nicely with experimental results from our ns-2 simulations at rates at 150 Kbps, 300 Kbps, 450 Kbps, 600 Kbps, 750 Kbps and 900 Kbps, all of which have delay much lower than 1 second.

4.5.2 Constant vs. Variable Bandwidth

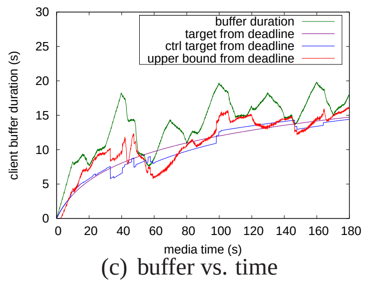
Figures 4.15 and 4.16 show results using TCP as the transport protocol, under constant and variable bandwidth conditions, respectively. In either case, in the startup phase, the coding rate is about half of the arrival rate, which allows fast startup and helps to build the client buffer quickly. The



(a) rate vs. time

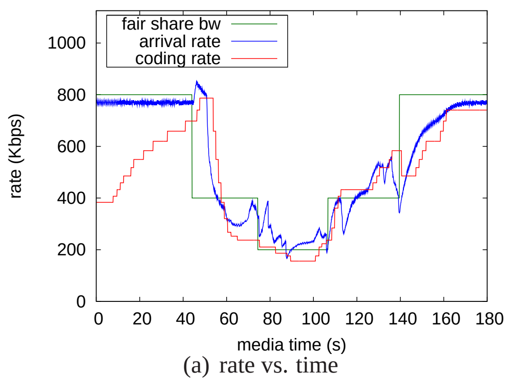


(b) schedule vs. time

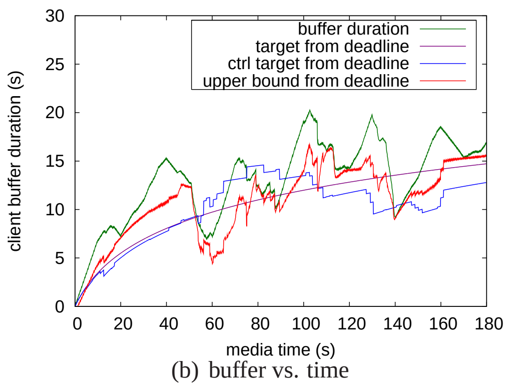


(c) buffer vs. time

Figure 4.15: Constant bandwidth over TCP.



(a) rate vs. time



(b) buffer vs. time

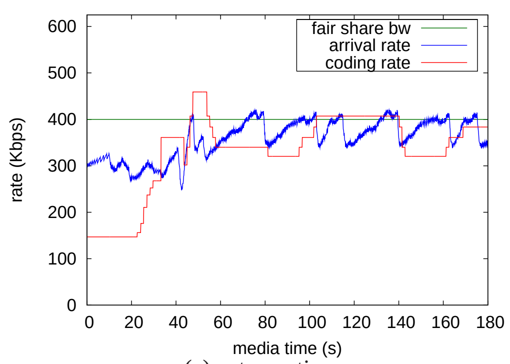
Figure 4.16: Variable bandwidth over TCP.

coding rate catches up smoothly with the arrival rate and tracks it smoothly despite fluctuations in the available bandwith. As the result of coding rate adjustments, the client buffer is well maintained around the logarithmic target schedule, ensuring that no frame misses its playback deadline.

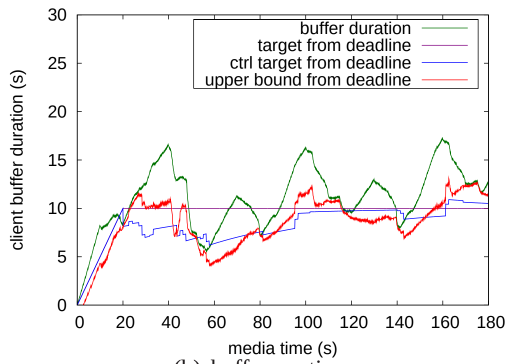
Figure 4.15(c) presents essentially the same information as Figure 4.15(b), but plots the *difference* between the playback deadline and 1) the arrival schedule, 2) the buffer tube upper bound schedule, 3) the control target schedule, and 4) the logarithmic target schedule, respectively. Note that the gap between the playback deadline and the arrival schedule is the client buffer duration. In the remainder of this chapter, we present all schedules using this format.

4.5.3 Two-piece linear vs. logarithmic target schedule.

Figures 4.17 and 4.18 show results using TCP as the transport protocol with the two-piece linear target schedule. Compared to the logarithmic target schedule, the two-piece linear target schedule holds the initial lower coding rate for a longer period (thus sacrificing more quality) in the startup phase, so that the client buffer can build up more quickly. After the startup phase, there is no further need to sacrifice quality to maintain the client buffer level. In contrast, with the logarithmic target

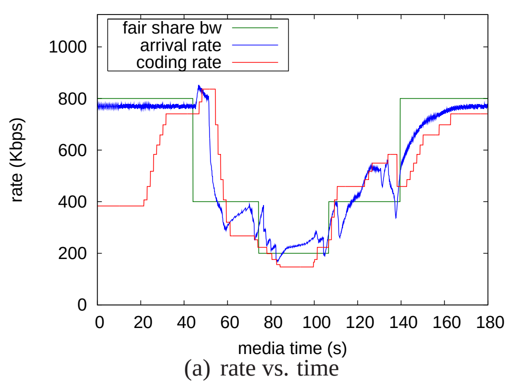


(a) rate vs. time

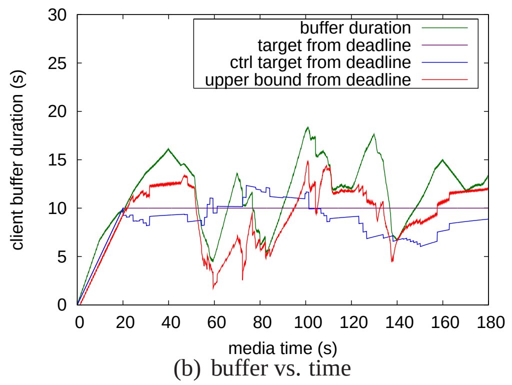


(b) buffer vs. time

Figure 4.17: Constant bandwidth over TCP with the two-piece linear target schedule.



(a) rate vs. time



(b) buffer vs. time

Figure 4.18: Variable bandwidth over TCP with the two-piece linear target schedule.

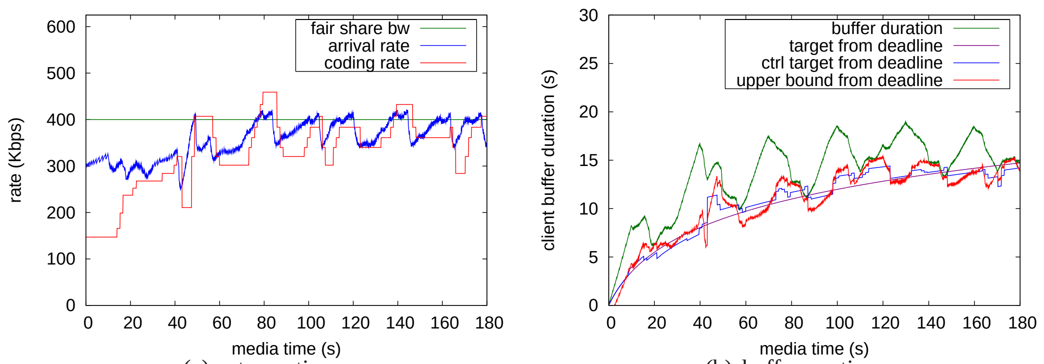
schedule, there is some sacrifice in quality over the entire streaming session, although the sacrifice diminishes gradually as the slope of the schedule approaches a constant.

