ON LARGE-SCALE PEER-TO-PEER STREAMING SYSTEMS

BY

CHEN FENG

A THESIS SUBMITTED IN CONFORMITY WITH THE REQUIREMENTS FOR THE DEGREE OF MASTER OF APPLIED SCIENCE, DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING, AT THE UNIVERSITY OF TORONTO.

COPYRIGHT © 2009 BY CHEN FENG. ALL RIGHTS RESERVED.
Abstract

Peer-to-peer (P2P) streaming has recently received much research attention, with successful commercial systems showing its viability in the Internet. Despite the remarkable popularity in real-world systems, the fundamental properties and limitations are not yet well understood from a theoretical perspective, as there exists a significant gap between the fundamental limits and the performance achieved in practice. In this thesis, we seek to provide an in-depth analytical understanding of fundamental properties and limitations of P2P streaming systems, with a particular spotlight on the performance gap. We first identify the major problem in existing streaming protocols and show that this problem accounts for most of the gap separating the actual and optimal performances of the streaming systems. We then propose a remedy based on network coding to address this problem and show that the gap to the fundamental limits can be significantly reduced.
To my parents
Acknowledgments

I would like to acknowledge the help of many incredible individuals during my M.A.Sc. study at the University of Toronto. Foremost, I would like to express my sincere gratitude to my supervisor, Professor Baochun Li, for his guidance and support throughout the past two and a half years. I have benefited tremendously from his unique blend of vision, technical insights and practical sensibility. I am deeply indebted to many stimulating discussions with him on my research work. His intellectual curiosity and total dedication to research have been endless sources of inspiration for me.

I thank Professor Frank R. Kschischang for being a great teacher and a good friend. His wonderful course on error control codes helped me to clarify my understanding of coding theory, in particular, the network coding theory, which is fundamental to the discussion in Chapter 4. Professor Kschischang also offered me excellent practical counsel on academia in general, especially the importance of communicating with other researchers and useful tips on giving a talk.

I thank Professor Ben Liang for giving me the opportunity to TA his stochastic processes course (when I really needed it!) and for being my thesis committee member even when he is still on sabbatical. I thank Professor Peter Marbach for his great course on systems modeling and analysis, from which I really gained much and had a lot of fun.

Various valuable and critical feedback has been provided to me from my thesis committee, which helped me to further improve the quality of my thesis. I would like to show my sincere gratitude to everyone in the committee: Professor Baochun Li, Professor Ben Liang, Professor Peter Marbach, and Professor Frank R. Kschischang.

I have learned much from my colleagues in the iQua research group, including Jin Jin, Elias Kehdi, Yunfeng Lin, Zimu Liu, Di Niu, Hassan Shojania, Mea Wang, Chuan
Wu, Jiahua Wu, Henry Xu and Xinyu Zhang. I have profited immensely from their suggestions, high standards and friendly company. Yunfeng, Di, Mea, Chuan, thank you for your comments on my papers, talks and most importantly my research taste! Jin, you deserve credit for tolerating me as your housemate for a couple of months. Xinyu, thanks for introducing me to the passion of foosball.

Finally, I wish to send my gratitude to my parents. Their unconditional love and support have been the most important power that kept my perseverance throughout the past years. Without their constant love, care and encouragement from overseas, I would not have come this far. The spirit of this thesis is dedicated to them.
Contents

Abstract ii

Acknowledgments iii

List of Tables vii

List of Figures xi

1 Introduction 1
   1.1 Background . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
   1.2 An Overview of the Thesis . . . . . . . . . . . . . . . . . . . . . . . . . 4

2 Preliminaries 7
   2.1 System Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
   2.2 Fundamental Constraints . . . . . . . . . . . . . . . . . . . . . . . . . . 9

3 Pull-based Streaming Protocols 13
   3.1 A Study of Fundamental Limits . . . . . . . . . . . . . . . . . . . . . . 13
   3.2 Understanding the Performance Gap . . . . . . . . . . . . . . . . . . . 24
      3.2.1 Effect of Periodic buffer-map Exchanges . . . . . . . . . . . . 24
List of Tables

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Theoretical Studies of P2P Streaming Systems</td>
</tr>
<tr>
<td>2.1</td>
<td>Key Notations in the System Model</td>
</tr>
<tr>
<td>3.1</td>
<td>Downloading blocks on Each Participating Peer</td>
</tr>
<tr>
<td>3.2</td>
<td>Numerical examples of playback skip rates. We set the relative server capacity $u_s$ to 2. The number of peers in the system is set to 10000. We vary the relative peer capacity $u_p$ from 1.1 to 1.5.</td>
</tr>
<tr>
<td>3.3</td>
<td>Numerical examples of playback skip rates. We set the relative peer capacity $u_p$ to 1.1. The number of peers in the system is set to 10000. We vary the relative server capacity $u_s$ from 2 to 10.</td>
</tr>
<tr>
<td>3.4</td>
<td>Downloading blocks on Each Participating Peer</td>
</tr>
<tr>
<td>4.1</td>
<td>Key Notations in the System Model</td>
</tr>
</tbody>
</table>
4.2 Theoretical and simulation results for relative additional server capacity to handle peer dynamics in the worst case. We can see that the theoretical bound is tight when bandwidth supply barely exceeds bandwidth demand, while the bound is loose when supply outstrips demand. In our simulations, we set the system scale $N$ to 10,000, the ratio $u_1$ to 2, and the ratio $u_2$ to 0.5. ........................................... 61

4.3 Simulation results for relative additional server capacity to handle peer dynamics in the average case. We can see that only a small amount of additional server capacity is required, even when 50% peers leave the system. We set the system scale $N$ to 10,000 in our simulations. .................. 62

4.4 Simulation results for fraction $\alpha$ of redundant coded blocks. We can see that the fraction of redundancy induced by network coding is in the order of 0.001, even when the field size $q$ is as small as 64. ....................... 76

4.5 Initial buffering delay (in seconds) in PULL using the same configuration as in Fig. 5.1(a) (denoted as flash) and in Fig. 5.1(b) (denoted as churn). In CODING, the initial buffering delay strictly ranges from 5 to 10 seconds regardless of peer dynamics. .............................. 79
List of Figures

1.1 An illustration of a peer-to-peer streaming system. . . . . . . . . . . . . . 2

3.1 An example to illustrate block scheduling using the Graph Labeling algo-

rithm. We set $N = 6$ and $u_s = u_p = 1$ in this example. That is, both

the streaming server and the participating peer can upload only one block
during each time slot. The minimum delay bound in this example is 4
time slots. This example shows that each block can be disseminated to all
participating peers within the minimum delay bound (4 time slots) after
it is injected by the streaming server. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

3.2 An example to illustrate block scheduling using the generalized Graph

Labeling algorithm. We set the number of peers $N$ to 6, the period of
buffer-map exchanges $T$ to 2. Both the relative server capacity $u_s$ and the
relative peer capacity $u_p$ are set to 1. In this example, the minimum initial
buffering delay is 6 time slots, and the time unit is 2 time slots. This
example shows that each block can be disseminated to all participating
peers within the minimum initial buffering delay (6 time slots or 3 time
units) after it is injected by the streaming server. . . . . . . . . . . . . . . . 29
3.3 Validation of the difference equation approximations for our simple pull-based streaming scheme. In (a), we set the relative server capacity $u_s$ to 2 and the relative peer capacity $u_p$ to 1.1. The number of peers in the system is set to 10000. In (b), we set the relative server capacity $u_s$ to 2 and the relative peer capacity $u_p$ to 1.1. The number of peers in the system is set to 100000.

3.4 Comparisons of the performance gap in terms of the initial buffering delay. In (a), we set the relative server capacity $u_s$ to 2 and the relative peer capacity $u_p$ to 1.1. The number of peers in the system is set to 100000. We vary the period of buffer-map exchanges from 2 time slots to 10 time slots. In (b), we set the relative server capacity $u_s$ to 2 and the relative peer capacity $u_p$ to 1.1. The period of buffer-map exchanges is set to 4 time slots. We vary the number of peers in the system from 20000 to 100000.

3.5 Comparisons of the performance gap in terms of the sustainable streaming rate. We set the server upload capacity $U_s$ to be 2 times of the average peer upload capacity $U_p$. We vary the number of peers in the system from 12500 to 800000.

4.1 An example to illustrate the coding operation on peer $p$, where peer $p$ has received 3 coded blocks within the segment $s$, and each segment consists of 6 blocks.

4.2 An illustration of random push on a random mesh structure. With network coding, multiple upstream peers are able to perform push operations on coded blocks within a segment without any explicit coordination.

4.3 An illustrative comparison of playback buffers between CODING and PULL.
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4</td>
<td>An illustration of initial buffering delays in CODING, which shows that the initial buffering delay on a newly joined peer is determined by the number of segments to be skipped and the current point of playback.</td>
<td>49</td>
</tr>
<tr>
<td>4.5</td>
<td>Validation of the required relative server capacity in several different flash crowd scenarios (theoretical results use dashed lines; simulation results use solid points).</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>In (a), we set the number of coded blocks in a segment (m) to 100 and vary the relative average peer capacity (u_p) from 1.05 to 1.09; while in (b), we set (u_p) to 1.05 and vary (m) from 100 to 200. We can see that analytical results match simulation results quite well, especially when the system scale is larger than 100,000.</td>
<td></td>
</tr>
<tr>
<td>4.6</td>
<td>Impact of restricted neighborhoods on the playback quality, with 95% confidence intervals.</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>In (a), we set the relative average peer capacity (u_p) to 1.05 and the number of coded blocks in a segment (m) to 100; while in (b), we set (u_p) to 1.07 and (m) to 100. The required relative server capacities in these two cases are 49 and 98, respectively, under the assumption of complete graphs. By restricting the size of neighborhood, we observe that the playback quality is still close to 1, as long as the average size of neighborhood is larger than 50.</td>
<td></td>
</tr>
<tr>
<td>4.7</td>
<td>A comparison of playback quality between CODING and PULL under different peer dynamic scenarios.</td>
<td>78</td>
</tr>
<tr>
<td></td>
<td>In (a), we set the relative server capacity (u_s) to 50 and the relative average peer capacity (u_p) to 1.08; In (b), we set (u_s) to 60 and (u_p) to 1.05.</td>
<td></td>
</tr>
</tbody>
</table>
4.8 The change of playback quality over time in CODING and PULL under a typical flash crowd scenario and a highly dynamic scenario. We set the relative server capacity $u_s$ to 60 and relative average peer capacity $u_p$ to 1.05 in both scenarios. The comparison shows that CODING has much better stability under peer dynamics than PULL. . . . . . . . . . . . . . . . . 80
Chapter 1

Introduction

Live peer-to-peer (P2P) streaming has recently witnessed unprecedented growth on the Internet, delivering live streaming content to millions of users at any given time. The essential advantage of peer-to-peer streaming is to dramatically increase the number of peers a streaming channel may sustain with dedicated streaming servers. Intuitively, as participating peers contribute their upload bandwidth capacities to serve other peers in the same channel, the load on dedicated streaming servers is significantly reduced.

Despite the remarkable commercial success in real-world systems and the wide research attention in the literature, there are still a number of challenges in the design of “good” P2P streaming systems. First, if media blocks do not arrive in a timely fashion, they have to be skipped at playback, which degrades the playback quality. How do we maintain a high playback quality at all participating peers? Second, an initial buffering delay must be experienced by a peer when it first joins or switches to a new channel. How do we improve user experience with the shortest initial buffering delay? Third, how do we design a streaming system that is simple to implement with low protocol overhead?
1.1 Background

Existing P2P streaming protocols generally fall into two strategic categories. In push-based tree streaming strategies (e.g., [1]), participating peers are organized into one or more multicast trees, and streaming content is disseminated along these trees. In pull-based mesh streaming strategies (e.g., [2]), the streaming content is presented as a series of blocks, each representing a short duration of playback, and each peer maintains a list of neighboring peers, with whom it periodically exchanges block availability information of streaming buffers. Based on such information, blocks are pulled from appropriate neighbors, in order to meet playback deadlines. Compared to push-based tree strategies, pull-based mesh strategies are more resilient to peer dynamics and simpler to implement. However, such an advantage is achieved at the cost of increased delay of distributing...
1.1. BACKGROUND

streaming content from servers to all participating peers, due to delays caused by periodic exchanges of block availability [3]. Nevertheless, most real-world systems (e.g., PPLive [4]) are implemented using pull-based mesh strategies, mainly due to their simplicity.

Beyond protocol design, there exist a small number of previous papers that focused on analytical studies of P2P streaming systems. L. Massoulie et al. [27] proposed a push-based streaming protocol and proved it is able to achieve the maximum streaming rate for static streaming systems. Although Massoulie et al. gave a rate optimality result, they did not provide any performance guarantees on the delay performance, which we believe are more important than the maximum streaming rate, as bit rates of most multimedia streams are known \emph{a priori}. T. Bonald et al. [11] (with L. Massoulie as a co-author) later pointed out that the delay performance of the original Massoulie protocol is poor, and proposed several new push-based protocols to improve the delay performance. Bonald et al. proved that some of these new protocols can achieve near-optimal streaming rate and delay in static streaming systems. However, performances of these new protocols in dynamic streaming systems still remain an open problem.

Instead of push-based protocols (which are not widely implemented in real-world applications), Y. Zhou et al. [28] studied pull-based streaming protocols, with a focus on the impact of block selection strategies. They further proposed a mixed strategy in order to achieve both good playback quality and good delay performance for a given streaming rate. However, they only considered \emph{static} streaming systems with \emph{homogeneous} peer upload capacities. In addition, they did not investigate the bandwidth cost on dedicated streaming servers.

With respect to performance bounds in P2P streaming systems, R. Kumar et al. [9] studied the maximum streaming rate in dynamic mesh-based streaming systems. Y. Liu
1.2. AN OVERVIEW OF THE THESIS

[10] provided the minimum delay for static mesh-based streaming systems. S. Liu et al. [29] derived the minimum server bandwidth cost and maximum streaming rate for static tree-based systems. However, all of them critically depend on centralized scheduling algorithms to approach the optimal values. Intuitively, centralized protocols are impossible to implement realistically, especially in large scale P2P streaming systems.

1.2 An Overview of the Thesis

Theoretical studies of P2P streaming systems established several fundamental limits on the maximum streaming rate and the minimum delay. They also provided a number of algorithms to approach the optimal values. However, none of them has been implemented by system designers so far, mainly due to the complexity and overhead issues. Instead, most real-world systems are still using simple pull-based mesh protocols, which, however, do not have theoretical guarantees to achieve the optimal performance. This illustrated a mismatch between the fundamental limits in theory and the achieved performance in practice.

The goal of this thesis is an attempt to analytically understand and bridge the performance gap between the fundamental limits in theory and the achieved performance in practice. In particular, we will first identify the major problem in practical pull-based streaming protocols. We will then provide a remedy with the help of network coding to address this major problem. Our final result is a practical network-coding-based streaming solution that enjoys theoretical guarantees to reduce the gap to the fundamental limits significantly.

In Chapter 3, we mathematically understand the performance of pull-based mesh streaming protocols, with a focus on a number of fundamental questions: What are the
fundamental limits of the performance of pull-based mesh protocols? How large is the gap between the fundamental limits and the actual performance? What factors account for most of the gap separating the actual and optimal performance of pull-based mesh protocols?

To achieve this objective, we have developed a theoretical framework based on the trellis graph technique that has been traditionally used in the network coding literature [6]. In particular, we have unified the treatment of the analysis of the fundamental limits and the actual performance of pull-based mesh protocols. This provides a solid theoretical foundation for the characterization of the performance gap between the fundamental limits and the actual performance. With this framework, we perform an in-depth study of several important factors that account for the performance gap and quantify their impact on the performance of pull-based mesh protocols.

We show that there exists a significant performance gap that separates the performance of pull-based protocols and the fundamental limits. We show that periodic exchanges of block availability information account for most of this performance gap. These analytical results bring us a number of important insights into pull-based protocols. These results will appear in [7].

In Chapter 4, we identify several simple design principles for streaming systems with network coding and show that any streaming protocol following our design principles is sufficient to achieve provably good performance in realistic settings. In particular, we prove that our design principles naturally lead to a near-optimal streaming rate and very short initial buffering delays during flash crowds. We also show that a reasonable server bandwidth cost is enough to handle even the most volatile peer dynamics with ease when adopting our design principles.
Table 1.1: Theoretical Studies of P2P Streaming Systems

<table>
<thead>
<tr>
<th></th>
<th>Decentralized</th>
<th>Dynamics</th>
<th>Heterogeneity</th>
<th>Coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>This thesis</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>L. Massoulie et al. [27]</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>T. Bonald et al. [11]</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Y. Zhou et al. [28]</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>R. Kumar et al. [9]</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Y. Liu [10]</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>S. Liu et al. [29]</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
</tbody>
</table>

Different from previous analytical papers on P2P streaming systems, as outlined in Table 1.1, we provide a rigorous performance analysis in Chapter 4 on the network-coding-based streaming solution that follows our design principles, simultaneously considering a number of performance metrics: playback quality, streaming rate, initial buffering delays, resilience to peer dynamics, as well as bandwidth cost on dedicated streaming servers. In addition, the simplicity in our design principles enables us to use a more realistic system model without restricting our analysis to centralized protocols. In particular, we consider highly dynamic systems with volatile peers and heterogeneous peer upload capacities. This work has been published in [5].

The remainder of the thesis is structured as follows. In Chapter 2, we provide mathematical preliminaries for P2P streaming systems. Chapter 3 investigates the performance of pull-based streaming protocols and performs an in-depth study of important factors that account for the performance gap between the fundamental limits and the actual performance of pull-based protocols. Chapter 4 discusses our design principles for streaming systems with network coding, and presents a number of analytical results with a focus on the provably good performance of our network-coding-based streaming solution. Finally, Chapter 5 concludes the thesis with a recap of the contributions.
Chapter 2

Preliminaries

In this chapter, we first present our mathematical model for P2P streaming systems, including the underlying assumptions and the key notations summarized in Table 2.1. We then provide several fundamental constraints in P2P streaming systems.

2.1 System Model

We consider a streaming system with \( N \) participating peers. The streaming content is presented as a series of blocks, each representing a short duration of playback. We assume peer upload capacities are the only bottlenecks in the streaming system, which is in accordance with measurement studies of existing P2P systems (e.g., [8]). Let \( U_i \) denote the upload capacity of peer \( i \) (\( i \in \{1, 2, \ldots, N\} \)). For a given streaming rate \( R \), we define the relative capacity \( u_i \) of peer \( i \) as the ratio of the upload capacity \( U_i \) to the streaming rate \( R \). Let \( U_p \) be the average peer upload capacity and let \( u_p \) denote the relative average peer capacity, which is defined as the ratio of the average peer upload capacity \( U_p \) to the streaming rate \( R \).
We assume that there is only one streaming server in the system with upload capacity $U_s$. If multiple streaming servers exist in the system, they can be regarded as a super streaming server with upload capacity $U_s$ equaling to the total upload capacities. The relative server capacity $u_s$ is defined as the ratio of the server upload capacity $U_s$ to the streaming rate $R$.

We use an undirected graph $G = (V, E)$ to represent the neighborhood relationship (referred to as overlay topology in the literature) in the streaming system. We assume that the nodes are numbered so that $V = \{0, 1, 2, \ldots, N\}$ and node $i$ corresponds to peer $i$ (node 0 is the streaming server). Two nodes $i$ and $j$ in the graph can communicate with each other if and only if they are neighbors ($\{i.e.,\} (i, j) \in E$).

<table>
<thead>
<tr>
<th>$N$</th>
<th>Number of peers in the system.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>Streaming rate.</td>
</tr>
<tr>
<td>$U_i$</td>
<td>Upload capacity of peer $i$.</td>
</tr>
<tr>
<td>$u_i$</td>
<td>Relative capacity of peer $i$ ($= U_i / R$).</td>
</tr>
<tr>
<td>$U_p$</td>
<td>Average peer upload capacity.</td>
</tr>
<tr>
<td>$u_p$</td>
<td>Relative average peer capacity ($= U_p / R$).</td>
</tr>
<tr>
<td>$U_s$</td>
<td>Server upload capacity ($= U_0$).</td>
</tr>
<tr>
<td>$u_s$</td>
<td>Relative server capacity ($= U_0 = U_s / R$).</td>
</tr>
</tbody>
</table>

A time model determines when nodes in the system communicate with each other corresponding to their upload capacities. We consider both a synchronous and an asynchronous time model. These models are defined as follows.

**Synchronous:** Time is measured in time slots, which are common to all nodes in the system. The duration of each time slot is equal to the duration of playing back one media block. In any time slot, node $i$ can transmit $\lfloor u_i \rfloor$ blocks, plus one additional block with probability $u_i - \lfloor u_i \rfloor$, corresponding to its relative capacity $u_i$.

**Asynchronous:** In this model, time is discretized according to the ticks of various
clocks. Node $i$ has an independent clock that ticks according to a Poisson process of rate $u_i$. When a node’s clock ticks, it chooses one neighbor and transmits one block.

## 2.2 Fundamental Constraints

Now we discuss several fundamental constraints and their implications for P2P streaming systems. First of all, we observe that the total bandwidth consumption should be no greater than the total bandwidth supply. This leads to the following lemma, which has been proved in [9].

**Lemma 2.1** For a streaming system with given server upload capacity $U_s$ and average peer upload capacity $U_p$, the maximum sustainable streaming rate $R_{\text{max}}$ is given by:

$$R_{\text{max}} = U_p + \frac{U_s}{N},$$

where $N$ is the number of peers in the system.

In particular, if $N$ tends to infinity, then the upper bound in Lemma 2.1 reduces to $R_{\text{max}} \leq U_p$. In other words, the relative average peer capacity $u_p$ satisfies $u_p \geq 1$ in large-scale streaming systems.

Another fundamental constraint for P2P streaming systems is as follows: a peer cannot upload a block until it completes the download of this block. As shown in [10], this constraint sets up a limit on how fast a block can be disseminated to all peers in the system.

**The synchronous time model.** The special case of homogeneous upload capacity with $u_p = 1$ has been discussed in both [11] and [10]. Here, we discuss the general
homogeneous case where \( u_i = u_p \geq 1 \) for all \( i \in \{1, 2, \ldots, N\} \). Without loss of generality, we assume only one peer has this block at the beginning of time slot 0. Afterwards, participating peers with this block cooperatively disseminate it to other peers subject to their upload capacities. We are particularly interested in the following question: What is the minimum number of time slots it takes for this block to reach all \( N \) peers in the system? We observe that such a block dissemination process can be well modeled by a branching process. More specifically, let \( Z_n \) denote the number of peers that have this block at the beginning of time slot \( n \). We know that \( \{Z_n, n = 0, 1, \ldots\} \) is a branching process, with \( Z_0 = 1 \) and \( E[Z_1] = 1 + u_p \). We define a delay function \( D(k) \) as the minimum number of time slots it takes for at least \( k \) peers to receive this block. That is, \( D(k) = \min\{n : Z_n \geq k\} \), for \( 1 \leq k \leq N \). We are interested in the asymptotic behavior of \( D(k) \), when \( k \) tends to infinity.

**Lemma 2.2** Let \( D(k) = \min\{n : Z_n \geq k\} \) be the minimum delay for at least \( k \) peers to receive a particular block in the synchronous time model. Then it holds almost surely that

\[
D(k) \approx \lceil \log_m(k) \rceil,
\]

when \( k \) tends to infinity, where \( m = E[Z_1] = 1 + u_p \).

**Proof:** Let \( W_n = Z_n/m^n, n = \{0, 1, \ldots\} \). Kesten and Stigum [12] proved that if \( m > 1 \) and \( E[Z_1 \log Z_1] < \infty \), then the random variables \( \{W_n\} \) converge almost surely to a random variable \( W \), for which

\[
E[W] = 1, \ Var[W] = \frac{Var[Z_1]}{m^2 - m}.
\]
2.2. FUNDAMENTAL CONSTRAINTS

It follows that

$$\frac{Z_{D(k)}}{m^{D(k)}} \rightarrow W,$$

almost surely as $k \rightarrow \infty$. Therefore,

$$\frac{W}{m} \leftarrow \frac{Z_{D(k)-1}}{m^{D(k)}} < \frac{k}{m^{D(k)}} \leq \frac{Z_{D(k)}}{m^{D(k)}} \rightarrow W.$$

In other words,

$$\log_m(k) - \log_m(W) \leq D(k) \leq \log_m(k) - \log_m(W) + 1.$$

Finally, note that $\log_m(k) \gg \log_m(W)$ with high probability when $k$ is large. We thus have

$$D(k) \approx \lceil \log_m(k) \rceil,$$

for sufficiently large $k$. □

Lemma 2.2 suggests that it takes at least $\lceil \log_m(N) \rceil$ time slots for a particular block to reach all $N$ peers in the system. Intuitively, although $D(N)$ is a random variable due to inherent randomness in the branching process, when the population $N$ is very large, the law of large numbers effect controls the growth of $D(N)$, so that $D(N)$ converges to $\lceil \log_m(N) \rceil$ as $N$ increases.

The asynchronous time model. The special case of homogeneous upload capacity with $u_p = 1$ has been discussed in [10]. It is quite straightforward to extend this to the general homogeneous case where $u_i = u_p \geq 1$ for all $i \in \{1, 2, \ldots, N\}$. Similarly, we assume only one peer has this block at time 0. Let $Z(t)$ denote the number of peers that have this block at time $t$. We know that $\{Z(t), t \geq 0\}$ is a pure birth process
2.2. FUNDAMENTAL CONSTRAINTS

[13] with birth rate \( \lambda_n = u_p n \) and \( Z(0) = 1 \). The delay function \( D^*(k) \) defined as \( D^*(k) = \min\{t : Z(t) \geq k\} \) has the following properties.

**Lemma 2.3** Let \( D^*(k) \) be the minimum delay for at least \( k \) peers to receive a particular block in the asynchronous time model. Then

\[
E[D^*(k)] < \frac{1 + \ln(k)}{u_p}, \text{ and } \text{Var}(D^*(k)) < \frac{2}{u_p^2}.
\]

**Proof:** Note that \( D^*(k) \) can be represented as \( D^*(k) = \sum_{i=1}^{k-1} T_i \), where \( T_i \) is the time to go from \( i \) peers receiving the particular block to \( i + 1 \) peers. As the \( T_i \) are independent exponential random variables with respective rates \( \lambda_i = iu_p \), we see that

\[
E[D^*(k)] = \frac{1}{u_p} \sum_{i=1}^{k-1} \frac{1}{i}
\]

and

\[
\text{Var}(D^*(k)) = \frac{1}{u_p^2} \sum_{i=1}^{k-1} \frac{1}{i^2}.
\]

Therefore, we have

\[
E[D^*(k)] = \frac{1}{u_p} \sum_{i=1}^{k-1} \frac{1}{i} < \frac{1}{u_p} \left( 1 + \int_{i=1}^{k} \frac{1}{t} dt \right) = \frac{1 + \ln(k)}{u_p}
\]

and

\[
\text{Var}(D^*(k)) = \frac{1}{u_p^2} \sum_{i=1}^{k-1} \frac{1}{i^2} < \frac{2}{u_p^2}.
\]

\( \Box \)

Lemma 2.3 suggests that the minimum delay in the asynchronous time model is much shorter than the synchronous time model. For example, if we set the relative peer capacity
2.2. FUNDAMENTAL CONSTRAINTS

\( u_p = 1 \), then the minimum delay in the asynchronous time model is only \( \ln 2 = 69.3\% \) of the synchronous time model.
Chapter 3

Pull-based Streaming Protocols

In this chapter, we study the performance of pull-based streaming protocols, with a particular focus on the performance gap between the fundamental limits and the actual performance. Our results suggest that periodic buffer-map exchanges are the major problem in pull-based protocols, which account for most of the performance gap. For mathematical tractability, we use a complete graph to represent the overlay topology in our analysis, and apply the synchronous time model presented in Chapter 2.

3.1 A Study of Fundamental Limits

In this section, we proceed to the fundamental limits of pull-based mesh protocols with regard to several important performance metrics. First, an initial buffering delay must be experienced by a peer when it first joins or switches to a new channel. How do we improve user experience with the shortest initial buffering delay? Second, if media blocks do not arrive in a timely fashion, they have to be skipped at playback. How do we consistently sustain a high streaming rate with as few playback skips as possible?
3.1. A STUDY OF FUNDAMENTAL LIMITS

We believe these two performance metrics should be given priority when evaluating a pull-based mesh protocol, as they matter most to the user satisfaction. We derive several fundamental limits on the initial buffering delay and sustainable streaming rate. In particular, we focus on the flash crowd scenario where most of the peers join the system at approximately the same time, just after a new live event has been released. We note that typically, in steady state, it is quite possible to maintain a high streaming rate with short initial buffering delays [31]. Hence, we have mainly focused on the flash crowd scenario as it exercises the streaming systems the most. Our theoretical framework may also be extended to other dynamic scenarios of interest, such as peer churning.

The first question we are interested in is that: What is the minimum initial buffering delay that should be experienced by a flash crowd of \(N\) peers? Here we give a lower bound on the minimum initial buffering delay. For simplicity, we assume that the relative server capacity \(u_s\) is an integer number. This is a reasonable assumption, as the total upload capacities of commercial streaming servers are typically much larger than the streaming rate [34] so that the round error can be ignored.

**Lemma 3.1** Let \(D_{\text{min}}\) denote the minimum initial buffering delay that should be experienced by a flash crowd of \(N\) peers, for the given relative server capacity \(u_s\) and relative peer capacity \(u_p\). Then

\[
D_{\text{min}} \geq \lceil \log_m (N/u_s) \rceil + 1,
\]

where \(m = 1 + u_p\).

**Proof:** Lemma 2.2 suggests that it takes at least \(\lceil \log_m N \rceil\) time slots for a particular block to reach all \(N\) peers, after only one copy of this block arrives to the system. Recall that the streaming server can upload \(u_s\) copies of a block during each time slot. It is
straightforward to verify that it takes at least \( \lceil \log_m(N/u_s) \rceil \) time slots for a particular block to reach all \( N \) peers, after \( u_s \) copies of this block arrive to the system. Note that one additional time slot is needed for the streaming server to upload \( u_s \) copies of this block. Therefore, a lower bound for the minimum initial buffering delay \( D_{\text{min}} \) is given by

\[
D_{\text{min}} \geq \lceil \log_m(N/u_s) \rceil + 1.
\]

It has been shown in [10] that this lower bound can be achieved by a centralized snow-ball algorithm in the special homogeneous case of \( u_p = 1 \). We will show later that this lower bound can also be achieved for the general homogeneous case of \( u_p \geq 1 \). To this end, we introduce the concept of \textit{trellis graphs} and develop a centralized graph labeling algorithm that achieves this lower bound.

**Trellis Graphs and the Graph Labeling Algorithm**

The concept of trellis graphs has been originally proposed in the network coding literature [6]. We now introduce it in the context of P2P streaming systems. Given a streaming system, the associated trellis graph is defined as follows. For each peer \( p \) in the system and \( t \in \{0, 1, \ldots\} \), the trellis graph includes a node \( p_t \), which corresponds to the associated peer \( p \) at the beginning of time slot \( t \). Similarly, for the streaming server \( S \) and \( t \in \{0, 1, \ldots\} \), the trellis graph includes a node \( S_t \). Fig. 3.1 illustrates an example of the trellis graph for a streaming system with 6 peers. The trellis graph can be treated as a detailed description of the original streaming system that allows us to conveniently design and analyze streaming schemes for the system.

A label on the node \( p_t \) in the trellis graph represents the newest block in the playback buffer on peer \( p \) at the beginning of time slot \( t \). When peer \( p \) has a chance to serve others during time slot \( t \), it would send the block corresponding to the label on the node \( p_t \). In other words, peers always select the newest block in the buffer to serve others in our
3.1. A STUDY OF FUNDAMENTAL LIMITS

The streaming server employs the same strategy: it always sends the newest block to participating peers subject to its upload capacity.

The basic idea behind our graph labeling algorithm is simple. The minimum delay bound can be achieved, if each node in the trellis graph is appropriately labeled according to certain patterns. We are particularly interested in such labeling patterns that achieve the minimum delay bound.