It is clear that both target schedules work well under either constant bandwidth or variable bandwidth situations. The choice, which reflects a balance between quality and buffer level in the startup phase as well as asymptotically, can be deferred to particular applications.

4.5.4 Controller Performance Tuning

Tuning σ

The performance figures show significant deviation of the buffer tube upper bound from the control target, which is especially obvious in the variable bandwidth case. It is clear from our controller design rationale that we can reduce this deviation by decreasing the σ value. A smaller value of σ value implies a relative larger penalty on the deviation term in the cost function and thus forces the upper bound to track the target more closely. This, however, happens at the cost of sacrificing coding



(a) rate vs. time (b) buffer vs. time

Figure 4.19: Constant bandwidth over TCP, $\sigma = 500$.
The upper bound tracks the control target more closely, while the coding rate is less smooth, compared to Figure 4.15.

rate smoothness, since the corresponding term in the cost function will be weighted less. Figure 4.19 shows simulation results with $\sigma = 500$ under the same network conditions as in Figure 4.15. It is clear that while the buffer tube upper bound deviates only slightly from the control target, the coding rate has undesirable oscillations.

On the other hand, a large σ value will certainly yield smoother coding rates, but might also incur client buffer underflow since the buffer tube upper bound is allowed to deviate significantly away from the control target. Therefore, a good choice of σ should take into account this trade-off. In our implementation, we choose $\sigma = 4000$ when the coding rate switches up and $\sigma = 2000$ when it switches down. Note that we allow a slightly more aggressive strategy in the latter case to further reduce the chance of client buffer underflow. It is straightforward to verify that this choice of σ maintains a stable closed-loop and good gain/phase margins; this is not repeated here.

Smoothing $e(k)$

The frame arrival time t_a, which is used to compute the controller input, is the client time at which a frame is completely received. This time could increase significantly if part of the frame arrives in retransmitted packets. When the controller is fed with $e(n)$, which is a deviation computed from the arrival time, the controller may misinterpret the increase and may generate oscillatory output over time. Note that this variation in arrival time is different from the variation in transmission rate and is not specifically addressed in our mathematical model. Thus, we need an additional mechanism to deal with it.

A straightforward approach is to apply our exponential averaging method on $e(k)$, which will certainly smooth out spiky values of the deviation and let the controller react upon the long time trend. Let $\tilde{e}(n)$ be a smoothed sequence input to the controller instead of $e(n)$, specifically

$$\tilde{e}(n) = \frac{e^{-\alpha} - e^{-\alpha n}}{1 - e^{-\alpha n}} \tilde{e}(n-1) + \frac{1 - e^{-\alpha}}{1 - e^{-\alpha n}} e(n). \tag{4.60}$$

We choose $\alpha = 1/f$ (the frame rate) to focus on history values in the last second, which will also allow $\tilde{e}(n)$ to follow the trend promptly when a significant change in bandwidth occurs. All results reported in this section use this mechanism.

4.5.5 Comparison with Benchmark Algorithm

As a benchmark, we compare our buffer management algorithm to the windowing algorithm in [60] (which is part of the rate-distortion optimized sender-driven streaming algorithm therein). In the benchmark algorithm, the server maintains a sending window, which contains the range of frames that are potentially in the client buffer. The sending window slides forward to mimic the playback (consumption) of frames at the client. At each transmission opportunity, the sender selects from the window a data unit that most decreases the distortion at the client (per transmitted bit). The sliding window looks ahead based on a logarithmic function (similar to the logarithmic target schedule herein), which starts small and grows slowly over time. Hence, the client can have low startup delay and can gradually increase its buffer over time.

Although conceptually simple and sound, the benchmark algorithm has two disadvantages. First, it does not send out data units in the order in which they appear in the media file (i.e., decoding order). This demands resources (e.g., caching large segments of data) that may be incompatible with high performance streaming. Second and more importantly, until the window becomes large enough to accommodate constant quality streaming (about 25 seconds for typical movies), the benchmark algorithm demands, essentially, constant bit rate streaming. This is because the duration of the client buffer is determined by the logarithmic function. In contrast, in our algorithm, only a portion of the client buffer duration (namely the safety zone between the target and the playback deadline) is determined by the logarithmic function. The remainder of the client buffer duration is determined by the leaky bucket state when processing the video content.

Figure 4.20 shows the buffer tube containing the coding schedule for a video sequence consisting of *Akiyo*, *Stefan* and *Foreman* (10 s each) at an average coding rate of 500 Kbps. Note that *Akiyo* requires relatively few bits per second of media time, and *Stefan* requires relatively more bits per second of media time, to achieve quality similar to *Foreman*. Thus if the three subsequences are all coded with roughly the same number of bits per second of media time, *Akiyo* will have higher quality, and *Stefan* will have lower quality, relative to *Foreman*.

Figures 4.21 shows the PSNR results after streaming with a constant bandwidth of 400 Kbps over TCP. Our optimal control algorithm with either target schedule is much smoother in terms of PSNR compared to the benchmark algorithm. Note that even with optimal control, the PSNR value shows a repetitive pattern over the entire session, instead of a constant value. This happens because

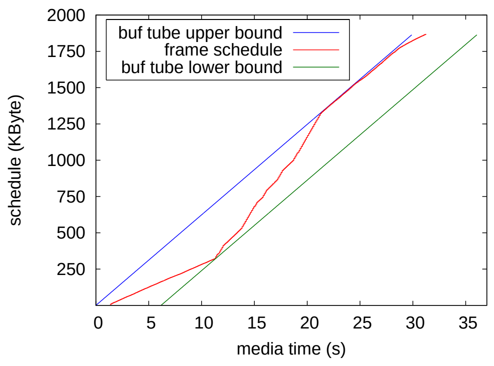
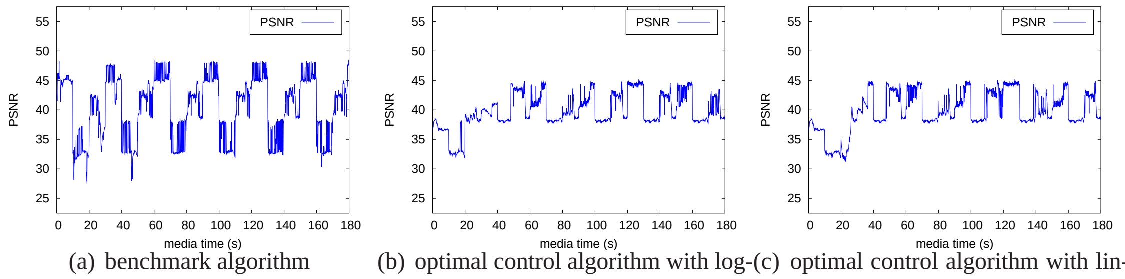


Figure 4.20: Coding schedule of a mixed video sequence (Akiyo, Stefan and Foreman) at an average coding rate of 500 Kbps.



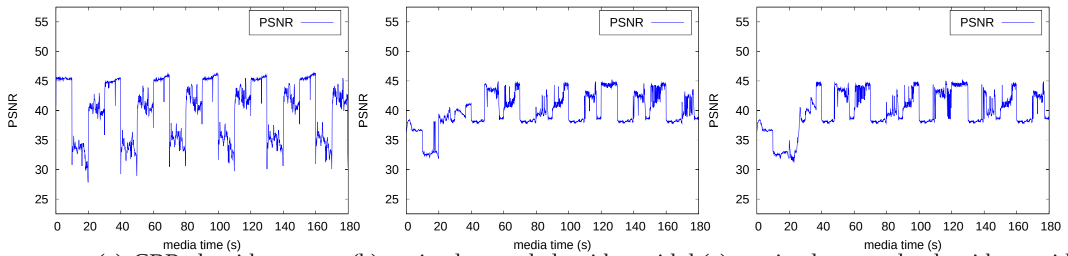
(a) benchmark algorithm (b) optimal control algorithm with logarithmic target (c) optimal control algorithm with linear target

Figure 4.21: PSNR with constant bandwidth (400 Kbps) over TCP.

the scalable codec we use in the experiments is a bit plane codec. There could be one bit plane difference (about 6 dB in PSNR) between frames of the same coding rate.