To this end, we introduce the following notations. Let \( m_i(t) \) denote the number of label \( i \) on the nodes in the trellis graph at the beginning of time slot \( t \). Denote by \( S_i(t) \) the set of peers that have label \( i \) at the beginning of time slot \( t \). Clearly, we have \( |S_i(t)| = m_i(t) \). For ease of presentation, denote by \( \text{binornd}(M, p) \) a binomial random number with parameters \( M \) and \( p \). Algorithm 1 specifies the evolution patterns for \( \{m_i(t)\} \) and \( \{S_i(t)\} \) that achieve the minimum delay bound.

To illustrate how to achieve the minimum delay bound using our graph labeling algorithm, we provide the following example.

**Example:** We consider a streaming system with \( N = 6 \) peers. We illustrate the associated trellis graph in Fig. 3.1. We set \( u_s = u_p = 1 \) in this example. That is, both the streaming server and the participating peer can upload only one block during each time slot. It is easily verified that the minimum delay bound equals to 4 time slots in this example.

Initially, we have \( m_1(1) = 1 \) and \( S_1(1) = \{1\} \). Hence, we give label 1 to peer 1 at the beginning of time slot 1. In the first iteration \( (t = 1) \), we have \( m_1(2) = 2 \) and \( m_2(2) = 1 \). We choose \( S_1(2) = \{1, 2\} \) and \( S_2(2) = \{3\} \). Clearly, \( S_1(2) \) and \( S_2(2) \) are disjoint and \( S_1(1) \subseteq S_1(2) \). Thus, we give label 1 to peer 2 and label 2 to peer 3 at the beginning of time slot 2. We point out that such labeling choice is not unique in Algorithm 1. Now we
Algorithm 1 Graph Labeling Block Scheduling Algorithm

1. Compute the minimum delay bound $D_{\text{min}}$.
   Here, $D_{\text{min}} = \lceil \log_m(N/u_s) \rceil + 1$.
2. Set current time slot $t = 1$.
3. Set $p = u_p - \lfloor u_p \rfloor$.
4. Initialize $m_i(1)$ and $S_i(1)$. Set
   $$m_i(1) = \begin{cases} u_s, & \text{if } i = 1, \\ 0, & \text{otherwise}, \end{cases}$$
   $$S_i(1) = \begin{cases} \{1, 2, \ldots, u_s\}, & \text{if } i = 1, \\ \emptyset, & \text{otherwise}. \end{cases}$$
5. while current time slot $t < \text{the maximum time slot}$ do
6.   Set $s = t - D_{\text{min}} + 2$.
7.   Compute $m_i(t+1)$ for each $i$ as follows:
8.   if $i = t + 1$ then
9.     $m_i(t+1) = u_s$.
10.  else if $\max\{s + 1, 0\} < i < t + 1$ then
11.    $m_i(t+1) = (1 + \lfloor u_p \rfloor)m_i(t) + \text{binornd}(m_i(t), p)$.
12.  else if $i = s + 1 > 0$ then
13.    $m_i(t+1) = \min\{g(t+1), (1 + \lfloor u_p \rfloor)m_i(t) + \text{binornd}(m_i(t), p)\}$,
14.    where $g(t+1) = N - \sum_{j=s+2}^{t+1} m_j(t+1)$.
15.  else
16.    $m_i(t+1) = 0$.
17.  end if
18.  Label the nodes in the trellis graph at the beginning of time slot $t + 1$ according to $m_i(t+1)$ such that $\{S_i(t + 1)\}$ are pairwise disjoint and $S_i(t) \subseteq S_i(t + 1)$ if $m_i(t) \leq m_i(t + 1)$.
19.  Schedule the block transmissions during time slot $t$ according to $\{S_i(t)\}$ and $\{S_i(t+1)\}$.
20.  Set $t = t + 1$.
21. end while
3.1. A STUDY OF FUNDAMENTAL LIMITS

We set \( N = 6 \) and \( u_s = u_p = 1 \) in this example. That is, both the streaming server and the participating peer can upload only one block during each time slot. The minimum delay bound in this example is 4 time slots. This example shows that each block can be disseminated to all participating peers within the minimum delay bound (4 time slots) after it is injected by the streaming server.

are ready to schedule the block transmissions during time slot 1 according to the labels. As shown in Fig. 3.1, the streaming server uploads block 2 to peer 3 and peer 1 uploads block 1 to peer 2.

In the second iteration \((t = 2)\), we have \( m_1(3) = 3, m_2(3) = 2, \) and \( m_3(3) = 1 \). We choose \( S_1(3) = \{1, 2, 6\}, S_2(3) = \{3, 4\}, \) and \( S_3(3) = \{5\} \). Note that this choice satisfies the requirement in Line 17 of Algorithm 1. We label the nodes based on \( \{S_i(3)\} \) at the beginning of time slot 3 and schedule the block transmissions during time slot 2 (see Fig. 3.1 for details). We repeat the iteration process in Algorithm 1, which results in a trellis graph with a block scheduling scheme as shown in Fig. 3.1.

Now we verify that the minimum delay bound has been achieved with no playback skips in this example. Therefore, we focus on the downloading process on each participating peer. Table 3.1 shows the downloading blocks on each participating peer during
Table 3.1: Downloading blocks on Each Participating Peer

<table>
<thead>
<tr>
<th>Time Slot</th>
<th>Participating Peers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 3 4 5 6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1 2</td>
</tr>
<tr>
<td>2</td>
<td>2 1,3 1</td>
</tr>
<tr>
<td>3</td>
<td>2,4 2 1 1</td>
</tr>
<tr>
<td>4</td>
<td>4 3,5 3 2</td>
</tr>
<tr>
<td>5</td>
<td>3 3 5 4,6 4</td>
</tr>
</tbody>
</table>

each time slot. It is easily verified that each block can be disseminated to all participating peers within the minimum delay bound (4 time slots) after it is injected by the streaming server.

The correctness of our graph labeling block scheduling algorithm is shown by the following theorem.

**Theorem 3.1** If the relative peer capacity $u_p$ is an integer number, the minimum delay bound can be achieved with no playback skips. Otherwise, the minimum delay bound can be achieved with some playback skips.

**Proof:** We begin with the case that the relative peer capacity $u_p$ is an integer. We first show that the minimum delay bound can be achieved with no playback skips if Algorithm 1 is feasible, and then prove the feasibility of Algorithm 1.

The correctness of Algorithm 1 is guaranteed by the explicit construction of $m_i(t)$ and $S_i(t)$. It is easily verified that each block can be disseminated to all participating peers within the minimum delay bound if it follows the evolution patterns of $m_i(t)$ and $S_i(t)$. To show the feasibility of Algorithm 1, we consider two cases based on the result of executing Line 13 of Algorithm 1.

**Case 1:** $m_{s+1}(t + 1) = (1 + u_p)m_{s+1}(t)$. 
In this case, it is easily verified that \( m_i(t) \leq m_i(t + 1) \) for \( i \in \{ s + 1, s + 2, \ldots, t + 1 \} \).

Thus we need to ensure that the sets \( \{ S_i(t + 1) \} \) are pairwise disjoint and \( S_i(t) \subseteq S_i(t + 1) \) for \( i \in \{ s + 1, s + 2, \ldots, t + 1 \} \). Note that \( |\bigcup_{j=s+1}^{t+1} S_j(t + 1) - \bigcup_{j=s+1}^{t} S_j(t)| \leq m_s(t) \).

Therefore it is possible to satisfy the requirement in Line 17 of Algorithm 1 by allocating the set \( \bigcup_{j=s+1}^{t+1} S_j(t + 1) - \bigcup_{j=s+1}^{t} S_j(t) \) from the set \( S_s(t) \), which results in a feasible labeling operation.

The block scheduling procedure in Line 18 of Algorithm 1 is quite straightforward. The peers associated with the nodes \( S_i(t) \) send block \( i \) during time slot \( t \) to the peers associated with the nodes \( S_i(t + 1) - S_i(t) \).

**Case 2:** \( m_{s+1}(t + 1) = g(t + 1) \).

In this case, it is easily verified that \( m_i(t) \leq m_i(t + 1) \) for \( i \in \{ s + 2, s + 3, \ldots, t + 1 \} \).

Thus we first need to ensure that the sets \( \{ S_i(t + 1) \} \) are pairwise disjoint and \( S_i(t) \subseteq S_i(t + 1) \) for \( i \in \{ s + 2, s + 3, \ldots, t + 1 \} \). Note that \( |\bigcup_{j=s+2}^{t+1} S_j(t + 1) - \bigcup_{j=s+2}^{t} S_j(t)| < m_s(t) + m_{s+1}(t) \) and \( \sum_{j=s+1}^{t+1} m_j(t + 1) \leq N \). Therefore it is feasible to ensure above conditions by allocating the set \( \bigcup_{j=s+2}^{t+1} S_j(t + 1) - \bigcup_{j=s+2}^{t} S_j(t) \) from the set \( S_s(t) \cup S_{s+1}(t) \).

Next we need to allocate the set \( S_{s+1}(t + 1) - S_{s+1}(t) \). If \( m_{s+1}(t) \leq m_{s+1}(t + 1) \), then the allocating scheme is exactly the same as that in Case 1, since we have \( |\bigcup_{j=s+1}^{t+1} S_j(t + 1) - \bigcup_{j=s+1}^{t} S_j(t)| \leq m_s(t) \). Otherwise, we could allocate the set \( S_{s+1}(t + 1) \) directly from the set \( S_{s+1}(t) \).

The block scheduling procedure is also quite straightforward. The peers associated with the nodes \( S_i(t) \) send block \( i \) during time slot \( t \) to the peers associated with the nodes \( S_i(t + 1) - S_i(t) \), for \( i \in \{ s + 2, s + 3, \ldots, t + 1 \} \). The peers associated with the nodes \( S_{s+1}(t) \) send block \( s + 1 \) during time slot \( t \) to a random subset of the peers that do not have this block subject to the upload capacities.
Combining Case 1 and Case 2, we have proved the feasibility of Algorithm 1. Thus we complete the proof for the case that \( u_p \) is an integer.

Next we turn to the case that the relative peer capacity \( u_p \) is not an integer. The proof of feasibility in this case is exactly the same as the proof in Case 1. We only need to show that the minimum delay bound can be achieved with few playback skips under the assumption that Algorithm 1 is feasible. To this end, we use playback skip rate as the metric of playback skips, which is defined as the average number of peers that have skipped a block divided by the total number of participating peers. Similarly, we consider the following two cases based on the result of executing Line 13 of Algorithm 1.

**Case 1:** \( m_{s+1}(t+1) = (1 + \lfloor u_p \rfloor)m_i(t) + \text{binornd}(m_i(t), p) \).

In this case, the corresponding block dissemination process can be well modeled by a branching process. It is therefore straightforward to calculate the playback skip rate for this case.

**Case 2:** \( m_{s+1}(t+1) = g(t+1) \).

In this case, the corresponding block dissemination process is a branching process except for the last stage, which is coupled with the dissemination processes of blocks \( \{s + 2, s + 3, \ldots, t + 1\} \). We shall use numerical methods to calculate the playback skip rate for this case. We present our numerical examples in Table 3.2 and Table 3.3. We vary the relative peer capacity \( u_p \) and the relative server capacity \( u_s \) in these numerical examples. The results show that the playback skip rate is typically less than 4%, and sometimes less than 0.2%.

Theorem 3.1 generalizes the minimum delay bound in [10] by allowing the relative peer capacity \( u_p \) to be a fraction. This relaxation enables us to derive tighter bounds on
Table 3.2: Numerical examples of playback skip rates. We set the relative server capacity $u_s$ to 2. The number of peers in the system is set to 10000. We vary the relative peer capacity $u_p$ from 1.1 to 1.5.

<table>
<thead>
<tr>
<th>capacity</th>
<th>1.1</th>
<th>1.2</th>
<th>1.3</th>
<th>1.4</th>
<th>1.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip rate</td>
<td>1.35%</td>
<td>1.83%</td>
<td>0.12%</td>
<td>3.97%</td>
<td>0.23%</td>
</tr>
</tbody>
</table>

Table 3.3: Numerical examples of playback skip rates. We set the relative peer capacity $u_p$ to 1.1. The number of peers in the system is set to 10000. We vary the relative server capacity $u_s$ from 2 to 10.

<table>
<thead>
<tr>
<th>capacity</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip rate</td>
<td>1.35%</td>
<td>0.3%</td>
<td>3.02%</td>
<td>0.13%</td>
<td>0.35%</td>
</tr>
</tbody>
</table>

the maximum sustainable streaming rate, as shown by the following theorem.

**Theorem 3.2** Let $R_{\text{max}}$ denote the maximum sustainable streaming rate for a streaming system with given server capacity $U_s$ and peer capacity $U_p$, then

$$R_{\text{max}} = \begin{cases} U_p & \text{if } N \leq 2^{B-1} \frac{U_s}{U_p}, \\ R^* & \text{otherwise}, \end{cases} \quad (3.1)$$

where $B$ is the number of blocks in the playback buffer and $R^*$ is the maximum $R$ such that

$$(1 + \frac{U_p}{R})^{B-1} \frac{U_s}{R} \geq N.$$ .

**Proof:** Theorem 3.1 states that it takes $D_{\text{min}}$ time slots for a particular block to arrive at almost all the peers in the system. Therefore, the buffer size $B$ should be no less than the
initial buffering delay $D_{\text{min}}$. In other words, we need to enforce the following condition:

$$\log_m(N/u_s) + 1 \leq B.$$  \hspace{1cm} (3.2)

After simple algebraic manipulations, we obtain an equivalent condition as follows:

$$(1 + \frac{U_p}{R})^{B-1} \frac{U_s}{R} \geq N.$$  \hspace{1cm} (3.3)

If we set the streaming rate $R$ to its maximum value $U_p$, then condition (3.3) reduces to

$$2^{B-1} \frac{U_s}{U_p} \geq N.$$

□

The term $2^{B-1} \frac{U_s}{U_p}$ reflects the scalability of the streaming system during a flash crowd. It suggests that a streaming system can accommodate a flash crowd of scale less than $2^{B-1} \frac{U_s}{U_p}$ with maximum streaming rate. Hence, the most effective approach to improve the scalability of a streaming system is to increase the buffer size on participating peers.

Previous performance bounds on the sustainable streaming rate (e.g., [29]) have not taken into account the impact of buffer size. In contrast, Theorem 3.2 characterizes how a limited buffer size affects the sustainable streaming rate, as well as the scalability of the streaming system during a flash crowd. We believe this will shed new insight on the design of playback buffers for practical streaming systems.
3.2 Understanding the Performance Gap

In this section, we identify several important factors that separate the actual and optimal performance of pull-based mesh protocols, and mathematically quantify their effects on the initial buffering delay and sustainable streaming rate. The essential features of pull-based mesh protocols include periodic buffer-map exchanges and lack of centralized scheduling. On the one hand, these features contribute most to the simplicity of pull-based mesh protocols, but on the other hand, they lead to a significant performance gap between the fundamental limits and the actual performance of pull-based mesh protocols.

3.2.1 Effect of Periodic buffer-map Exchanges

In practical streaming systems, the buffer maps are periodically exchanged so as to maintain an acceptable level of overhead. We are interested in how periodic buffer-map exchanges affect the system performance in terms of the initial buffering delay and sustainable streaming rate. Without loss of generality, we assume buffer maps are exchanged every $T$ time slots.

**Theorem 3.3** Assume that buffer maps are exchanged every $T$ time slots. Let $D_{\text{min}}(T)$ denote the corresponding minimum initial buffering delay that should be experienced by a flash crowd of $N$ peers, for the given relative server capacity $u_s$ and relative peer capacity $u_p$. Then

$$D_{\text{min}}(T) = T\lceil \log(1+u_pT)(N/u_s) \rceil + T.$$ 

**Proof:** First of all, we sketch the main idea of the proof. Again, using trellis graph techniques, we will show that the minimum initial buffering delay $D_{\text{min}}(T)$ can be achieved by a centralized scheduling algorithm—the generalized graph labeling algorithm.
3.2. UNDERSTANDING THE PERFORMANCE GAP

Note that the only constraint imposed by periodic buffer-map exchanges is that: a new block on a peer cannot be uploaded until the peer has a chance to exchange buffer maps with its neighbors. Therefore, we need to schedule the block transmissions every $T$ time slots, corresponding to the period of buffer-map exchanges. We hence group every continuous $T$ blocks into a block group. Without loss of generality, we call the $i$th block group as the blocks $\{(i-1)T + 1, (i-1)T + 2, \ldots, iT\}$.

In the first stage of our generalized graph labeling algorithm, we schedule the transmissions of block groups rather than individual blocks. The output of the first stage of the algorithm is a trellis graph with labels representing the newest block group on each node.

In the second stage of our generalized graph labeling algorithm, we further schedule the transmissions of individual blocks within each block group based on the trellis graph obtained in the first stage of the algorithm. The final output of our algorithm is a trellis graph with labels representing individual blocks being uploaded. Note that a node in the final trellis graph may have multiple labels, as the associated peer may upload several different blocks in $T$ time slots.

Now we formally describe the first stage of our generalized graph labeling algorithm in Algorithm 2 for the case that $u_p$ is an integer. It is straightforward to extend to the case that $u_p$ is not an integer.

Using the same proof technique as that for Theorem 3.1, we can show that if the relative peer capacity $u_p$ is an integer, each peer would receive $T$ blocks in a given block group within the minimum initial buffering delay after it is injected by the streaming server. Otherwise, most of the peers would receive $T$ blocks in the given block group.
3.2. UNDERSTANDING THE PERFORMANCE GAP

Algorithm 2 First Stage of the Generalized Graph Labeling Algorithm

1. Compute the minimum delay bound $D_{\text{min}}$ (in time units).
   Here, $D_{\text{min}} = \lceil \log_{1+u_pT}(N/u_s) \rceil + 1$.
2. Set current time unit $k = 1$.
3. Initialize $M_i(1)$ and $S_i(1)$. Set
   \[
   M_i(1) = \begin{cases} 
   u_s T, & \text{if } i = 1, \\
   0, & \text{otherwise}, 
   \end{cases}
   \]
   \[
   S_i(1) = \begin{cases} 
   \{1, 2, \ldots, u_s T\}, & \text{if } i = 1, \\
   \emptyset, & \text{otherwise}. 
   \end{cases}
   \]
4. while current time unit $k <$ the maximum time unit do
5. Set $s = k - D_{\text{min}} + 2$.
6. Compute $M_i(k + 1)$ as follows: $M_i(k + 1) =$
   \[
   \begin{cases} 
   u_s T, & \text{if } i = k + 1, \\
   (1 + u_pT)M_i(k), & \text{if } \max\{s + 1, 0\} < i < k + 1, \\
   \min\{g(t + 1), \\
   (1 + u_pT)M_i(k)\}, & \text{if } i = s + 1 > 0, \\
   0, & \text{otherwise}, 
   \end{cases}
   \]
   where $g(t + 1) = N - \sum_{j=s+2}^{t+1} M_j(k + 1)$.
7. Label the nodes in the trellis graph at the beginning of time unit $k + 1$ according to $M_i(k + 1)$ such that $\{S_i(k + 1)\}$ are pairwise disjoint and $S_i(k) \subseteq S_i(k + 1)$ if $M_i(k) \leq M_i(k + 1)$.
8. Schedule the group transmissions during time unit $k$ according to $\{S_i(k)\}$ and $\{S_i(k + 1)\}$.
9. Set $k = k + 1$.
10. end while
Next let us turn to the second stage of our generalized graph labeling algorithm—
scheduling the transmissions of individual blocks within each block group. Again for
ease of presentation, we formally describe it for the case that $u_p$ is an integer. It is
straightforward to extend to the case that $u_p$ is not an integer. The basic idea behind
the second stage of our algorithm is to uniformly distributed each block within a given
block group, as shown in Algorithm 3.

**Algorithm 3 Second Stage of the Generalized Graph Labeling Algorithm**

1. Set current time unit $k = 1$.
2. **while** current time unit $k <$ the maximum time unit **do**
3. \[ s = k - D_{\text{min}} + 2. \]
4. Schedule the transmissions of individual blocks for block group $i$ as follows:
5. **if** $i = t + 1$ **then**
6. Divide the set $S_i(k + 1)$ into $T$ pairwise disjoint sets with the same order of $u_s$.
   Establish a bijective map from these $T$ sets to the $T$ labels \{$(i - 1)T + 1, (i - 1)T + 2, \ldots, iT$\}.
7. **else if** $\max\{s + 1, 0\} < i < t + 1$ **then**
8. Divide the set $S_i(k + 1) - S_i(k)$ into $T$ pairwise disjoint sets with the same order.
   Establish a bijective map from these $T$ sets to the $T$ labels \{$(i - 1)T + 1, (i - 1)T + 2, \ldots, iT$\}.
9. **else if** $i = s + 1 > 0$ **then**
10. Divide the set $S_i(k + 1)$ into $T + 1$ pairwise disjoint sets such that $T$ sets have
equally many elements.
   Establish a bijective map from these $T$ sets with the same order to the $T$ labels \{$(i - 1)T + 1, (i - 1)T + 2, \ldots, iT$\}.
   Let each peer associated with the remaining one set have $T$ labels \{$(i - 1)T + 1, (i - 1)T + 2, \ldots, iT$\}.
11. **end if**
12. Set $k = k + 1$.
13. **end while**

It is easily verified that in each step of Algorithm 3, the blocks within a given block
group are uniformly distributed. Note that each peer has only one copy of a block in its
buffer in Algorithm 3. Therefore, if the relative peer capacity $u_p$ is an integer, each peer
would receive all $T$ blocks in a given block group within the minimum initial buffering
3.2. UNDERSTANDING THE PERFORMANCE GAP

delay after it is injected by the streaming server. Otherwise, most of the peers would receive all $T$ blocks in the given block group. This completes the proof.

To illustrate how to schedule blocks using our generalized graph labeling algorithm, we provide the following example.

**Example:** We consider a streaming system with $N = 6$ peers. We set $T = 2$ and $u_s = u_p = 1$ in this example. That is, buffer maps are exchanged every 2 time slots and both the streaming server and the participating peer can upload only one block during each time slot. It is easily verified that the minimum initial buffering delay $D_{\text{min}}(2)$ equals to 6 time slots in this example.

We schedule the block transmissions every 2 time slots in this example, corresponding to the period of buffer-map exchanges. Hence, we change the time unit from 1 time slot to 2 time slots. The associated trellis graph is defined based on the new time unit, as shown in Fig. 3.2.

<table>
<thead>
<tr>
<th>Time Unit</th>
<th>Participating Peers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4,7</td>
</tr>
</tbody>
</table>

Fig. 3.2 shows the final output of our generalized graph labeling algorithm for this example: a trellis graph with labels on each node representing the blocks being uploaded. Table 3.4 shows the downloading blocks on each participating peer during each time unit. It is easily verified that each block can be disseminated to all participating peers within the minimum initial buffering delay (6 time slots or 3 time units) after it is injected by the streaming server.
3.2. UNDERSTANDING THE PERFORMANCE GAP

Figure 3.2: An example to illustrate block scheduling using the generalized Graph Labeling algorithm. We set the number of peers \( N \) to 6, the period of buffer-map exchanges \( T \) to 2. Both the relative server capacity \( u_s \) and the relative peer capacity \( u_p \) are set to 1. In this example, the minimum initial buffering delay is 6 time slots, and the time unit is 2 time slots. This example shows that each block can be disseminated to all participating peers within the minimum initial buffering delay (6 time slots or 3 time units) after it is injected by the streaming server.

Theorem 3.3 characterizes the performance gap in terms of the initial buffering delay due to periodic buffer-map exchanges. This is closely related to the fundamental overhead-delay tradeoff \([3, 32]\) in pull-based mesh protocols. To minimize the initial buffering delay, each peer has to exchange buffer maps in a timely fashion, resulting in an excessive overhead. On the other hand, to reduce the overhead, peers need to exchange buffer maps periodically, leading to a considerable delay.

A by-product of Theorem 3.3 is the first exact characterization of the overhead-delay tradeoff during flash crowds. As the overhead is inversely proportional to the period of buffer-map exchanges, the key step in quantifying the overhead-delay tradeoff is to investigate how the initial buffering delay increases with the period of buffer-map exchanges, which has been completely answered in Theorem 3.3. For large-scale streaming systems, Theorem 3.3 can be restated as follows.
3.2. UNDERSTANDING THE PERFORMANCE GAP

**Corollary 3.1** Assume that buffer maps are exchanged every $T$ time slots. The corresponding minimum initial buffering delay $D_{\min}(T)$ that should be experienced by a large flash crowd can be approximated as follows.

$$\frac{D_{\min}(T)}{D_{\min}} \approx \frac{T}{\log_m(1 + u_p T)},$$

where $m = 1 + u_p$ and $D_{\min}$ is the fundamental limit on the initial buffering delay.

**Proof:** From Theorem 3.3, we have

$$D_{\min}(T) = T\lceil \log_{(1+u_p T)}(\frac{N}{u_s}) \rceil + T.$$

The fundamental limit $D_{\min}$ of the initial buffering delay is given by

$$D_{\min} = \lceil \log_{(1+u_p)}(\frac{N}{u_s}) \rceil + 1.$$

Therefore,

$$\frac{D_{\min}(T)}{D_{\min}} = \frac{T\lceil \log_{(1+u_p T)}(N/u_s) \rceil + T}{\lceil \log_{(1+u_p)}(N/u_s) \rceil + 1} \approx \frac{T \log_{(1+u_p T)}(N/u_s)}{\log_{(1+u_p)}(N/u_s)} (\text{when } N \text{ is large)}$$

$$= \frac{T}{\log_m(1 + u_p T)}.$$

\[ \square \]

Corollary 3.1 suggests that the performance gap in terms of the initial buffering delay increases significantly with the period of buffer-map exchanges. However, it does not depend on the scale of the system. The periodic buffer-map exchanges also lead to a
3.2. UNDERSTANDING THE PERFORMANCE GAP

performance gap in terms of the sustainable streaming rate, as shown by the following theorem.

**Theorem 3.4** Assume that buffer maps are exchanged every $T$ time slots. Let $R_{\text{max}}(T)$ denote the corresponding maximum sustainable streaming rate for a streaming system with given server capacity $U_s$ and peer capacity $U_p$. Then

$$R_{\text{max}}(T) = \begin{cases} U_p & \text{if } N \leq (1 + T) \frac{B - 1}{U_p}, \\ R^*(T) & \text{otherwise}, \end{cases}$$

(3.4)

where $B$ is the number of blocks in the buffer and $R^*(T)$ is the maximum $R$ such that

$$(1 + T \frac{U_p}{R})^{\frac{B - 1}{R}} U_s \geq N.$$  

The proof of Theorem 3.4 is a straightforward extension of that of Theorem 3.2 and we thus omit the details here. Theorem 3.4 suggests that under the constraint of periodic buffer-map exchanges, a streaming system can accommodate a flash crowd of scale less than $(1 + T) \frac{B - 1}{U_p}$ with the maximum streaming rate. It unveils the overhead-scalability tradeoff in pull-based mesh protocols during flash crowds. This tradeoff has received little attention in the literature, as it is not as intuitive as the overhead-delay tradeoff.

3.2.2 Effect of Lack of Centralized Scheduling

In practical streaming systems, the participating peers employ simple decentralized schemes in order to maintain the simplicity. Intuitively, a simple decentralized scheme would lead
3.2. UNDERSTANDING THE PERFORMANCE GAP

to a certain degree of performance loss, compared to a sophisticated centralized streaming scheme that approaches the optimal performance (e.g., our generalized graph labeling algorithm in Sec. 3.2.1). Such performance loss is referred to as the performance gap due to lack of centralized scheduling. We are interested in characterizing this performance gap and comparing it with the performance gap caused by periodic buffer-map exchanges.

The first question is that: What pull-based streaming scheme should be analyzed in this section? We naturally prefer a simple streaming scheme with minimum performance gap to the optimal centralized scheme and minimum disruption to traditional pull-based protocols that real-world streaming protocols use. To achieve this objective, we choose to make minimum modifications to traditional pull-based protocols, based on the insights from our generalized graph labeling algorithm. Specifically, we slightly modify the block selection component of the pull-based protocol implemented in [3] and keep other components (e.g., overlay construction and peer selection) of that protocol unchanged.

The pull-based protocol in [3] employs a random block selection strategy: when a downstream peer has a chance to request a block from an upstream peer, it randomly selects a missing block in its own playback buffer. In our simple pull-based streaming scheme, we adopt a newest group first block selection strategy: when a downstream peer has a chance to request a block from an upstream peer, it randomly selects a block in the newest group on that upstream peer, which is inspired by our generalized graph labeling algorithm. Similarly, the streaming server always sends the blocks in the newest group to participating peers.

We again use trellis graph techniques to study the performance of our simple streaming scheme. For ease of presentation, we introduce the following notations. We say a peer has block group \( i \) if it has at least one block in that group. A label on each node in the
3.2. UNDERSTANDING THE PERFORMANCE GAP

The trellis graph represents the newest block group on the associated peer. Let \( m_i(k) \) denote the average number of label \( i \) on the nodes in the trellis graph at the beginning of time unit \( k \). Let \( q_i(k) \) denote the average number of peers that have block group \( i \) at the beginning of time unit \( k \). We are interested in the evolution patterns for \( \{m_i(k)\} \) and \( \{q_i(k)\} \) in the trellis graph.

**Proposition 3.1** The evolution patterns for \( \{m_i(k)\} \) and \( \{q_i(k)\} \) in the trellis graph using our simple streaming scheme can be approximated by the following difference equations.

\[
\begin{align*}
q_{k+1}(k + 1) &= u_s T \\
q_k(k + 1) &= q_k(k) + u_p T m_k(k) \left( 1 - \frac{q_k(k)}{N} \right) \\
q_{k-i}(k + 1) &= q_{k-i}(k) + u_p T m_{k-i}(k) \left( 1 - \frac{q_{k-i}(k)}{N} \right) \\
m_k(k) &= q_k(k) \\
m_{k-i}(k) &= q_{k-i}(k) \prod_{j=0}^{i-1} \left( 1 - \frac{q_{k-j}(k)}{N} \right)
\end{align*}
\]

**Proof:** Consider the system performance during time unit \( k \). Note that block group \( k + 1 \) is the newest group on the streaming server during this time unit. Thus, we have \( q_{k+1}(k + 1) = u_s T \). Let us turn to block group \( k \). Note that there are a total of \( m_k(k) \) peers with label \( k \) that would upload block group \( k \) during this time unit. If a peer without any block in group \( k \) receives such a block during time unit \( k \), then the number of peers that have block group \( i \) at the beginning of time unit \( k + 1 \) would be increased by 1. Therefore, we have \( q_k(k + 1) = q_k(k) + u_p T m_k(k) \left( 1 - \frac{q_k(k)}{N} \right) \). This argument also applies to other block groups.

Now let us turn to the number of labels at the beginning of time unit \( k \). Note that a
peer has label $k - i$ at the beginning of time unit $k$ if and only if block group $k - i$ is the newest group in its buffer. Assume that for any given peer, the event that it has block group $k - i$ is independent of the event that it has any other block group. Thus, the probability that block group $k - i$ is the newest group for a given peer can be approximated by $\prod_{j=0}^{i-1} (1 - q_{k-j}(k)/N)$. We therefore have $m_{k-i}(k) = q_{k-i}(k) \prod_{j=0}^{i-1} (1 - q_{k-j}(k)/N)$. □

From Proposition 3.1, we obtain the following result, which characterizes the evolution of peers that have a given block group.

**Corollary 3.2** For a given block group, let $f(k)$ denote the fraction of peers that have this block group $k$ time units later after it is injected by the streaming server. Then the evolution of $f(k)$ can be approximated as follows.

\[
\begin{align*}
  f(1) &= \frac{u_s T}{N} \\
  f(i + 1) &= f(i) + u_p T f(i) (1 - f(i)) \prod_{j=1}^{i-1} (1 - f(j))
\end{align*}
\]

Fig. 3.3 compares the fraction of peers that have a given block group obtained by Corollary 3.2 and by running large-scale simulations, in a number of different scenarios. We observe that our analytical approximations correctly predict the evolution behavior of the system, both qualitatively and quantitatively. As seen in Fig. 3.3, the fraction of peers that have a given block group increases almost exponentially under our newest group first strategy. This implies that our simple streaming scheme is able to approach the performance of our generalized graph labeling algorithm, in which the fraction of peers that have a given block group increases exponentially.