4.5.6 Comparison with Constant Bit Rate Algorithm

The CBR algorithm is a simple rate control mechanism that takes advantage of the ability of to truncate an FGS encoded frame at any point. Thus it is possible to control the rate by sending the media data in real time, but truncating each frame to match to available transmission rate. If the transmission rate is constant, this yields a constant number of bits per frame. The algorithm is simple and effective in the sense that it successfully avoids any risk of rebuffering by matching the instantaneous coding rate to the transmission rate. However, without taking into account the variable bit rate nature of constant quality coding, this algorithm results in high quality for smooth content (which is easy to encode), and low quality for high-action content (which is hard to encode). The quality oscillation is significant over constant bandwidth channels as shown in Figure 4.22. The experimental settings for these figures are the same as for Figures 4.21.



(a) CBR algorithm

(b) optimal control algorithm with logarithmic target

(c) optimal control algorithm with straight target

Figure 4.22: PSNR with constant bandwidth (400 Kbps) over TCP.

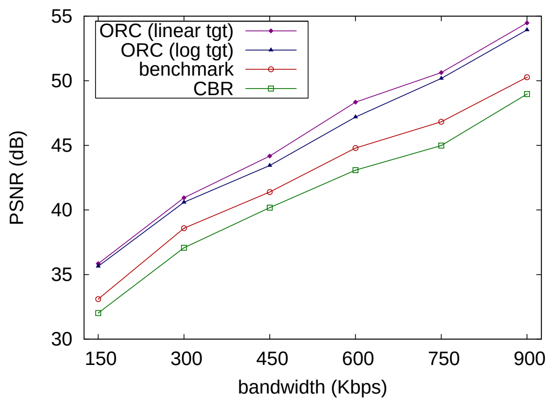


Figure 4.23: Rate-Distortion comparison.

4.5.7 Rate-Distortion Comparison

To compare the rate-distortion performance of all aforementioned algorithms, experiments over a wide range of available bandwidth (150-900 Kbps) are carried out. Each experiment sets a constant available bandwidth for the streaming session and TCP protocol is used for all experiments. The average distortion in terms of PSNR over each session is computed on the client side and plotted in Figure 4.23. Note that frames over the first 40 s (media time) are excluded from the average distortion computation. These frames correspond roughly to the time period (about 30 s in client time) when the client buffer is built up by streaming at lower coding rates than the available bandwidth. The quality sacrifice during the initial period will be easily amortized over streaming sessions of reasonable length and it is appropriate not to be considered in this rate-distortion comparison (where each session is just 3 minutes long).

From the reported results, we can see that the optimal coding rate control algorithm has better rate-distortion performance than the benchmark and the CBR algorithms. Over the wide range of bandwidth, the optimal coding rate control algorithm yields about 2-3 dB PSNR gain over the benchmark algorithm. We can also see that, in general, the linear target schedule has slightly better performance than the logarithmic target schedule. This is understandable since the quality sacrifice happens only during the initial period for the linear target schedule, while it spreads over the entire streaming session for the logarithmic target schedule. The reason that the CBR algorithm has worse performance than the benchmark algorithm is also clear. The CBR algorithm can be regarded as an extreme case of the benchmark algorithm, where the sending window maintained on the server side contains only one frame data at any time. Hence, the limited ability of the benchmark algorithm to smooth quality is further reduced in this case.

Chapter 5

Optimal Coding Rate Control for Multi Bit Rate Streaming

Multiple bit rate (MBR) streaming is a network adaptive technique that is widely used in commercial streaming media systems (e.g. Windows Media 9 Series [51]). In MBR streaming, in contrast to scalable streaming, the content is encoded into several (at most 5–7) independent streams at different coding rates. Often, each stream is optimized for a common type of network connection (e.g., dial-up, DSL, cable). During an MBR streaming session, the proper coding rate is dynamically selected based on the available network bandwidth, with the goal of achieving the maximum possible quality under the condition of uninterrupted playback. It is easy to see that MBR streaming is analogous to scalable streaming. Indeed MBR streaming can be viewed as a special case of scalable streaming with a limited number of coding rates available. Hence, our optimal control approach should be applicable to this case.

There are, however, several differences that complicate MBR streaming, which need to be carefully addressed. First, as just mentioned, in MBR streaming there are only a limited number of coding rates available. This coarse quantization of the desired coding rate introduces a significant nonlinearity into the closed loop system. In fact, the large gaps between the available coding rates introduce oscillations. For example, if two neighboring coding rates straddle a constant arrival rate, the controller will oscillate between the two coding rates in an attempt to keep the client buffer at a target level.

Second, in MBR streaming the coding rate cannot be switched at an arbitrary time. In fact, before the server can switch to a new stream, it must wait for the next clean point (e.g., *I* frame) in the new stream, which could be five or ten seconds away. Thus, the old coding rate may continue for quite a while before it changes to the new coding rate. From the controller's perspective, this long random extra delay tends to destabilize the closed-loop system.

Third and finally, in MBR streaming, server performance issues are critical. The commercial-grade streaming media systems that use MBR streaming do so because of the minimal computational load that it imposes on the server compared to scalable streaming. Thus, for MBR streaming it is important to keep almost all computation and state maintenance on the client side. In particular, the server will not be able to update the leaky bucket information for each stream, as we have proposed in the previous chapter. Instead, the client must use some mechanism for estimating and maintaining this information.

5.1 Conservative Up-Switching

In this section we discuss a technique to help stabilize the control system and reduce steady state oscillations to a period of at least a minute. With this technique, rapid down-switching is permitted. In fact, we reduce the value of σ from 4000(2000) to 1000(500), changing the balance between responsiveness and smoothness of the coding rate in favor of rapid switching response. However, only conservative up-switching is permitted. Conservative up-switching ensures that spurious changes in coding rate do not occur, and that oscillations in the coding rate have a low frequency. In particular, conservative up-switching reduces the oscillations between two adjacent but widely spaced MBR coding rates, one above the arrival rate and one below the arrival rate.

The method behind conservative up-switching is to establish a conservative limit on how high the coding rate can be raised above the arrival rate. If the current coding rate is below the arrival rate, and the client buffer duration begins to increase above its target level, then the coding rate can be switched up to a new coding rate above the arrival rate only if the new coding rate is below the conservative limit. When the client buffer duration begins at the target level, the conservative limit is equal to the arrival rate. However, as the client buffer duration increases, the conservative limit increases as well. Thus, if the current coding rate is below the arrival rate, and the next higher coding rate is above the arrival rate, then it will be possible to switch up to the next higher coding rate only after the client buffer duration has increased sufficiently so that the conservative limit rises above the higher coding rate. Once the coding rate is switched up to the higher coding rate, the client buffer begins to drain since the coding rate is then above the arrival rate. Eventually, when the buffer drains back below its target level, the controller will rapidly switch the coding rate back down to the coding rate below the arrival rate.

Given the current client buffer duration, the conservative limit is set to a value such that if the coding rate is switched up to a new coding rate at this value, the client buffer would take at least Δt seconds of client time to drain back to the target level. Thus, the mechanism ensures that the period of oscillation will be at least Δt seconds. In our experiments, we set Δt to be 60 seconds.