Now we are ready to study the performance gap due to lack of centralized scheduling and compare it with the performance gap caused by periodic buffer-map exchanges. To
achieve this objective, we compare the fundamental limits (referred to as limit) obtained in Sec. 3.1 with the performances of our generalized graph labeling algorithm (referred to as period) and our simple streaming scheme (referred to as simple). The performance difference between limit and period reflects the performance gap due to periodic buffer-map exchanges, as buffer maps are periodically exchanged in our generalized graph labeling algorithm. The performance difference between period and simple reflects the performance gap due to lack of centralized scheduling, as explained earlier.

Fig. 3.4 illustrates the comparisons of the performance gap between the fundamental limits and the actual performance of pull-based mesh protocols in terms of the initial buffering delay. From Fig. 3.4, we observe that periodic buffer-map exchanges account for most of the gap that separates the actual and optimal initial buffering delay in pull-based mesh protocols. In contrast, the lack of centralized scheduling only results in a small degree of performance loss. Moreover, we observe that the performance gap
3.2. UNDERSTANDING THE PERFORMANCE GAP

Figure 3.4: Comparisons of the performance gap in terms of the initial buffering delay. In (a), we set the relative server capacity $u_s$ to 2 and the relative peer capacity $u_p$ to 1.1. The number of peers in the system is set to 100000. We vary the period of buffer-map exchanges from 2 time slots to 10 time slots. In (b), we set the relative server capacity $u_s$ to 2 and the relative peer capacity $u_p$ to 1.1. The period of buffer-map exchanges is set to 4 time slots. We vary the number of peers in the system from 20000 to 100000.

increases significantly as the period of buffer-map exchanges increases, but is insensitive to the number of peers in the system. This observation agrees with Corollary 3.1.

We now turn to the performance gap in terms of the sustainable streaming rate. As shown in Fig. 3.5, we observe again that periodic buffer-map exchanges account for most of the performance gap, while the lack of centralized scheduling only leads to a small degree of performance loss. Furthermore, the sustainable streaming rate deteriorates significantly after the system scale exceeds a threshold and this threshold depends critically on the period of buffer-map exchanges. This confirms the overhead-scalability tradeoff in Sec. 3.2.1. Both Fig. 3.4 and Fig. 3.5 suggest that periodic buffer-map exchanges play a critical role in the actual performance of pull-based mesh protocols and should deserve special treatment in the system design. Moreover, simple pull-based streaming schemes with fine tuned system parameters are good enough to achieve high
streaming rates and short initial buffering delays, as the lack of centralized scheduling only leads to a small degree of performance loss.

![Figure 3.5: Comparisons of the performance gap in terms of the sustainable streaming rate. We set the server upload capacity $U_s$ to be 2 times of the average peer upload capacity $U_p$. We vary the number of peers in the system from 12500 to 800000.](image)

### 3.3 Extensions to Heterogeneous Cases

In previous sections, we mainly focus on the homogeneous case where participating peers have the same upload capacity. However, in real-world P2P streaming systems, participating peers have different types of network access, therefore, different upload capacities. In this section, we study the impact of heterogeneous upload capacities on the performance gap by considering several typical cases. We will show that our main conclusions still hold in these typical heterogeneous cases.

To characterize the heterogeneity of peer upload capacities, we introduce three types of peers: super peers, ordinary peers, and free-riders. Super peers correspond to the high bandwidth Ethernet users, while ordinary peers correspond to the low bandwidth DSL
users. We further observe that some users are not able to efficiently contribute their upload capacities due to connectivity restrictions posed by NATs and firewalls [30, 31], which are referred to as free-riders.

For simplicity, we begin with the case where the streaming systems consist of only ordinary peers and free-riders. We shall show that the heterogeneity of peer upload capacities does not affect our main conclusions in this special case. We then extend to the general case where the streaming systems include all three types of peers—super peers, ordinary peers, and free-riders.

### 3.3.1 Case 1: Ordinary Peers and Free-Riders

We introduce the following notations here. Consider a streaming system with $N$ participating peers. Suppose there are $N_o$ ordinary peers in the system with the same upload capacity $U_o$. All the remaining peers are free-riders and do not contribute their upload capacities. For a given streaming rate $R$, the relative ordinary peer capacity is defined as the ratio of the upload capacity $U_o$ to the streaming rate $R$. We first extend the minimum delay bound in Lemma 3.1.

**Lemma 3.2** Let $D_{\text{min}}$ denote the minimum delay bound in this case for a flash crowd of $N_o$ ordinary peers and $N_f$ free-riders, with the given relative server capacity $u_s$ and relative ordinary peer capacity $u_o$. Then

$$D_{\text{min}} \geq \lceil \log_m (N/u_s) \rceil + 1,$$

where $m = 1 + u_o$. 


Proof: In the original streaming system, we have $N_o$ ordinary peers and $N_f$ free-riders, where $N_o + N_f = N$. Now we consider a new streaming system with $N$ ordinary peers and no free-riders. By Lemma 3.1, the minimum delay bound for the new streaming system is given by $\lceil \log_m (N/u_s) \rceil + 1$, where $m = 1 + u_o$. Clearly, this also serves as a lower bound for the original streaming system.

This minimum delay bound turns out to be achievable in this heterogeneous case, as shown in the following theorem.

**Theorem 3.5** If the relative capacity of ordinary peers $u_o$ is an integer number, the minimum delay bound can be achieved with no playback skips. Otherwise, the minimum delay bound can be achieved with some playback skips.

Proof: Here we provide a heuristic proof by considering a new streaming system with $N$ ordinary peers and no free-riders. A formal proof of this theorem is straightforward to obtain by using trellis graph techniques to make this argument rigorous.

First of all, we notice that the minimum delay bound can be achieved by using our graph labeling algorithm for the new streaming system with $N$ ordinary peers. Furthermore, we observe that only a subset of participating peers are contributing their upload capacities because the streaming rate is much smaller than the upload capacity of ordinary peers. If the number of such subset of peers is no greater than $N_o$ (the number of ordinary peers in the original streaming system), it is possible to restrict such subset in the new streaming system to these $N_o$ ordinary peers in the original system. Therefore, we have established a mapping from the new system to the original system.

Now we show that the number of such subset of peers is indeed no greater than $N_o$. By Lemma 2.1, we know the maximum streaming rate $R_{\text{max}} \leq \frac{N_o}{N} U_o$. In other words, the
relative capacity of ordinary peers $u_o \geq \frac{N}{N_o}$. Therefore, a fraction of $\frac{N_o}{N}$ ordinary peers are sufficient to maintain the streaming rate, which completes the proof.

Using the same argument, we can also take the periodic buffer-map exchanges into account, which leads to the following theorem.

**Theorem 3.6** Assume that buffer maps are exchanged every $T$ time slots. Let $D_{\text{min}}(T)$ denote the corresponding minimum initial buffering delay that should be experienced by a flash crowd of $N_o$ ordinary peers and $N_f$ free-riders, for the given relative server capacity $u_s$ and relative ordinary peer capacity $u_o$. Then

$$D_{\text{min}}(T) = T \lceil \log_{(1+u_o T)}(N/u_s) \rceil + T.$$ 

We have extended Theorem 3.1 and Theorem 3.3 from the homogenous case to the heterogeneous case with ordinary peers and free-riders. Similarly, we could also extend Theorem 3.2 and Theorem 3.4 in the same way. We therefore conclude that our main conclusions still hold in this heterogeneous case.

### 3.3.2 Case 2: Super Peers, Ordinary Peers, and Free-Riders

Now we turn to the general heterogeneous case where the streaming systems include all three types of peers—super peers, ordinary peers, and free-riders. Suppose there are $N_{sp}$ super peers, $N_o$ ordinary peers, and $N_f$ free-riders in the streaming system. Clearly, we have $N_{sp} + N_o + N_f = N$. Let $U_{sp}$ denote the upload capacity of super peers. For a given streaming rate $R$, the relative super peer capacity is defined as the ratio of the upload capacity $U_{sp}$ to the streaming rate $R$.

We observe that the upload capacity of super peers is much larger than the capacity
of ordinary peers [35]. This observation allows us to approximate the system performance in the general heterogeneous case by borrowing the results obtained in Case 1.

Imagine that the streaming server first uploads blocks to the super peers in the system, and then these super peers help ordinary peers and free-riders by contributing their upload capacities. Note that it takes little time for the super peers to serve new blocks in the system due to their larger upload capacities. In other words, the super peers are playing the same role of the streaming server in the system. Therefore, we can regard the streaming server together with the super peers as a new streaming server with the total upload capacity equaling to \( U_s + N_{sp} U_{sp} - N_{sp} R \). The term \( N_{sp} R \) is due to the fact that these super peers are also consuming bandwidth resources in order to maintain the streaming rate \( R \).

Therefore, we have established a mapping from the general heterogeneous case where the streaming systems include all three types of peers to the special heterogeneous case where the streaming systems consist of only ordinary peers and free-riders. Since the heterogeneity of peer upload capacities does not affect our main conclusions in this special case, we conclude that our main conclusions still hold in the general heterogeneous case.

3.4 Summary

The unique strength of pull-based mesh streaming protocols is the simplicity, which makes them the choice of many real-world streaming systems. The essential features of pull-based mesh protocols include periodic buffer-map exchanges and lack of centralized scheduling. These features contribute most to the simplicity of pull-based mesh protocols, but at the same time, lead to a performance gap between the fundamental limits and the actual performance.
3.4. SUMMARY

In this chapter, we have developed a unified framework based on the trellis graph technique to mathematically analyze and understand the performance of pull-based mesh protocols, with a particular focus on such a performance gap. Our analytical results show that there exists a significant performance gap between the fundamental limits and the actual performance of pull-based mesh protocols. Moreover, periodic buffer-map exchanges account for most of the gap that separates the actual and optimal performance of pull-based mesh protocols. In contrast, the lack of centralized scheduling only results in a small degree of performance loss.

Our analytical characterization of the performance gap brings us not only a better understanding of several fundamental tradeoffs in pull-based mesh protocols, but also important insights on the design of practical streaming systems. For example, we give the first exact characterization of the overhead-delay tradeoff in pull-based mesh protocols during flash crowds. We further unveil the overhead-scalability tradeoff that receives little attention in the literature. More importantly, our analytical results in this chapter motivate our design principles for network-coding-based streaming solution proposed in Chapter 4. By eliminating the need for periodic buffer-map exchanges and thus addressing the most inefficient factor in pull-based protocols, it is intuitive that our network-coding-based solution is able to significantly reduce the permanence gap and reach the fundamental limits of P2P streaming systems.
Chapter 4

New Protocols using Network Coding

In this chapter, we first present an overview of our design principles for P2P streaming systems with network coding. We then prove that any streaming protocol using our design principles is sufficient to achieve provably good performance with respect to many important metrics, such as playback quality, initial buffering delay, resilience to peer dynamics, as well as bandwidth cost on dedicated streaming servers. For mathematical tractability, we apply the asynchronous time model presented in Chapter 2. A complete graph model is used to represent the overlay topology in our analysis, and a random graph model is used in our simulations.

4.1 Design Principles with Network Coding

The first motivating factor for using network coding in a new streaming system is to improve playback quality. Intuitively, as coded blocks are received from multiple senders,
it is helpful to receive a media segment in a timely manner. It turns out that network coding can also help to simplify the protocol design, shorten initial buffering delays, and minimize server bandwidth cost by allowing more bandwidth contributions from peers. In this section, we are able to intuitively explain these benefits of network coding. For brevity, a streaming protocol that uses network coding following our design principles in this chapter is referred to as CODING.

### 4.1.1 Random Push on a Random Mesh Structure

In traditional pull-based mesh streaming strategies (henceforth referred to as PULL for brevity), the streaming content to be served is divided into blocks with a fixed number of bytes in each block, such that their availability can be better exchanged among peers (usually as a bitmap). In CODING, much smaller blocks are used, and every $m$ blocks are grouped into a big segment. Coding is performed within each segment, but not across different segments. This is primarily designed for the purpose of reducing the number of blocks to code, leading to much reduced encoding and decoding complexity.

![Figure 4.1: An example to illustrate the coding operation on peer $p$, where peer $p$ has received 3 coded blocks within the segment $s$, and each segment consists of 6 blocks.](image)

Suppose a peer $p$ has received $k$ ($k \leq m$) coded blocks within a segment $s$ so far, denoted as $[b_1, b_2, \ldots, b_k]$. When peer $p$ is to encode segment $s$ for its downstream peers,
as shown in Fig. 4.1, it independently and randomly chooses a set of coding coefficients \([c_1, c_2, \ldots, c_k]\) in a Galois field for each coded block, and then produces one coded block \(x\) in the form of \(x = \sum_{i=1}^{k} c_i \cdot b_i\). Results from randomized network coding [36] ensure that with high probability, the coded block \(x\) is useful to such downstream peers that have not completely received the segment \(s\) (refer to Sec. 4.2.4 for a detailed discussion).

By sending coded blocks instead of an entire segment, multiple upstream peers may serve a missing segment on a common downstream peer simultaneously without any explicit coordination. This excellent property of cooperative transmission is illustrated in Fig. 4.2. In this way, peers are able to perform push operations on a random mesh structure, without assuming the risk of sending duplicate segments. Intuitively, such push operations not only eliminate the need for request messages, but also lead to much shorter initial buffering delays, as demonstrated by their counterpart—push-based tree operations.

From the viewpoint of a downstream peer, after it has completely received \(m\) linearly independent coded blocks within a segment, this downstream peer can successfully decode the original segment with Gaussian or Gauss-Jordan elimination. Again, we are not concerned with the detailed design of a particular streaming protocol using network coding.

### 4.1.2 Timely Feedback from Downstream Peers

Before sending coded blocks, an upstream peer should obtain precise knowledge of the missing segments on its downstream peers at any time. This requires any peer in the system to exchange its segment availability bitmaps of streaming buffers, which are commonly referred to as buffer maps. In PULL, to avoid excessive overhead, these buffer maps
4.1. DESIGN PRINCIPLES WITH NETWORK CODING

Figure 4.2: An illustration of random push on a random mesh structure. With network coding, multiple upstream peers are able to perform push operations on coded blocks within a segment without any explicit coordination.

are exchanged periodically; the more frequent such exchanges are, the closer PULL is to a streaming protocol using the push-based tree strategy. It would be ideal to send a new buffer map whenever the buffer status changes — when it has played back a segment, or when it has completed the downloading of a segment. Due to the size of such buffer maps in PULL and the frequency of buffer status changes, such a “real-time” push strategy of buffer maps leads to excessive overhead.
With network coding, however, such a “real-time” buffer map push is a feasible strategy. Since large segments are used, not only is the size of the buffer maps an order of magnitude smaller, but there are many fewer segments in the buffer as well. This leads to less frequent buffer-status changes, as it takes much longer to finish downloading or to playback a large segment. Such a “real-time” push of buffer maps makes it possible to push coded blocks to downstream peers without any explicit requests, until the segment is fully received at downstream peers.

Why is CODING able to use large segments? We emphasize again that a missing segment on a downstream peer in CODING can be served by multiple upstream peers, while a block in PULL can only be served by one upstream peer. Thus, the process of receiving a large segment is not adversely affected by the departure of a subset of upstream peers in CODING.

### 4.1.3 Synchronized Playback and Initial Buffering Delays

The use of large segments also makes it easier to synchronize playback buffers on all participating peers, so that all peers play the same segment at approximately the same time. One advantage of such synchronized playback is that, peers can help each other
more effectively, as their playback buffers overlap as much as possible. With network coding, as large segments are used, it is important to feature synchronized playback. When a peer joins a streaming session, it first retrieves buffer maps from its neighboring peers, along with the information of the current segment being played back. The new peer then skips a few segments and only retrieves segments that are $D$ seconds after the current point of playback. The peer starts playback after precisely $D$ seconds have elapsed in real time, regardless of the current status of the playback buffer. As shown in Fig. 4.4, the duration of $D$ seconds corresponds to the initial buffering delay, which depends on the number of segments to be skipped and the current point of playback.