Figure 5.1 shows how we compute the conservative limit. Let $\Delta\tau_1$ be the client buffer duration (in media time) at the moment that the coding rate is switched up from r_c^{old} to r_c^{new}. Thus $\Delta\tau_1$ is the number of seconds of content that will be consumed at the old coding rate r_c^{old} before

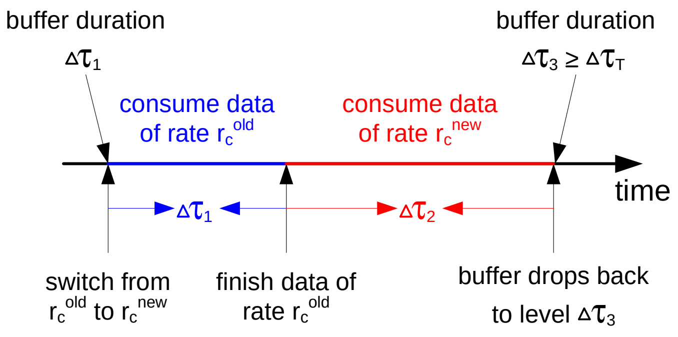

buffer duration
Δτ1
buffer duration
Δτ3 ≥ ΔτT
consume data of rate rc old
consume data of rate rc new
time
Δτ1
Δτ2
switch from rc old to rc new
finish data of rate rc old
buffer drops back to level Δτ3

Figure 5.1: Conservative rate up-switching.

content at the new coding rate begins to be consumed. (For simplicity we assume that all of the content in the client buffer at the time of the switch is coded at rate r_c^{old}.) Let $\Delta\tau_2$ be the number of seconds of content that is consumed at the new coding rate r_c^{new} before the client buffer duration drops to some level $\Delta\tau_3$ seconds (in media time), greater than the target level $\Delta\tau_T$. The duration of this phase is determined such that the total time since the switch is exactly $\Delta t = (\Delta\tau_1 + \Delta\tau_2)/\nu$ seconds (in client time). Now, the number of bits that arrive in this time is $r_a\Delta t = r_c^{new}(\Delta\tau_2 + \Delta\tau_3) \geq r_c^{new}(\Delta\tau_2 + \Delta\tau_T) = r_c^{new}(\nu\Delta t - \Delta\tau_1 + \Delta\tau_T)$, or

$$r_c^{new} \leq \frac{r_a\Delta t}{\nu\Delta t - \Delta\tau_1 + \nu\Delta t_T}, \tag{5.1}$$

where Δt_T is the target buffer duration in client time. The parameter Δt can be tuned to yield the desired behavior. A large Δt means that up-switching will be more conservative, while a smaller Δt means that up-switching will be more prompt. In our implementation, Δt is set to 60 seconds while the target Δt_T is typically about 10 seconds.

5.2 Buffer Tube Upper Bound Estimation

In Section 4.4.4 we specified that the server sends three values to the client at the beginning of each change in coding rate: the new coding rate $\hat{r}_c^{new}$, the current gap to the upper bound $g^{new}(n)$, and the control target shift $\Delta g(n-1) = g^{new}(n-1) - g^{old}(n-1)$. The server computes the latter two values by running a leaky bucket simulator for each coding rate. The client continues to update $g(n)$ for the new coding rate by running its own leaky bucket simulator for the new coding rate. That is, beginning with the initial condition $F^e(n) = B - b(n) - g^{new}(n)$, for each successive frame the client computes

$$\begin{aligned} B^e(n) &= F^e(n) + b(n) & (5.2)\\ F^e(n+1) &= \max\{0, B^e(n) - \hat{r}_c/f(n)\}, & (5.3)\end{aligned}$$

where

$$f(n) = \frac{1}{\tau(n+1) - \tau(n)} \tag{5.4}$$

is the instantaneous frame rate, as in (4.2), (4.3), and (4.4). From this, the client can compute

$$g(n) = B - B^e(n) \tag{5.5}$$

for each frame.

However, if the server is unable to simulate the leaky buckets and cannot send $g^{new}(n)$ to the client, then the client must estimate this information for itself. In this case we recommend that the client estimates $g^{new}(n)$ as an upper bound such as $\hat{g}^{new}(n) = B - b(n) \geq g^{new}(n)$. Then, beginning with initial condition $\hat{F}^e(n) = B - b(n) - \hat{g}^{new}(n)$ (which equals 0 in this case), for each successive frame the client computes

$$\begin{aligned} \hat{B}^e(n) &= \hat{F}^e(n) + b(n) & (5.6) \\ \hat{F}^e(n+1) &= \max\{0, \hat{B}^e(n) - \hat{r}_c/f(n)\}, & (5.7) \end{aligned}$$

as well as

$$\hat{g}(n) = B - \hat{B}^e(n). \tag{5.8}$$

It is easy to see by induction that $\hat{F}^e(n) \leq F^e(n)$, $\hat{B}^e(n) \leq B^e(n)$, and $\hat{g}(n) \geq g(n)$. Moreover, these bounds each become tighter by $\delta(n) = \hat{r}_c/f(n) - B^e(n)$ whenever $\delta(n) > 0$, i.e., whenever $F^e(n+1)$ is clipped to 0 in (5.7). In fact, given enough time they may eventually become tight.

Note that whenever the bounds tighten by $\delta(n) > 0$, the control target must be shifted by $\Delta g(n)/\tilde{r}_a$, where $\Delta g(n) = -\delta(n)$. Furthermore, whenever n is the first frame of a new coding rate, the control target must be shifted by $\Delta g(n)/\tilde{r}_a$, where $\Delta g(n) = \hat{g}^{new}(n) - \hat{g}^{old}(n)$. Here, $\hat{g}^{old}(n)$ can be determined by running (5.6), (5.7), and (5.8) for one extra step, namely if n is the first frame of the new coding rate,

$$\begin{aligned} \hat{F}^e(n) &= \max\{0, \hat{B}^e(n-1) - \hat{r}_c^{old}/f(n-1)\} & (5.9) \\ \hat{B}^e(n) &= \hat{F}^e(n) + b(n) & (5.10) \\ \hat{g}^{old}(n) &= B - B^e(n). & (5.11) \end{aligned}$$

It is easy to see that if $\hat{g}^{new}(n) = B - b(n)$, then $\Delta g(n) = \hat{F}^e(n)$ as computed in (5.9).

We may also use for $\hat{g}^{new}(n)$ any better bound on $g^{new}(n)$. Better bounds are the subject of future study.

5.3 Virtual Stream

In MBR streaming, video and audio data are usually encoded separately and generate multiple streams (*substreams*, hereafter), respectively. Various combinations of video and audio substreams lead to more choices of aggregate bit rates (thus, quality levels). On the other hand, this freedom of choice provides a mechanism to balance the preference between the video and audio quality. For example, if video quality is more important, then the control mechanism would try to adjust audio substreams before video substreams in the change of available bandwidth. Vice versa, if audio quality is more preferable, then it is possible to keep a high bit rate audio substream and only change video substreams to adapt to network dynamics.

Although our optimal coding rate control method is derived based on a single stream model, it can be easily extended to accommodate this video and audio substream combination by introducing the concept of *virtual stream*. A virtual stream is a combination of a video and an audio substream (possibly only single video/audio substream). And the rate control method updates the status of a virtual stream and makes switching decisions among virtual streams.