We naturally wish to shorten such an initial buffering delay, as it is one of the most important performance metrics that affects user experience when switching to a new channel. Suppose every peer skips $k$ segments when it first joins to a new channel. As shown in Fig. 4.4, the initial buffering delay $D$ satisfies $(k - 1)m/R \leq D \leq km/R$. If $k$ is set to be 1, then some “unlucky” peers have to start playback immediately after they join the new channel, even when they have few coded blocks in their buffer, leading to a degraded playback quality. Therefore we set $k$ to be 2 in our design principles.

We adopt a simple segment and peer selection strategy with network coding: a peer selects the most urgent segment (closest to the playback deadline) and pushes coded blocks in this particular segment to a limited number of its downstream peers. These limited number of peers are chosen uniformly at random from the downstream peers which have not fully received the particular segment. Since all upstream peers use the same strategy, a new peer who recently joined a streaming channel will naturally receive coded blocks from a large number of existing peers, saturating its downloading capacity, which leads to a shorter initial buffering delay.
4.1. DESIGN PRINCIPLES WITH NETWORK CODING

Figure 4.4: An illustration of initial buffering delays in coding, which shows that the initial buffering delay on a newly joined peer is determined by the number of segments to be skipped and the current point of playback.

### 4.1.4 System Model and Notations

Finally, we make a few assumptions in our system model. The key notations introduced in the system model are summarized in Table 4.1 for easy reference. In order to characterize the heterogeneity in terms of peer upload capacities, we adopt the two-class heterogeneous model in [9], in which peers in the system are broadly classified into two classes, with each class having approximately the same upload capacity. This assumption is reasonable as there are roughly two classes of peers in P2P streaming systems: high bandwidth Ethernet peers and low bandwidth DSL peers.

Although we assume only two classes in this chapter, our analysis can easily be extended along the same lines to accommodate more classes of peer upload capacities. The upload capacity of a class-\(i\) peer is denoted as \(U_i\) (\(i \in \{1, 2\}\)). Without loss of generality, we assume the block size is 1 and \(U_1 > U_2\).

With respect to peer dynamics, we focus on two typical scenarios: the flash crowd...
Table 4.1: Key Notations in the System Model

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(U_i)</td>
<td>Upload capacity of a class-(i) peer (in blocks per second).</td>
</tr>
<tr>
<td>(U_p)</td>
<td>Average upload capacity of participating peers.</td>
</tr>
<tr>
<td>(U_s)</td>
<td>Server upload capacity (in blocks per second).</td>
</tr>
<tr>
<td>(R)</td>
<td>Streaming rate (in blocks per second).</td>
</tr>
<tr>
<td>(D)</td>
<td>Initial buffering delay (in seconds).</td>
</tr>
<tr>
<td>(N)</td>
<td>Scale of a flash crowd.</td>
</tr>
<tr>
<td>(\alpha)</td>
<td>Fraction of redundant blocks induced by network coding.</td>
</tr>
<tr>
<td>(m)</td>
<td>Number of coded blocks in a segment.</td>
</tr>
<tr>
<td>(u_p)</td>
<td>Relative average peer capacity ((= U_p/R)).</td>
</tr>
<tr>
<td>(u_s)</td>
<td>Relative server capacity ((= U_s/R)).</td>
</tr>
<tr>
<td>(\delta)</td>
<td>Server strength ((= U_s/NU_p)).</td>
</tr>
</tbody>
</table>

scenario and highl

highly dynamic scenario. During a flash crowd, most of the peers join the system in a short time period, just after a new live event has been released. In a highly dynamic scenario, peers join and leave the system in a highly volatile fashion, also referred to as peer churn. For a better flow of presentation, we defer detailed characterizations of peer dynamics to Sec. 4.2.

4.2 Performance Analysis of CODING

Under our system model, we seek to investigate the overall performance of CODING. In particular, we provide quantitative answers to the following two questions:

▷ What is the sufficient condition for CODING to achieve good overall performance?

▷ How far from optimality is the performance of CODING?

We believe such performance analysis is crucial to understand the fundamental properties and limitations of CODING.
4.2. PERFORMANCE ANALYSIS OF CODING

4.2.1 Flash Crowd Scenarios

In a flash crowd scenario, most of the peers join the system at approximately the same time. For presentation clarity, we assume time is slotted in the sense that it takes one time slot to playback a segment. We further assume that most peers join the system within one time slot. We emphasize here that these assumptions are not necessary, and can easily be relaxed in our analysis. We now introduce the following definitions.

**Definition 4.1** The scale of a flash crowd, denoted by $N$, is defined as the maximum number of peers in the system during the flash crowd.

**Definition 4.2** The server strength, denoted by $\delta$, is defined as follows:

$$\delta = \frac{U_s}{NU_p},$$

where $U_p$ is the average upload capacity of participating peers, and $U_s$ is the server upload capacity.

The following theorem establishes a sufficient condition on smooth playback at a streaming rate $R$ during any flash crowd with scale $N$, for given server capacity $U_s$ and average peer capacity $U_p$.

**Theorem 4.1** Assume that the following condition holds:

$$U_s + NU_p \geq (1 + \varepsilon)NR,$$  \hspace{1cm} (4.1)

where $\varepsilon = \alpha + \frac{\ln(1+\delta)-\ln\delta}{m}$, $m$ is the number of coded blocks in each segment, and $\alpha$ denotes the fraction of linearly dependent coded blocks induced by network coding (refer
to Sec. 4.2.4 for details). Then CODING is able to achieve smooth playback at a streaming rate $R$ for any flash crowd with sufficiently large scale $N$.

Theorem 4.1 implies that heterogeneity in peer upload capacities is not an issue in CODING. Intuitively, this is a consequence of the random push operations, which naturally empower high bandwidth peers to contribute more bandwidth resources. For a better flow of presentation, we defer the formal proof of Theorem 4.1. We now apply Theorem 4.1 to understand the performance gap between CODING and optimal streaming scheme in terms of the sustainable streaming rate and initial buffering delay, which leads to the following two interesting theorems.

**Theorem 4.2** CODING can achieve a factor of $1 + \varepsilon$ of the maximum streaming rate $R_{\text{max}}$, where $\varepsilon$ is given by

$$
\varepsilon = \alpha + \frac{\ln(1 + \delta) - \ln \delta}{m},
$$

and $\alpha$ is typically in the order of 0.1% (refer to Sec. 4.2.4 for details).

**Proof:** By Lemma 2.1, we have

$$R_{\text{max}} = U_p + \frac{U_s}{N}.$$

By Theorem 4.1, we know that

$$R \leq \frac{U_s + NU_p}{(1 + \varepsilon)N}.$$

Combining above two results, we conclude that the sufficient condition can be rewritten as

$$R \leq \frac{R_{\text{max}}}{1 + \varepsilon}.$$

In other words, CODING can achieve a factor of $1 + \varepsilon$ of the maximum streaming rate.
Theorem 4.2 demonstrates that CODING is near-optimal in terms of sustainable streaming rate during a flash crowd. We provide a simple numerical example here. Let us set the server strength $\delta$ to 0.001, and the number of coded blocks in each segment $m$ to 100, then we can easily calculate the sustainable streaming rate in CODING, which satisfies $R \geq \frac{R_{\text{max}}}{1.07}$. This is due to the near-optimal bandwidth utilization enjoyed by CODING. During a flash crowd, a newly joined peer is able to effectively utilize its upload capacity immediately after receiving one or more coded blocks. However, in traditional pull-based protocols, such bandwidth utilization is impaired by the need of exchanging buffer maps and waiting for explicit requests.

**Theorem 4.3** CODING can achieve a factor of $2(1 + \varepsilon)$ of the minimum initial buffering delay, where $\varepsilon$ is given by

$$\varepsilon = \alpha + \frac{\ln(1 + \delta) - \ln \delta}{m}.$$ 

**Proof:** Note that in CODING, each peer buffers at least one segment ($m$ blocks) in order to maintain smooth playback during the flash crowd. This process takes more than $\frac{Nm}{Us + NU_p}$ seconds, since the maximum bandwidth supply is $Us + NU_p$. Thus the minimum initial buffering delay satisfies

$$D_{\text{min}} = (1 + \gamma) \frac{Nm}{Us + NU_p},$$

for some fixed $\gamma > 0$.

Now recall that in CODING, every new peer skips only two segments before the actual playback, thus the initial buffering delay satisfies

$$D = (1 + \beta) \frac{m}{R},$$

where $0 \leq \beta \leq 1$. 
Combining these two results with Theorem 4.1, we conclude that the sufficient condition can be rewritten as

\[ D \geq (1 + \varepsilon) \frac{1 + \beta}{1 + \gamma} D_{\text{min}}. \]

Note that

\[ \frac{1 + \beta}{1 + \gamma} < 2, \text{ as } \beta \leq 1, \gamma > 0. \]

In other words, CODING can achieve a factor of \( 2(1 + \varepsilon) \) of the minimum initial buffering delay.

Theorem 4.3 shows that CODING manages to guarantee very short initial buffering delays during a flash crowd. This is in sharp contrast to PULL, which suffers from long initial buffering delays due to inherent design limitations [3]. Theorem 4.2 and Theorem 4.3, when taken together, suggest that the performance gap between CODING and optimal streaming scheme is surprisingly small with regard to user experience.

In addition to user experience, the server bandwidth cost is also an important metric, as it directly determines most of the ongoing operational expense for streaming companies. However, it is an open problem to determine the minimum server bandwidth cost during a flash crowd for mesh-based streaming systems, with or without network coding. We therefore only characterize the required server capacity in CODING without a comparison to the optimal scheme. For convenience, we use the relative upload capacity (the ratio of upload capacity to the streaming rate) in the remaining part of this section.

Theorem 4.4 Let \( u_s \) denote the required relative server capacity that supports smooth playback at a streaming rate \( R \) during a flash crowd with scale \( N \). Then \( u_s \) satisfies
\[ u_s \geq N u_p \delta^*, \text{ where } \delta^* \text{ is given by} \]

\[ \delta^* = \min \{ \delta : (1 + \delta) u_p - 1 - \alpha \} m \geq \ln(1 + \delta) - \ln \delta \],

and \( u_p \) is the relative average peer capacity.

This theorem is a rather straightforward corollary of Theorem 4.1 and thus we omit the proof here.

Fig. 4.5 compares the required relative server capacity obtained by Theorem 4.4 and by running large-scale simulations, in a number of different flash crowd scenarios. We observe that our analytical results correctly predict the required server capacity, both qualitatively and quantitatively, especially when the scale of a flash crowd is larger than 100,000. This is because the proof of Theorem 4.4 requires the scale of a flash crowd \( N \) to be sufficiently large, which indicates the larger the scale, the better the prediction.

As seen from Fig. 4.5, the required server capacity is only about 10 times the streaming rate, in order to support a flash crowd with scale 200,000 in the scenario of \( \{u_p = 1.09, m = 100\} \) or \( \{u_p = 1.05, m = 200\} \). We also observe that the required server capacity is sensitive to the system parameters \( u_p \) and \( m \). This is again due to the near-optimal bandwidth utilization property.

So far, we have used a complete graph to represent the mesh structure in our analysis and simulations. However, a complete graph is hard to implement in practice because of overhead issues. In practical streaming systems, each peer maintains a limited number of neighbors in order to reduce the overhead of exchanging buffer maps. It is thus of great interest to investigate the impact of restricted neighborhoods.

Instead of a complete graph, we now use a random graph to model the mesh structure in our simulations. More specifically, when joining the system, each peer chooses a limited
4.2. PERFORMANCE ANALYSIS OF CODING

Figure 4.5: Validation of the required relative server capacity in several different flash crowd scenarios (theoretical results use dashed lines; simulation results use solid points). In (a), we set the number of coded blocks in a segment $m$ to 100 and vary the relative average peer capacity $u_p$ from 1.05 to 1.09; while in (b), we set $u_p$ to 1.05 and vary $m$ from 100 to 200. We can see that analytical results match simulation results quite well, especially when the system scale is larger than 100,000.
4.2. PERFORMANCE ANALYSIS OF CODING

Figure 4.6: Impact of restricted neighborhoods on the playback quality, with 95% confidence intervals. In (a), we set the relative average peer capacity $u_p$ to 1.05 and the number of coded blocks in a segment $m$ to 100; while in (b), we set $u_p$ to 1.07 and $m$ to 100. The required relative server capacities in these two cases are 49 and 98, respectively, under the assumption of complete graphs. By restricting the size of neighborhood, we observe that the playback quality is still close to 1, as long as the average size of neighborhood is larger than 50.
number of neighbors uniformly at random among all existing peers in the system. We let the average size of neighborhood vary from 20 to 100. We use continuity [28] as the metric of playback quality, which is defined as the number of peers that have successfully played the segment in each time slot divided by the total number of participating peers. We only present our simulation results of two scenarios in Fig. 4.6, although we have repeated our experiments in many other scenarios and have obtained similar results. The required relative server capacities in these two scenarios, predicted by Theorem 4.4, are 49 and 98, respectively.

Fig. 4.6 suggests that the playback quality is close to 0.98, as long as each peer maintains around 50 neighboring peers, which is a substantially small number compared to the total number of participating peers. In addition, the playback quality is close to 1 (the case of complete graphs), if the size of neighborhood is increased to 100. Moreover, we observe that such a trend is insensitive to the scale of flash crowds. All of these imply that CODING only requires a small size of neighborhood in order to achieve good performance even in large scale systems. This is not a coincidence, for our proofs of Theorem 4.1 and Theorem 4.4 do not depend heavily on the complete graph assumption, as shown in Sec. 4.2.3.

4.2.2 Highly Dynamic Scenarios

Let us turn to the highly dynamic scenario, in which peers join and leave the system in a highly volatile fashion. Without loss of generality, we only focus on the system performance in the current time slot. We first introduce some notation here. We denote by $N_i$ the number of class-$i$ peers in the system at the beginning of current time slot. To characterize peer dynamics in current time slot, we denote by $A_i$ and $W_i$ the number
of arrivals and departures of class-\(i\) peers in current time slot. These peer dynamics would clearly degrade user experience to a certain degree. However, this effect is not a major focus in this section, as most commercial streaming systems have resorted to the over-provisioning server upload capacity to handle peer dynamics. A more interesting problem is therefore to quantify such over-provisioning server upload capacity.

Denote the most urgent segment (\textit{i.e.}, the segment to be played in the next time slot) as segment \(s\). First, we observe that the arrivals of \(A_i\) new peers in current time slot do not affect the playback quality of segment \(s\), as these new peers simply skip segment \(s\) after they join the system. In contrast, the departures of \(W_i\) existing peers play a central role in the playback quality, which will be investigated in the sequel.

Recall that a class-1 peer is a high bandwidth peer, which produces more bandwidth supply than its bandwidth consumption. As a result, the earlier it leaves the system, the worse the playback quality may be. Similarly, the later a low bandwidth peer leaves the system, the worse the playback quality becomes. To summarize, the worst case in current time slot is as follows: all \(W_1\) departures of high bandwidth peers happen at the beginning of current time slot, while all \(W_2\) departures of low bandwidth peers occur at the end of current time slot. This observation leads to the following theorem.

\textbf{Theorem 4.5} \textit{The additional server capacity, which is required to handle peer dynamics in current time slot, is strictly less than} \(W_1U_1 - (1 + \varepsilon)W_1R\).

\textit{Proof:} Suppose there are no peer dynamics in the current time slot. Then by Theorem 4.1, the required server capacity \(U_s\) satisfies:

\[ U_s + N_1U_1 + N_2U_2 \geq (1 + \varepsilon)(N_1 + N_2)R. \] (4.2)
4.2. PERFORMANCE ANALYSIS OF CODING

On the other hand, to handle the worst case peer dynamics in the current time slot, the required server capacity $U'_s$ should satisfy

$$U'_s + (N_1 - W_1)U_1 + N_2U_2 \geq (1 + \varepsilon')(N_1 + N_2 - W_1)R. \tag{4.3}$$

Let $\Delta U$ denote the additional server capacity to ensure smooth playback in the worst case peer dynamics, we have

$$\Delta U = (1 + \varepsilon')(N_1 + N_2 - W_1)R - (1 + \varepsilon)(N_1 + N_2)R
+ N_1U_1 + N_2U_2 - (N_1 - W_1)U_1 - N_2U_2
= W_1U_1 + (\varepsilon' - \varepsilon)(N_1 + N_2)R - (1 + \varepsilon')W_1R
= W_1U_1 - W_1R - \varepsilon(N_1 + N_2)R + \varepsilon'(N_1 + N_2 - W_1)R$$

Notice that $\varepsilon > \varepsilon'$. It follows that

$$\varepsilon'(N_1 + N_2 - W_1)R < \varepsilon(N_1 + N_2 - W_1)R.$$ 

Therefore, we obtain

$$\Delta U < W_1U_1 - (1 + \varepsilon)W_1R.$$ 

That is, the additional server capacity is strictly less than $W_1U_1 - (1 + \varepsilon)W_1R$. \hfill \Box

Theorem 4.5 gives an upper bound for the additional server capacity to handle peer dynamics. For convenience, we restate Theorem 4.5 in terms of relative capacity. That is, the relative additional server capacity is strictly less than $W_1u_1 - (1 + \varepsilon)W_1$, where $u_1$ is the ratio of $U_1$ to the streaming rate $R$. Simulation results are shown in Table 4.2. We use a relative average peer capacity of 1.08 to represent the case where bandwidth
Table 4.2: Theoretical and simulation results for relative additional server capacity to handle peer dynamics in the worst case. We can see that the theoretical bound is tight when bandwidth supply barely exceeds bandwidth demand, while the bound is loose when supply outstrips demand. In our simulations, we set the system scale $N$ to 10,000, the ratio $u_1$ to 2, and the ratio $u_2$ to 0.5.

<table>
<thead>
<tr>
<th>$\frac{W_1}{N_1}$</th>
<th>$u_p = 1.02$</th>
<th>$u_p = 1.08$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bound</td>
<td>Simulation</td>
</tr>
<tr>
<td>0.01</td>
<td>33</td>
<td>22</td>
</tr>
<tr>
<td>0.02</td>
<td>67</td>
<td>46</td>
</tr>
<tr>
<td>0.03</td>
<td>100</td>
<td>69</td>
</tr>
<tr>
<td>0.04</td>
<td>133</td>
<td>94</td>
</tr>
<tr>
<td>0.05</td>
<td>166</td>
<td>119</td>
</tr>
</tbody>
</table>

supply outstrips demand, and a relative average peer capacity of 1.02 to represent an approximate match between the supply and demand. We observe our theoretical bound matches simulation results well when bandwidth supply barely exceeds bandwidth demand ($u_p = 1.02$), while the bound turns out to be loose when supply outstrips the demand ($u_p = 1.08$).

Intuitively, when bandwidth supply from participating peers barely exceeds the bandwidth demand, server capacity plays a central role in dealing with peer dynamics. With departures of $\gamma N(k)$ class-1 peers, we observe a net bandwidth loss, which is close to $\gamma N(k)(u_1 - 1 - \alpha)$. Such loss should be compensated by the dedicated streaming servers, as Theorem 4.5 states. In contrast, when bandwidth supply outstrips demand, the required additional server capacity can be significantly reduced. This is because the left-hand side of (4.3) is much greater than the right-hand side, leading to a quite loose theoretical bound.

After performing the worst case analysis as above, we now turn to a study of the average case, where peer departures may occur at any time in current time slot, rather than only at the beginning or at the end. We ran a set of simulation-based experiments,
varying the total number of peer departures in current time slot. We use the term *churn rate* to represent the ratio of the number of peer departures to the total number of participating peers (set to 10,000 in our simulation). The effect of churn rate is shown in Table 4.3. We observe that as churn rate increases, the additional required server capacity also increases. Moreover, CODING can survive a churn rate of 50% using an additional capacity of less than 10, which is in sharp contrast with the situation in the worst case analysis. The intuition is that the high bandwidth peers, if not leaving the system at the very beginning of current time slot, are still able to contribute more bandwidth than they consume, thereby mitigating the load on streaming servers.

Table 4.3: Simulation results for relative additional server capacity to handle peer dynamics in the average case. We can see that only a small amount of additional server capacity is required, even when 50% peers leave the system. We set the system scale $N$ to 10,000 in our simulations.

<table>
<thead>
<tr>
<th>churn rate</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>additional capacity</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>

However, when the departing peers are chosen randomly among all participating peers, not only subject to class-1 (high bandwidth) peers, the situation is completely different. As mentioned earlier, the departure of class-2 peers even slightly improves system performance. Moreover, a high bandwidth peer is very likely to supply more bandwidth than what it consumes, unless it leaves the system at the very beginning of time slot $k$. Therefore, we believe such a kind of peer dynamics has little impact on the required server capacity, as long as the average upload capacity is not significantly affected by random peer departures. Contrary to the worse case analysis in Theorem 4.5, only a small amount of additional server capacity is required in the *average case*, in which both the departing peers and departure times are chosen at random.
4.2. PERFORMANCE ANALYSIS OF CODING

4.2.3 Formal Proof of the Sufficient Condition

We now proceed to present a formal proof of Theorem 4.1, which is instrumental to establish an in-depth understanding of CODING. As mentioned earlier, we only consider the asynchronous time model in the proof. We will mainly focus on the general homogeneous case, as it is straightforward to extend this to the multiple-class heterogeneous case.

Recall that the most urgent segment is denoted as segment \( s \), which consists of \( m \) blocks. Let \( X_i(t) \) denote the number of peers holding exactly \( i \) coded blocks within segment \( s \) at time \( t \), for any \( i \in \{0,1,\ldots,m\} \). A peer holding exactly \( i \) coded blocks within segment \( s \) is also called a type-\( i \) peer.

Given the system state \( X(t) := \{X_i(t)\} \ (i \in \{0,1,\ldots,m\}) \), node \( i \) chooses one neighbor and transmits one coded block at the instants of a Poisson process of rate \( u_i \). We assume that each node, when transmitting a coded block, selects the target peer uniformly at random from all its neighbors that have not received all \( m \) coded blocks.

Since the overlay topology is a complete graph, we know that the probability that the target peer is of type \( i \) is given by \( \frac{X_i(t)}{N-X_m(t)} \). This defines a Markov process \( \{X(t)\}_{t \geq 0} \) taking its values in \( \mathbb{Z}_{m+1}^+ \). Denote by \( e_i \) a unit vector of \( \mathbb{Z}_{m+1}^+ \) with a one in position \( i \). Then the transition rates of this Markov process are given by

\[
q(X, X') = \begin{cases} 
\frac{X_i(t)}{N-X_m(t)} (u_s + u_p(N - X_0)) & \text{for } X' = X - e_{i+1} + e_{i+2}, 0 \leq i \leq m-1 \\
0 & \text{for all other choices of } X'
\end{cases}
\]

We analyze \( \{X(t)\}_{t \geq 0} \) under a large population asymptotic regime. We first show that this is a density dependent jump Markov process \([37]\) if the relative server capacity \( u_s \) scales linearly with \( N \). We then use Kurtz’s theorem \([38]\) to prove that the rescaled process \( N^{-1}X(t) \) converges almost surely to the solution of the following system of differential
4.2. PERFORMANCE ANALYSIS OF CODING

\[ \frac{dx_i(t)}{dt} = \begin{cases} 
-\frac{x_0(t)}{1-x_m(t)} \left( \frac{u_s}{N} + u_p(1 - x_0(t)) \right) & \text{for } i = 0 \\
\frac{x_{i-1}(t)-x_i(t)}{1-x_m(t)} \left( \frac{u_s}{N} + u_p(1 - x_0(t)) \right) & \text{for } 0 < i < m \\
\frac{x_{m-1}(t)}{1-x_m(t)} \left( \frac{u_s}{N} + u_p(1 - x_0(t)) \right) & \text{for } i = m 
\end{cases} \quad (4.4) \]

with initial conditions \( x_0(0) = 1 \) and \( x_i(0) = 0 \) \((1 \leq i \leq m)\) under mild conditions. For convenience, we drop the vector notation where it can be understood by context.

**Definition 4.3** Given a set of transitions \( L \subseteq \mathbb{Z}^d \) and a collection of nonnegative functions \( \beta_l \) for \( l \in L \) defined on a subset \( E \subseteq \mathbb{R}^d \), a density dependent family of Markov chains \( X_n \) is a sequence \( \{X_n\} \) of jump Markov processes such that the state space of \( X_n \) is \( E_n = E \cap \{n^{-1}k : k \in \mathbb{Z}^d\} \) and the transition rates of \( X_n \) are

\[ q^{(n)}_{x,y} = n \beta_{n(y-x)}(x), \ x, y \in E_n. \]

We rewrite the transition rates of our Markov process as follows.

\[ q^N(X, X + l_i) = N \frac{X_i/N}{1 - X_m/N} \left( u_s/N + u_p(1 - X_0/N) \right) = N \beta_{l_i}(N^{-1}X), \]

where \( l_i = e_{i+2} - e_{i+1}, \) for \( 0 \leq i \leq m - 1 \). This is precisely the definition of density dependent jump Markov processes if the relative server capacity \( u_s \) scales linearly with \( N \). Kurtz’s theorem provides a law of large numbers for density dependent jump Markov processes, which is presented in Theorem 4.6.

**Theorem 4.6 (Kurtz)** Suppose we have a density dependent family satisfying the Lipschitz condition

\[ |F(x) - F(y)| \leq M|x - y| \]
for some constant $M$, where $F(x) = \sum_{l \in L} l \beta_l(x)$. Further suppose $\lim_{n \to \infty} X_n(0) = x_0$, and $X$ satisfies

$$X(t) = x_0 + \int_0^t F(X(s))ds, \quad t \geq 0.$$  

Consider the path $\{X(s) : s \leq t\}$ for some fixed $t \geq 0$, and assume that there exists a compact $K \subset E$ around this path satisfying,

$$\sum_{l \in L} |l| \sup_{x \in K} \beta_l(x) < \infty.$$  

Then

$$\lim_{n \to \infty} \sup_{s \leq t} |X_n(s) - X(s)| = 0 \quad a.s.$$  

The preconditions of Kurtz’s theorem include boundedness and the Lipschitz condition, which we should verify for our Markov process. Note that the rate at which jumps occur is bounded above by $\frac{u_s}{N} + u_p$ everywhere, because $\frac{x_i}{1-x_m} \leq 1$ for all $0 \leq i \leq m - 1$. Further, the Lipschitz condition is valid for the domain $D$ defined by $-\varepsilon < x_i < 1 + \varepsilon$ ($i < m$) and $x_m < 1 - \varepsilon$, for $\varepsilon > 0$. We thus obtain the following theorem.

**Theorem 4.7** Suppose $t(\varepsilon)$ is the time before which the solution to (4.4) does not leave the domain $D$ defined by $-\varepsilon < x_i < 1 + \varepsilon$ ($i < m$) and $x_m < 1 - \varepsilon$, for $\varepsilon > 0$. The rescaled process $N^{-1}X(t)$ converges almost surely to the solution of the system of differential equations given in (4.4), with initial conditions $x_0(0) = 1$ and $x_i(0) = 0$ ($1 \leq i \leq m$), for all $s \leq t(\varepsilon)$.

Unfortunately, this theorem cannot be applied directly to prove Theorem 4.1, because it is in general difficult to obtain a closed-form solution to the system of differential equations (4.4). We thereby construct another Markov process to “bound” the original Markov process $\{X(t)\}_{t \geq 0}$. 
In our new Markov process, we assume that each node, when transmitting a coded block, selects the target peer uniformly at random from all its neighbors regardless of their states. Let $Z_0(t)$ denote the number of peers holding no coded blocks within segment $s$ at time $t$, and $Z_1(t)$ denote the number of peers holding one or more coded blocks within segment $s$ at time $t$. This defines a new Markov process $\{Z(t)\}_{t\geq 0}$ taking its values in $\mathbb{Z}_+^2$. The transition rates of the Markov process $\{Z(t)\}_{t\geq 0}$ are given by

$$q(Z, Z') = \begin{cases} \frac{Z_0}{N}(u_s + u_p(N - Z_0)) & \text{for } Z' = Z + (1, 1), \\ 0 & \text{for all other choices of } Z' \end{cases}$$

Similarly, we can apply Kurtz’s theorem to prove that the rescaled process $N^{-1}Z(t)$ converges almost surely to the solution of the system of differential equations as follows.

$$\frac{dz_i(t)}{dt} = \begin{cases} -z_0(t)(\frac{u_s}{N} + u_p(1 - z_0(t))) & \text{for } i = 0 \\ z_0(t)(\frac{u_s}{N} + u_p(1 - z_0(t))) & \text{for } i = 1 \end{cases} \tag{4.5}$$

with initial conditions $z_0(0) = 1$ and $z_1(0) = 0$.

**Theorem 4.8** The rescaled process $N^{-1}Z(t)$ converges almost surely to the solution of the system of differential equations given in (4.5), with initial conditions $z_0(0) = 1$ and $z_1(0) = 0$.

**Proof:** Note that the rate at which jumps occur is bounded above by $\frac{u_s}{N} + u_p$ everywhere.
Let \( x = (x_i) \) and \( y = (y_i) \) be two states of the new Markov process, for \( i \in \{1, 2\} \). Then

\[
|F(x) - F(y)| \leq |y_1\left(\frac{u_s}{N} + u_p(1 - y_1)\right) - x_1\left(\frac{u_s}{N} + u_p(1 - x_1)\right)| \\
+ |x_1\left(\frac{u_s}{N} + u_p(1 - x_1)\right) - y_1\left(\frac{u_s}{N} + u_p(1 - y_1)\right)| \\
\leq 2\left(\frac{u_s}{N} + u_p\right)|x_1 - y_1| + 2|x_1^2 - y_1^2| \\
\leq \left(2\left(\frac{u_s}{N} + u_p\right) + 4\right)|x_1 - y_1|
\]

Therefore, the rescaled process \( N^{-1} Z(t) \) satisfies the preconditions of Kurtz’s theorem, completing the proof. \( \square \)

The system of differential equations (4.5) have the following closed-form solution:

\[
z_0(t) = \frac{1}{1 + \delta + \frac{\delta}{1 + \delta}e(1 + \delta)u_p t}, \text{ and } z_1(t) = 1 - z_0(t)
\]

where \( u_p \) is the relative peer capacity, and \( \delta = \frac{u_s}{Nu_p} \) is the server strength. This closed-form solution can be used to provide some useful information on the original Markov process \( \{X(t)\}_{t \geq 0} \). Towards this end, we introduce the following definition.

**Definition 4.4** We say that the random vector \((X_1, X_2, \ldots, X_n)\) is stochastically greater than the random vector \((Y_1, Y_2, \ldots, Y_n)\), written \((X_1, X_2, \ldots, X_n) \geq_{st} (Y_1, Y_2, \ldots, Y_n)\) if for all increasing functions \( f \)

\[
E[f(X_1, X_2, \ldots, X_n)] \geq E[f(Y_1, Y_2, \ldots, Y_n)].
\]

We say that the stochastic process \( \{X(t), t \geq 0\} \) is stochastically greater than \( \{Y(t), t \geq 0\} \) if

\[
(X(t_1), X(t_2), \ldots, X(t_n)) \geq_{st} (Y(t_1), Y(t_2), \ldots, Y(t_n))
\]
for all $n, t_1, t_2, \ldots, t_n$.

We will show that $\{Z_0(t), t \geq 0\} \succeq_{st} \{X_0(t), t \geq 0\}$, when starting from the same initial conditions. The intuition is that: the new Markov process $\{Z(t)\}_{t \geq 0}$ is likely to waste some coded blocks so that the number of peers holding no coded blocks tends to be larger.

**Theorem 4.9** The stochastic process $\{Z_0(t), t \geq 0\}$ is stochastically greater than $\{X_0(t), t \geq 0\}$, when starting from the same initial conditions.

**Proof:** We use the coupling method [13, 39] to prove the above. For each node $i$ ($i \in \{0, 1, \ldots, N\}$), we generate an independent clock that ticks according to a Poisson process of rate $u_i$. When a node’s clock ticks, it chooses one neighbor uniformly at random and transmits one coded block. We count the number of peers that holding no coded blocks at time $t$ and denote it by $Z^*(t)$. Then $\{Z^*(t), t \geq 0\}$ has the same probability distributions as $\{Z_0(t), t \geq 0\}$. Next we generate another stochastic process by doing the following: whenever a node selects a neighbor that has received all $m$ coded blocks, it reselects a neighbor from those that do not have received all $m$ coded blocks. We again count the number of peers that holding no coded blocks at time $t$ and denote it by $X^*(t)$. Then $\{X^*(t), t \geq 0\}$ has the same probability distributions as $\{X_0(t), t \geq 0\}$. Note that $Z^*(t) \geq X^*(t)$, for all $t \geq 0$, by the construction. It follows that $f(Z^*(t_1), Z^*(t_2), \ldots, Z^*(t_n)) \geq f(X^*(t_1), X^*(t_2), \ldots, X^*(t_n))$ for all $n, t_1, t_2, \ldots, t_n$ and all increasing functions $f$. This proves the result.