Next, we show that the leaky bucket (B, F^e, R) of a virtual stream can be easily derived from the composing video substream (B_v, F_v^e, R_v) and audio substream (B_a, F_a^e, R_a). We know that the average coding rate is the largest bit rate such that the encoder buffer will not run dry, therefore, $B^e(n) - \hat{r}_c/f(n)$ in (5.2) is always non-negative for $\hat{r}_c = R_v$ and $\hat{r}_c = R_a$. Thus,

$$F_v^e(n+1) = B_v^e(n) - R_v/f_v(n) \geq 0 \tag{5.12}$$

$$F_a^e(n+1) = B_a^e(n) - R_a/f_a(n) \geq 0, \tag{5.13}$$

Set the virtual stream leaky bucket as $B = B_v + B_a$, $F^e = F_v^e + F_a^e$ and $R = R_v + R_a$. It is easy to show the following by induction (even when the video and audio substream have different frame rates):

$$F^e(n+1) = B^e(n) - R/f(n) \geq 0. \tag{5.14}$$

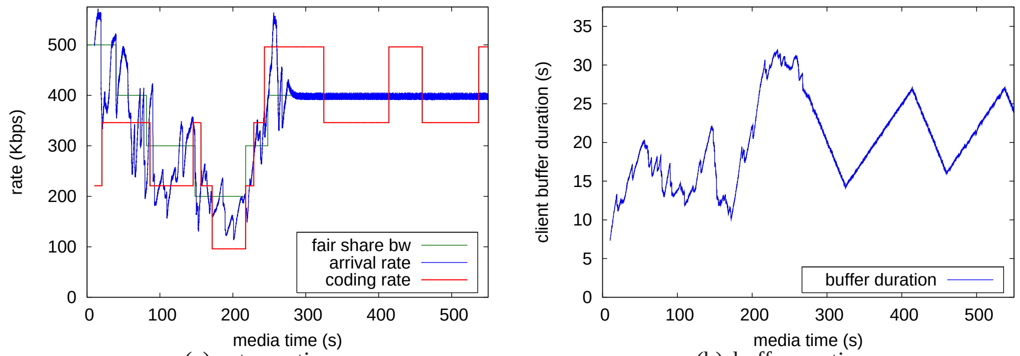
Therefore, (B, F^e, R) is a valid leaky bucket for the virtual stream, although it is not necessarily the tightest one corresponding to the coding rate R.

5.4 Performance Results

In this section, we present experimental results of our rate control with an MBR stream set under two sets of bandwidth conditions, both of which cause the client buffer to underflow in the the Windows Media 9 system. The bandwidth conditions are summarized in Table 5.1, and the results are shown in Figures 5.2 and 5.3.

Table 5.1: Bandwidth conditions with and without initial transmission rate burst

	without initial burst	with initial burst
0–5 s	500 Kbps	2 Mbps
5–25 s	500 Kbps	1 Mbps
25–70 s	400 Kbps	400 Kbps
70–130 s	286 Kbps	286 Kbps
130–190 s	200 Kbps	200 Kbps
190–220 s	286 Kbps	286 Kbps
220–550 s	400 Kbps	400 Kbps



(a) rate vs. time (b) buffer vs. time

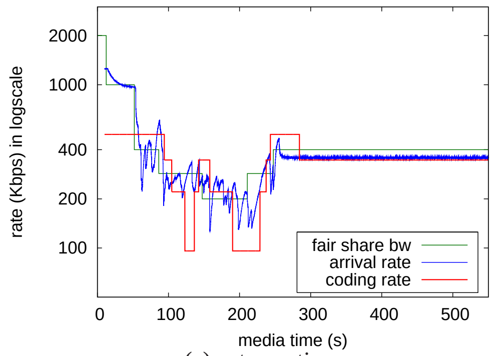
Figure 5.2: TCP variable bandwidth experiment (without initial transmission rate burst).

We then study the performance of our rate control under adversary network environments. In particular, we are interested in networks with severe data loss and long RTT.

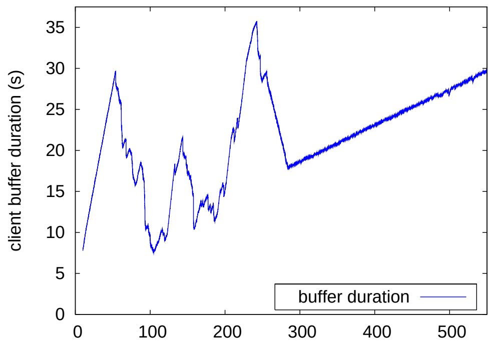
5.4.1 Performance Impact of Data Loss

Data loss is recovered by retransmission in all our experiments. When TCP protocol is used, retransmission is automatically taken care of by the transport protocol itself. When TFRC protocol is used, a NAK-based retransmission module is added to recover data loss. Due to retransmission, complete reception of frames might no longer in order although the server delivers them in sequence. Out-of-order frames would confuse the controller and thus are simply ignored by the rate control algorithm. It is, however, important to investigate the impact of omission frames on the rate control performance. On the other hand, data loss also has direct impact on transport protocols, which usually include a mechanism to adapt to packet loss. The change in transport layer (in turn transmission rate) will again affect the rate control performance.

Therefore, it is beneficial to design experiments such that the above two factors could be isolated. Indeed, we first simulate data loss at the client side after packets are received from the transport layer. This *application data loss* is transparent to transport layer and will hardly affect the

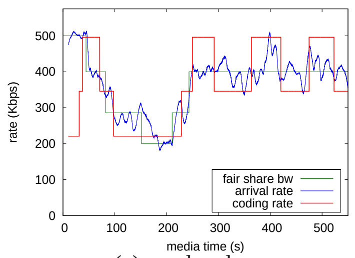


(a) rate vs. time

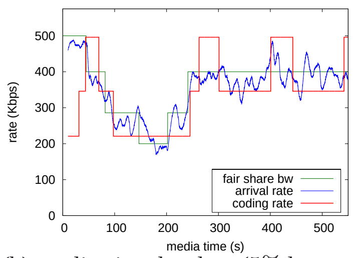


(b) buffer vs. time

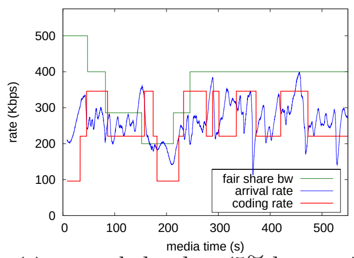
Figure 5.3: TCP variable bandwidth experiment (with initial transmission rate burst).



(a) no data loss



(b) application data loss (5% loss rate)



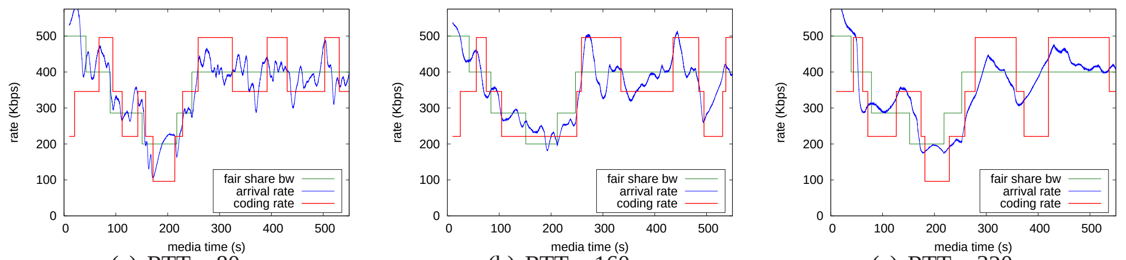
(c) network data loss (5% loss rate)

Figure 5.4: Performance Impact of Data Loss (over TFRC protocol)

transmission rate. Next, we simulate data loss inside the network layer (*network data loss*), which will affect the transmission rate. The performance impact of data loss becomes clear when the loss rate is increased to 5%, as in Figure 5.4. By comparing Figure 5.4(b) to 5.4(a), we can see that data loss does *not* affect the rate controller a lot even at high data loss. However, the overall impact of data loss is still significant as in Figure 5.4(c), where the fluctuation of the coding rate occurs as the transmission rate oscillates severely. Hence, the rate control algorithm should be applicable in network environments with severe data loss (e.g. wireless network), if the transport protocol could achieve stable transmission rate.

5.4.2 Performance Impact of RTT

The control interval is chosen to be $1s$ (virtual frame rate $f = 1$) and sufficient larger than RTTs in normal streaming sessions. Hence, our rate control model does *not* have to explicitly consider network delay, as explained in details in the previous chapter. It is also desirable to investigate experimentally the performance impact of various RTT values. From the results in Figure 5.5, we can see that doubling RTT from $80ms$ to $160ms$ and even $320ms$ does *not* have much impact on the coding rate control. And the coding rate differences over various RTT networks happen mainly due



(a) RTT = 80ms (b) RTT = 160ms (c) RTT = 320ms

Figure 5.5: Performance Impact of RTT (over TFRC protocol)

to the change of transmission rate pattern. Note that buffer status figures also show no underflow and are not duplicated here.

5.5 Related Work

Hsu, Ortega and Reibman [5] address the problem of joint selection of source and channel rates (which are notions analogous to coding and transmission rates in this chapter) for VBR video. They propose a rate-distortion optimization solution that maximizes receiving quality subject to end-to-end delay guarantees. Luna, Kondi and Katsaggelos [6] pursue this direction further by introducing network cost as an optimization objective and balancing the trade-off between user satisfaction and network cost. Both approaches assume networks that offer QoS support while using various policing mechanisms (such as a leaky bucket model) to constrain network traffic. The algorithms in these papers can be modified to address the problem, which we deal with in this chapter, where the channel rate is completely determined by network conditions and not subject to choice. However, a drawback of these algorithms compared to our optimal control mechanism is that they require complete knowledge of channel rates *a priori*, which makes them less practical for streaming media applications, where dynamic rate adjustment is required on the fly. Moreover, these algorithms have higher complexity, even with fast approximation variations [7]. The algorithms are good, however, for determining performance bounds in offline analysis.

With *a prior* knowledge of the network bandwidth, Nelakuditi, Harinath, Kusmierek and Zhang [8] design a bidirectional scan algorithm to optimize the perceived video quality, measured by a set of smoothness metrics. Their work uses layered video and simplifies analysis by assuming that each layer is of CBR. The recent work of Kim and Ammar [9] develops along this direction and proposes a more sophisticated algorithm targeting optimal quality adaptation for MPEG-4 FGS VBR video. Both work also provide online heuristics, when the available bandwidth is not known in advance. These online heuristics appeal to have reasonable good performance for limited scalability (one base layer and two enhancement layers in both work), although it is not clear how well they will work with a rich set of available bit rates (e.g. 50 streams in our case). Similarly, it might be difficult as well to extend the dynamic bandwidth allocation algorithm proposed by Saparilla and Ross [10] beyond a few but yet limited bit rates.

To our knowledge, the most closely related contemporaneous work is that by de Cuetos and Ross [11], which also decouples the transmission rate and the coding rate. They assume that the transmission rate is determined by the network transport protocol (TCP or TFRC), which is the same assumption that we make in this chapter. They develop a heuristic real time algorithm for adaptive coding rate control and compare its performance to an optimal offline coding rate control policy if the transmission rate is given prior to streaming. Our work differs from theirs in two ways. One is that our rate control algorithm is optimal in a control theoretic sense, in addition to being a low complexity real time algorithm. The other is that we take into account the variable instantaneous bit rate of the media coding and thereby further improve and stabilize the receiving quality.

The work of Rejaie, Handley and Estrin [12] proposes a scheme for transmitting layered video in the context of unicast congestion control, which basically includes two mechanisms. One mechanism is a coarse-grained mechanism for adding and dropping layers (changing the overall coding rate and quality). The other is a fine-grained interlayer bandwidth allocation mechanism to manage the receiver buffer (not changing the overall coding rate or quality). A potential issue with this approach is that it changes the coding rate by adding or dropping one (presumably coarse) layer at a time. If the layers are fine-grained, as in the case of FGS coded media, then adding or dropping one (fine-grained) layer at a time typically cannot provide a prompt enough change in coding rate. Moreover, since the adding and dropping mechanism is rather empirical, the mechanism may simply not be suitable for FGS media.

The work of Q. Zhang, Zhu and Y-Q. Zhang [13] proposes a resource allocation scheme to adapt the coding rate to estimated network bandwidth. The novelty of their approach is that they consider minimizing the distortion (or equivalently maximizing the quality) of all applications, such as file-transfers and web browsing in addition to audio/video streaming. However, their optimization process does not include the smoothness of individual streams and might lead to potential quality fluctuations. In our work, we explicitly take into account the smoothness of the average coding rate over consecutive frames in our optimal controller, which yields a higher and more stable quality as network conditions change.

Chapter 6

Conclusions

We conclude the thesis work with a summary of our contributions and an outline of future directions.

6.1 Summary

This thesis discusses several schemes for efficient and effective streaming media delivery, by identifying and addressing some key problems in various types of streaming media applications.

We study using MDS array codes as efficient FEC schemes for streaming media delivery with strong delay constraints. In particular, we propose the STAR code as a novel scheme for triple erasure recovery. The geometric property of the code construction leads to an efficient decoding algorithm. And the lower complexity of this scheme makes it attractive for many applications, such as streaming of live media, surveillance content, etc.

We also study using MDS array codes as practical FEC schemes at a bit level. In particular, we propose the XEOD as an efficient algorithm for bit level decoding of the EVENODD code. Our analysis shows significant throughput benefits and energy savings of this scheme, compared to the widely adopted RS code. The XEOD also achieves comparable loss recovery performance to the RS code, especially when data loss patterns are bursty.

For streaming media on demand, we describe the ORC scheme for client buffer management. Our approach is the first application of optimal control theory in this problem. We also explicitly incorporate the leaky bucket concept to maintain smooth user perception quality. Further, the ORC scheme is extended to MBR streaming, which is directly applicable to existing systems.

6.2 Future Directions

Throughout the thesis work, we have extensively studied using MDS array codes as FEC schemes for streaming media delivery, treating either columns or symbols inside columns as data packets. We have shown that both schemes can be applicable to certain types of streaming applications.

Compared to codes based on finite field operations, the benefit of these schemes mainly attributes to the efficiency of the basic operation – XOR sum. However, it also becomes clear that both schemes have their own limitations. Array codes tend to have limited block length when used at column level. To make array codes more flexible and applicable, we would like to continue to seek array codes with larger block length while maintaining the MDS property. On the other hand, when it is used at a symbol level, the decoding performance of array codes deviates from those of MDS codes. Along this direction, the class of fountain codes performs fairly well in terms of decoding performance. Fountain codes are also XOR-based and have efficient decoding algorithms. However, they usually require very large block lengths and are thus not directly applicable to streaming applications. To close the gap, we would like to continue investigating XOR-based schemes, which could provide flexible choices of coding parameters while achieving close to optimal coding performance.

The advancement of peer-to-peer technologies and the expansion of P2P networks provide huge platforms to store and disseminate streaming media content. These networks are often self-organized and have rather good adaptivity, although their scales are in general much smaller than the Internet. Recent research has suggested revisiting Internet flow regulations as distributed control problems and new findings along that direction are quite encouraging. During this thesis work, we also realized the effectiveness of control theory knowledge and how it helps us to understand and solve problems from that perspective. We would like to exploit the similarities between the streaming media delivery in P2P networks and the flow regulations in the Internet. Also, we would like to investigate the possibilities of addressing the P2P streaming problem by utilizing distributed control approaches. Along this direction, some well developed knowledge from other disciplines might also be worth exploiting, such as game theoretic approach.

As a double-edged sword, the overwhelmingly popularity of P2P networks might, on the other hand, jeopardize the accessibility of streaming media content, which could be simply cut back by content providers due to the lack of copyright protection. We believe another very important direction of streaming media research is *Digital Rights Management* (DRM) related technologies, which ensure media content protection through the entire session of streaming service.

References

[1] M. Claypool and J. Riedl, “End-to-end quality in multimedia applications,” in *Handbook on Multimedia Computing*, chapter 40. CRC Press, Boca Raton, Florida, 1999.

[2] Y. Wang and Q. Zhu, “Error control and concealment for video communication: A review,” in *Proceedings of the IEEE*, May 1998, vol. 86, pp. 974–997.

[3] I. S. Reed and G. Solomon, “Polynomial codes over certain finite fields,” *Journal of SIAM*, vol. 8, no. 10, pp. 300–304, 1960.

[4] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*, North-Holland, Amsterdam, 1977.

[5] C.-Y. Hsu, A. Ortega, and A. Reibman, “Joint selection of source and channel rate for VBR video transmission under ATM policing constraints,” *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 5, pp. 1016–1028, Aug. 1997.

[6] C. E. Luna, L. P. Kondi, and A. K. Katsaggelos, “Maximizing user utility in video streaming applications,” *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, no. 2, pp. 141–148, Feb. 2003.

[7] A. Ortega, K. Ramchandran, and M. Vetterli, “Optimal trellis-based buffered compression and fast approximation,” *IEEE Trans. Image Processing*, vol. 3, pp. 26–40, Jan. 1994.

[8] Srihari Nelakuditi, Raja R. Harinath, Ewa Kusmierek, and Zhi-Li Zhang, “Providing smoother quality layered video stream,” in *Proc. Int’l Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Chapel Hill, NC, June 2000.

[9] Taehyun Kim and Mostafa H. Ammar, “Optimal quality adaptation for MPEG-4 fine-grained scalable video,” in *Proc. Conf. Computer Communications (INFOCOM)*, San Francisco, CA, Apr. 2003.

[10] Despina Saparilla and Keith W. Ross, “Optimal streaming of layered video,” in *Proc. Conf. Computer Communications (INFOCOM)*, Tel-Aviv, Israel, Mar. 2000.

[11] P. de Cuetos and K. W. Ross, “Adaptive rate control for streaming stored fine-grained scalable video,” in *Proc. Int’l Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Miami Beach, FL, May 2002.

[12] R. Rejaie, M. Handley, and D. Estrin, “Layered quality adaptation for Internet streaming video,” *IEEE J. Selected Areas in Communications*, vol. 18, no. 12, pp. 2530–2543, Dec. 2000.

[13] Q. Zhang, Y.-Q. Zhang, and W. Zhu, “Resource allocation for multimedia streaming over the Internet,” *IEEE Trans. Multimedia*, vol. 3, no. 3, pp. 339–355, Sept. 2001.

[14] M. Blaum, P.G. Farrell, and H.C.A. van Tilborg, *Array Codes*, Chapter 22 in Handbook of Coding Theory. Elsevier Science B.V., 1998.

[15] M. Blaum, J. Brady, J. Bruck, and J. Menon, “EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures,” *IEEE Trans. Information Theory*, vol. 44, no. 2, pp. 192–202, Feb. 1995.

[16] Chih-Shing Tau and Tzone-I Wang, “Efficient parity placement schemes for tolerating triple disk failures in RAID architectures,” in *Proceedings of the17 th International Conference on Advanced Information Networking and Applications (AINA’03)*, Xi’an, China, mar 2003.

[17] Chong-Won Park and Jin-Won Park, “A multiple disk failure recovery scheme in RAID systems,” *Journal of Systems Architecture*, vol. 50, pp. 169–175, 2004.

[18] M. Blaum, J. Bruck, and A. Vardy, “MDS array codes with independent parity symbols,” *IEEE Trans. Information Theory*, vol. 42, no. 2, pp. 529–542, Mar. 1996.

[19] M. Blaum, J. Brady, J. Bruck, J. Menon, and A. Vardy, “The EVENODD code and its generalization,” in *High Performance Mass Storage and Parallel I/O*, pp. 187–208. John Wiley & Sons, INC., 2002.

[20] L. Xu, J. Bruck, and D. Wagner, “Low density MDS codes and factors of complete graphs,” *IEEE Trans. Information Theory*, vol. 45, no. 6, pp. 1817–1826, Sept. 1999.

[21] L. Xu and J. Bruck, “X-code: Mds array codes with optimal encoding,” *IEEE Trans. Information Theory*, vol. 45, no. 1, pp. 272–276, Jan. 1999.

[22] Nam-Kyu Lee, Sung-Bong Yang, and Kyoung-Woo Lee, “Efficient parity placement schemes for tolerating up to two disk failures in disk arrays,” *Journal of Systems Architecture*, vol. 46, pp. 1383–1402, 2000.

[23] M. Blaum and R. M. Roth, “New array codes for multiple phased burst correction,” *IEEE Trans. Information Theory*, vol. 39, no. 1, pp. 66–77, Jan. 1993.

[24] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, “An XOR-based erasure-resilient coding scheme,” Technical Report No. TR-95-048, ICSI, Berkeley, California, Aug. 1995.

[25] J. S. Plank, “A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems,” *Software: Practice and Experience*, vol. 27, no. 9, pp. 995–1012, Jan. 1999.

[26] L. Rizzo, “Sources for an erasure code based on Reed-Solomon coding with vandermonde matrices,” http://info.iet.unipi.it/∼luigi/vdm98/vdm980702.tgz.

[27] L. Rizzo, “Effective erasure codes for reliable computer communication protocols,” *ACM Computer Communication Review*, Apr. 1997.

[28] L. Rizzo and L. Vicisano, “A reliable multicast data distribution protocol based on software fec techniques (rmdp),” in *Proc. of the Fourth IEEE HPCS’97 Workshop*, Chalkidiki, Greece, June 1997, IEEE.

[29] U. Horn and B. Girod, “Scalable video transmission for the Internet,” *Computer Networks and ISDN Systems*, vol. 29, no. 15, pp. 1833–1842, Nov. 1997.

[30] B. Girod, K. Stuhlmuller, M. Link, and U. Horn, “Packet loss resilient internet video streaming,” in *Proc. Visual Communications and Image Processing*, San Jose, CA, Jan. 1999, SPIE.

[31] R. Puri and K. Ramchandran, “Multiple description source coding through forward error correction codes,” in *Proc. Asilomar Conf. Signals, Systems, and Computers*, Asilomar, CA, Oct. 1999, IEEE, vol. 1, pp. 342–346.

[32] P. A. Chou, A. E. Mohr, A. Wang, and S. Mehrotra, “FEC and pseudo-ARQ for receiver-driven layered multicast of audio and video,” in *Proc. Data Compression Conference*, Snowbird, UT, Mar. 2000, IEEE Computer Society, pp. 440–449.

[33] P. A. Chou, A. E. Mohr, A. Wang, and S. Mehrotra, “Error control for receiver-driven layered multicast of audio and video,” *IEEE Trans. Multimedia*, vol. 3, no. 1, pp. 108–122, Mar. 2001.

[34] T. Nguyen and A. Zakhor, “Distributed video streaming with forward error correction,” in *Proc. Int’l Packet Video Workshop*, Pittsburg, PA, Apr. 2002.

[35] P. A. Chou, H. J. Wang, and V. N. Padmanabhan, “Layered multiple description coding,” in *Proc. Int’l Packet Video Workshop*, Nantes, France, Apr. 2003.

[36] Q. Zhang, W. Zhu, Z. Ji, and Y. Q. Zhang, “A power-optimized joint source channel coding for scalable video streaming over wireless channel,” in *ISCAS-Proc*, Sydney, Australia, may 2001.

[37] A. Majumdar, D. G. Sachs, I. V. Kozintsev, K. Ramchandran, and M. M. Yeung, "Multicast and unicast real-time video streaming over wireless LANs," *IEEE-Trans-CSVT*, vol. 12, no. 6, pp. 524–534, jun 2002.

[38] W. Kumwilasisak, J. Kim, and C.-C. J. Kuo, "Reliable wireless video transmission via fading channel estimation and adaptation," in *WCNC*, Chicago, IL, Sept. 2000, IEEE.

[39] Q. Zhang, G. Wang, Z. Xiong, J. Zhou, and W. Zhu, "Error robust scalable audio streaming over wireless IP networks," *IEEE Trans. Multimedia*, vol. 6, no. 6, pp. 897–907, Dec. 2004.

[40] M. Blaum, J. Fan, and L. Xu, "Soft decoding of several classes of array codes," in *ISIT-Proc*, Lausanne, Switzerland, June 2002.

[41] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *IEEE-Trans-IT*, vol. 47, no. 2, pp. 569–584, Feb. 2001.

[42] M. A. Shokrollahi, "Raptor codes," in *Proceedings of ISIT*, July 2004.

[43] M. Luby, "LT codes," in *FOCS 2002*, Nov. 2002.

[44] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege, "A digital fountain approach to reliable distribution of bulk data," in *Proceedings of ACM SIGCOMM*, 1998, pp. 56–67.

[45] M. Guthaus, J Ringenberg, and et al. Dan Ernst, "Mibench: A free, commercially representative embedded benchmark suite," in *IEEE 4th Annual Workshop on Workload Characterization*, Dec. 2001.

[46] "Sim-panalyzer," http://www.eecs.umich.edu/∼panalyzer.

[47] "Smpeg library," http://www.lokigames.com/development/smpeg.php3.

[48] W. Stallings, *Cryptography and Network Security, Principles and Practices*, Prentice Hall, 3 edition, 2003.

[49] "Crypto++ library 5.1," http://www.eskimo.com/∼weidai/cryptlib.html.

[50] J. Bolot, S. Fosse-Parisis, and D. Towsley, "Adaptive FEC-based error control for interactive audio on the Internet," in *Proc. Infocom*, New York, NY, Mar. 1999, IEEE.

[51] W. Birney, "Intelligent streaming," http://www.microsoft.com/windows/windowsmedia/-howto/articles/intstreaming.aspx, May 2003.

[52] M. Kalman, E. Steinbach, and B. Girod, "Adaptive media playout for low delay video streaming over error-prone channels," *IEEE Trans. Circuits and Systems for Video Technology*, To appear.

[53] G.J. Conklin, G.S. Greenbaum, K.O. Lillevold, A.F. Lippman, and Y.A. Reznik, "Video coding for streaming media delivery on the Internet," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 269–281, Mar. 2001, special issue on Streaming Video.

[54] T. Wiegand and G. Sullivan, "Joint video specification rec. h.264 & 14496-10 avc," Non-Final Draft of Final Draft International Standard (FDIS) JVT-G050, ITU-T & ISO/IEC, Pattaya, Thailand, Mar. 2003.

[55] B. G. Haskell and A. Puri and, *Digital Video: An Introduction to MPEG-2*, Chapman & Hall, New York, 1997.

[56] B.-J. Kim, Z. Xiong, , and W. A. Pearlman, "Low bit-rate scalable video coding with 3D set partitioning in hierarchical trees (3-D SPIHT)," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 10, no. 8, pp. 1374–1387, Dec. 2000.

[57] F. Wu, S. Li, and Y.-Q. Zhang, "A framework for efficient progressive fine granularity scalable video coding," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 301–317, Mar. 2001.

[58] J. Li, "Embedded audio coding (eac) with implicit psychoacoustic masking," in *Proc. Int'l Conf. Multimedia*, Nice, France, Dec. 2002, ACM, pp. 592–601.

[59] P. A. Chou and Z. Miao, "Rate-distortion optimized streaming of packetized media," Tech. Rep. MSR-TR-2001-35, Microsoft Research, Redmond, WA, Feb. 2001.

[60] P. A. Chou and Z. Miao, "Rate-distortion optimized streaming of packetized media," *IEEE Trans. Multimedia*, 2001, submitted.

[61] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proc. Data Communication, Ann. Conf. Series (SIGCOMM)*, Stockholm, Sweden, Aug. 2000, ACM.

[62] R. Rejaie, M. Handley, and D. Estrin, "RAP: an end-to-end based congestion control mechanism for realtime streams in the Internet," in *Proc. Conf. Computer Communications (INFOCOM)*, New York, NY, Mar. 1999, IEEE, vol. 3, pp. 1337–1345.

[63] D. Bansal and H. Balakrishnan, "Binomial congestion control algorithms," in *Proc. Infocom*. IEEE, Apr. 2001.

[64] I. Rhee, V. Ozdemir, and Y. Yi, "TEAR: TCP emulation at receivers: flow control for multimedia streaming," Tech. Rep., Dept. of Computer Science, North Carolina State University (NCSU), Apr. 2000.

[65] D. Sisalm and A. Wolisz, “LDA+ TCP-friendly adaptation: a measurement and comparison study,” in *Proc. Int’l Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Miami Beach, FL, May 2002.

[66] J. Ribas-Corbera, P. A. Chou, and S. Regunathan, “A generalized hypothetical reference decoder for H.264/AVC,” *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, no. 7, July 2003.

[67] B. D. O. Anderson and J. B. Moore, *Optimal Control: Linear Quadratic Methods*, Prentice Hall, 1990.

[68] G. Franklin, J. Powell, and M. Workman, *Digital Control of Dynamic Systems*, Addison-Wesley Longman, Inc., 3rd ed. edition, 1997.

[69] C. Huang, P. A. Chou, and A. Klemets, “Optimal coding rate control for scalable and multi bit rate streaming media,” Tech. Rep. MSR-TR-2005-47, Microsoft Research, Redmond, WA, Apr. 2005.

[70] S. Keshav, “Packet-pair flow control,” http://www.cs.cornell.edu/skeshav/papers.html.

[71] V. J. Ribeiro, R. H. Riedi, J. Navratil, L. Cottrell, and R. G. Baraniuk, “pathChirp: efficient available bandwidth estimation for network paths,” in *Proc. Passive and Active Measurement Workshop (PAM)*, La Jolla, CA, Apr. 2003.

[72] K. Fall and eds. K. Varadhan, “The *ns* manual,” Tech. Rep., The VINT Project, Dec. 2003, http://www.isi.edu/nsnam/ns/.

Vita

Cheng Huang

Degrees	B.Sc., Electrical Engineering, Shanghai Jiao Tong University, Jul. 1997. M.Sc., Electrical Engineering, Shanghai Jiao Tong University, Jan. 2000. M.Sc., Computer Science, Washington University, May 2002.

Publications

- Cheng Huang and Lihao Xu, *Optimal Broadcast Scheduling for Random-Loss Channels*, (to appear) Proc. IEEE International Symposium on Information Theory (ISIT 2005), Adelaide, Australia, Sep. 2005.
- Cheng Huang and Lihao Xu, *Study of A Practical FEC Scheme for Wireless Data Streaming*, Proc. IASTED Internet and Multimedia Systems and Applications (EuroIMSA 2005), Grindelwald, Switzerland, Feb. 2005.
- Cheng Huang, Philip A. Chou and Anders Klemets, *Optimal Coding Rate Control for Scalable Streaming Media*, Proc. International Packet Video Workshop (PV 2004), Irvine, CA, Dec. 2004.
- Cheng Huang, Philip A. Chou and Anders Klemets, *Optimal Control of Multiple Bit Rates for Streaming Media*, Proc. Picture Coding Symposium (PCS 2004), San Francisco, CA, Dec. 2004.
- Cheng Huang, Ramaprabhu Janakiraman and Lihao Xu, *Optimal Loss-Resilient Media Streaming using Priority Encoding*, Proc. ACM International Conference on Multimedia (MM 2004), New York, NY, Oct. 2004.
- Cheng Huang and Lihao Xu, *SRC: Stable Rate Control for Streaming Media*, Proc. IEEE Global Telecommunications Conference (GLOBECOM 2003), San Francisco, CA, Dec. 2003.

August 2005

Short Title: Streaming Media Delivery Huang, D.Sc. 2005