Combining Theorem 4.7, Theorem 4.8, and Theorem 4.9, one can bound the stochastic process $\{X_0(t), t \geq 0\}$.\[\square\]
Theorem 4.10 Suppose \( t(\varepsilon) \) is the time before which the solution to (4.4) does not leave the domain \( D \) defined by \(-\varepsilon < x_i < 1 + \varepsilon \ (i < m) \) and \( x_m < 1 - \varepsilon \), for \( \varepsilon > 0 \). Then

\[
\frac{X_0(s)}{N} \leq \frac{1}{1+\delta} + \frac{\delta}{1+\delta} e^{(1+\delta)u_p s}, \quad \text{for all} \ s \leq t(\varepsilon),
\]

which holds with high probability for a sufficiently large \( N \).

Now we are ready to prove the sufficient condition for smooth playback during a flash crowd. We need only to make sure that the most urgent segment \( s \) has been successfully received at all participating peers by the end of current time slot. To this end, we count the number of coded blocks to ensure smooth playback of segment \( s \) in current time slot. On the one hand, as we will see in Sec. 4.2.4, the expected number of coded blocks to achieve smooth playback is given by \( N(1+\alpha)m \). On the other hand, the expected supply of coded blocks within segment \( s \) in the new Markov process \( \{Z(t)\}_{t \geq 0} \) is given as follows:

\[
u_sm + \int_0^m u_p(N - Z_0(t))dt = u_sm + Nu_pm - \int_0^m u_pZ_0(t)dt.
\]

Notice that

\[
\int_0^m u_pZ_0(t)dt = u_p\int_0^m \frac{N}{1+\delta} + \frac{\delta}{1+\delta} e^{(1+\delta)u_pm} dt
\]

\[
= N(1+\delta)u_pm - N \ln \left( \frac{1}{1+\delta} + \frac{\delta}{1+\delta} e^{(1+\delta)u_pm} \right)
\]

\[
\leq N(1+\delta)u_pm - N \ln \left( \frac{\delta}{1+\delta} \right) - N \ln \left( e^{(1+\delta)u_pm} \right)
\]

\[
= N \ln(1+\delta) - N \ln \delta.
\]

It follows that the expected supply of coded blocks within segment \( s \) in the new Markov
process is no less than \( N(1 + \delta)u_pm + N \ln \delta - N \ln(1 + \delta) \).

By Theorem 4.10, we know that if the expected supply of coded blocks in the new Markov process is greater than the expected demand given by \( N(1 + \alpha)m \),

\[
N(1 + \delta)u_pm + N \ln \delta - N \ln(1 + \delta) \geq N(1 + \alpha)m,
\]

then smooth playback can be achieved in the original Markov process \( \{X(t)\}_{t \geq 0} \). This gives a sufficient condition for smooth playback, which can be rewritten as

\[
U_s + NU_p \geq (1 + \varepsilon)NR,
\]

where \( \varepsilon = \alpha + \frac{\ln(1+\delta)-\ln\delta}{m} \), completing the proof of Theorem 4.1 for the general homogenous case.

We next turn to the two-class heterogeneous case. The proof for this case is essentially exactly the same as that given above for the general homogeneous case. Let \( u_i \) be the ratio of the upload capacity of a class-\( i \) peer \( U_i \) to the streaming rate \( R \). Let \( N_i \) be the number of class-\( i \) peers during a flash crowd. We have \( \sum_i N_i = N \), and the relative average peer capacity \( u_p \) is given by \( u_p = (N_1u_1 + N_2u_2)/N \), where \( N \) is the scale of the flash crowd.

Let \( X_i^j(t) \) denote the number of class-\( j \) peers holding exactly \( i \) coded blocks within segment \( s \) at time \( t \), for any \( i \in \{0, 1, \ldots, m\} \) and any \( j \in \{1, 2\} \). Again, a peer holding exactly \( i \) coded blocks within segment \( s \) is called a type-\( i \) peer. This defines a Markov process \( X(t) := \{X_i^j(t)\} \), where \( i \in \{0, 1, \ldots, m\} \) and \( j \in \{1, 2\} \). Using Kurtz’s theorem, one can prove that the rescaled process \( N^{-1}X(t) \) converges almost surely to the solution
of the following system of differential equations

\[
\frac{dx_j^i(t)}{dt} = \begin{cases} 
-x_0^j(t) \frac{u_j}{N} + \sum_{j=1}^2 u_j (1 - x_0^j(t)) & \text{for } i = 0, j \in \{1, 2\} \\
\frac{x_j^{i-1}(t) - x_j^i(t)}{1 - \sum_{j=1}^2 x_m(t)} \frac{u_j}{N} + \sum_{j=1}^2 u_j (1 - x_j^i(t)) & \text{for } 0 < i < m, j \in \{1, 2\} \\
\frac{x_m(t)}{1 - \sum_{j=1}^2 x_m(t)} \frac{u_j}{N} + \sum_{j=1}^2 u_j (1 - x_m^i(t)) & \text{for } i = m, j \in \{1, 2\}
\end{cases}
\]

(4.6)

with initial conditions \(x_0^j(0) = \frac{N_j}{N}\) and \(x_1^i(0) = 0\) (\(1 \leq i \leq m\)) under mild conditions, as stated in the following theorem.

**Theorem 4.11** The rescaled process \(N^{-1}X(t)\) converges almost surely to the solution of the system of differential equations given in (4.6), with initial conditions \(x_0^j(0) = \frac{N_j}{N}\) and \(x_1^i(0) = 0\) (\(1 \leq i \leq m\)), in any domain \(D\) where the Lipschitz condition holds.

**Proof:** It is easily verified that the Markov process \(\{X(t)\}_{t \geq 0}\) defined above is a density dependent jump Markov process with the transition rates at which jumps occur bounded above by \(\frac{u_j}{N} + u_p\) everywhere. Therefore, the rescaled process \(N^{-1}X(t)\) satisfies the preconditions of Kurtz’s theorem in any domain \(D\) where the Lipschitz condition holds, completing the proof. \(\square\)

Since this theorem cannot be applied directly to prove Theorem 4.1, we construct another Markov process to “bound” the original Markov process \(\{X(t)\}_{t \geq 0}\). The construction is exactly the same as that for the general homogeneous case. Let \(Z_0^j(t)\) denote the number of class-\(j\) peers holding no coded blocks within segment \(s\) at time \(t\), and \(Z_1^i(t)\) denote the number of class-\(j\) peers holding one or more coded blocks within segment \(s\) at time \(t\). This defines a new Markov process \(Z(t) := \{Z_i^j(t)\}\), where \(i \in \{0, 1\}\) and \(j \in \{1, 2\}\).

Similarly, one can apply Kurtz’s theorem to prove that the rescaled process \(N^{-1}Z(t)\)
converges almost surely to the solution of the system of differential equations as follows.

\[
\frac{dz^j(t)}{dt} = \begin{cases} 
-\frac{u_s}{N} z^j_0(t) - z^j_0(t) \sum_{j=1}^{2} u_j (n_j - z^j_0(t)) & \text{for } i = 0, j \in \{1, 2\} \\
\frac{u_s}{N} z^j_0(t) + z^j_0(t) \sum_{j=1}^{2} u_j (n_j - z^j_0(t)) & \text{for } i = 1, j \in \{1, 2\}
\end{cases}
\]  

with initial conditions \(z^j_0(0) = \frac{N_j}{N}\) and \(z^j_1(0) = 0\).

The closed-form solution of differential equations (4.7) is given as follows:

\[
z^j_0(t) = \frac{n_j}{1 + \delta} + \frac{\delta}{1 + \delta} e^{(1 + \delta)u_p t}, \text{ and } z^j_1(t) = n_j - z^j_0(t), \text{ for } j \in \{1, 2\},
\]

where \(n_j = \frac{N_j}{N}\), \(u_p = n_1 u_1 + n_2 u_2 = \frac{N_1 u_1 + N_2 u_2}{N}\) is the relative average peer capacity, and \(\delta = \frac{u_s}{N u_p}\) is the server strength. From this, we can obtain the following theorem by the coupling method.

**Theorem 4.12** Suppose \(t(D)\) is the time before which the solution to (4.6) does not leave the domain \(D\) in which the Lipschitz condition holds. Then

\[
\frac{X^j_0(s)}{N} \leq \frac{n_j}{1 + \delta} + \frac{\delta}{1 + \delta} e^{(1 + \delta)u_p s}, \text{ for all } s \leq t(D),
\]

which holds with high probability for a sufficiently large \(N\).

Theorem 4.12 suggests that if the expected supply of coded blocks in the new Markov process \(\{Z(t)\}_{t \geq 0}\) is greater than the expected demand given by \(N(1 + \alpha)m\), then smooth playback can be achieved in the original Markov process \(\{X(t)\}_{t \geq 0}\). This provides a sufficient condition for smooth playback during flash crowds.

Notice that the expected supply of coded blocks in the new Markov process \(\{Z(t)\}_{t \geq 0}\)
4.2. PERFORMANCE ANALYSIS OF CODING

is given as follows:

\[ u_s m + \sum_{j=1}^{2} \int_{0}^{m} u_j (N_j - Z_j^0(t)) dt \]

\[ = u_s m + N u_p m - \sum_{j=1}^{2} \int_{0}^{m} u_j Z_j^0(t) dt \]

\[ = u_s m + N u_p m - \sum_{j=1}^{2} \int_{0}^{m} \frac{u_j N_j}{1 + \delta} \frac{1}{1 + \delta} e^{(1+\delta)u_p t} \]

\[ = u_s m + N u_p m - u_p \int_{0}^{m} \frac{N}{1 + \delta} + \frac{\delta}{1 + \delta} e^{(1+\delta)u_p t} \]

\[ \geq u_s m + N u_p m + N \ln \delta - N \ln(1 + \delta), \]

which is exactly the same as that for the general homogenous case. This completes the proof of Theorem 4.1 for the two-class heterogeneous case.

We provide several interesting comments on Theorem 4.1 here. First, the heterogeneity in peers’ upload capacities is not an issue in coding. This is a consequence of the random push operations. More specifically, it is upstream peers rather than downstream peers that decide how to allocate their total upload capacities. Thus, high bandwidth peers naturally serve more, without the need of intricate protocol design on the downstream side. Moreover, as both high bandwidth and low bandwidth peers employ random downstream peer selection strategy, only the average upload capacity matters in the sufficient condition (4.1).

Second, Theorem 4.1 applies well under a large population asymptotic regime, but it does not point out how large the population should be. As seen in Fig. 4.5, Theorem 4.1 provides very accurate predictions when the scale of a flash crowd is larger than 100,000. The predictions are less accurate when the scale becomes smaller, however, they still give a correct characterization of the required server capacity, both qualitative
and quantitative.

The assumption that the topology of the mesh structure is a complete graph is quite strong. As we will see in Fig. 4.6, however, the results derived by Theorem 4.1 still hold when the size of neighborhood is about 100 in the mesh structure. Moreover, even if each peer maintains only 50 neighbors, the playback quality is as high as 0.98, regardless of the simulation scenarios. This is not a coincidence, as our analysis does not depend heavily on the complete graph assumption. In fact, this assumption can be relaxed to a random graph mesh structure, as long as the average number of neighboring peers is reasonably large. However, the derivations become more complicated; for this reason, we simply validate that our results also hold for the case of random graphs through simulations.

\section*{4.2.4 On the Fraction of Redundant Blocks}

When network coding is employed, a peer may receive redundant (linearly dependent) coded blocks from its upstream peers, leading to a waste of bandwidth resources. We denote by $\alpha$ the fraction of linearly dependent coded blocks in CODING. We are interested in an estimation of $\alpha$, which is critical as we evaluate the bandwidth utilization of CODING.

Without loss of generality, we consider a downstream peer $d$ to which multiple upstream peers are serving the same segment simultaneously. The following lemma states that with high probability, any coded block from an upstream peer is useful to peer $d$, as long as the space spanned by the coded blocks on the upstream peer is not a subspace of the space spanned on peer $d$.

\begin{lemma}
\text{(Lemma 2.1, [40]) Let $S_d$ denote the space spanned by the coded blocks on peer $d$ and $S_v$ denote the space spanned on one of peer $d$’s upstream peer, namely peer $v$.}
\end{lemma}
4.2. PERFORMANCE ANALYSIS OF CODING

Consider a coded block $x$ sent from peer $v$ to peer $d$. Then,

$$
\Pr(\text{coded block } x \text{ is useful } | S_v \subseteq S_d) \geq 1 - \frac{1}{q},
$$

where $q$ is the size of the Galois field.

Assume that the probability of the event $\{S_v \subseteq S_d\}$, denoted as $p$, is the same for all upstream peers of peer $d$. We shall approximate the block accumulating process on peer $d$ as follows. With probability $1 - p$, an upstream peer $v$ is helpful to peer $d$ (i.e., $S_v \subseteq S_d$). With probability $1 - \frac{1}{q}$, a helpful upstream peer sends a useful coded block to peer $d$.

**Proposition 4.1** In the simple model, the expected fraction $\alpha$ of redundant coded blocks is given as follows:

$$
\alpha = \frac{1}{(1-p)(1-\frac{1}{q})} - 1.
$$

**Proof:** Let $Y_i$ be the indicator function of the event that the $i$th coded block is useful to peer $d$. Clearly, $Y_i$ is a binary random variable with $\Pr(Y_i = 1) = (1-p)(1-\frac{1}{q})$. Let $M$ denote the number of coded blocks needed for peer $d$ to successfully decode the original segment. We can then write

$$
Y_1 + Y_2 + \cdots + Y_M = m.
$$

Note that the random variable $M$ is a *stopping rule* for $\{Y_n : n \geq 1\}$. Using Wald’s equality [13], we have

$$
E[M] = \frac{m}{E[Y_i]} = \frac{m}{(1-p)(1-\frac{1}{q})}.
$$
The expected fraction $\alpha$ of redundant coded blocks is then given by

$$\alpha = \frac{E[M] - m}{m} = \frac{1}{(1 - p)(1 - \frac{1}{q})} - 1.$$\[\square\]

Proposition 4.1 states that there are two reasons why peers might send redundant coded blocks. First, the randomized encoding algorithm on an upstream peer does not take into account the coded blocks accumulated on its downstream peers, and thus inevitably produces some redundant coded blocks due to this blind operation. Such redundancy is closely related to the size of the Galois field $q$. The larger the size, the less the redundancy. The second cause is the rare event that an upstream peer has no innovative coded blocks for its downstream peers. The probability of such event is substantially small, as the random push operations naturally create sufficient diversity.

If we choose the probability $p$ of such event to be 0.1% and the size $q$ to be 256, then the fraction $\alpha$ is less than 0.5% according to Proposition 4.1. To validate our estimation of $\alpha$, we take advantage of Galois field computation functions in MATLAB for encoding and decoding implementation. We only present simulation results for the flash crowd scenario here, as similar results have been obtained for the highly dynamic scenario. We let the size $q$ vary from 32 to 512. The results are shown in Table 4.4. As expected, the fraction of redundant coded blocks $\alpha$ decreases with the size $q$. Moreover, $\alpha$ is less than 0.3% in all cases, which agrees with our analysis well.

Table 4.4: Simulation results for fraction $\alpha$ of redundant coded blocks. We can see that the fraction of redundancy induced by network coding is in the order of 0.001, even when the field size $q$ is as small as 64.

<table>
<thead>
<tr>
<th>size $q$</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
</tr>
</thead>
<tbody>
<tr>
<td>redundancy $\alpha$</td>
<td>0.0024</td>
<td>0.0016</td>
<td>0.0014</td>
<td>0.0013</td>
<td>0.0012</td>
</tr>
</tbody>
</table>

We have also investigated the impact of the scale of a flash crowd. We have observed
that the fraction $\alpha$ of redundant coded blocks decreases as the scale increases, which suggests large-scale systems often create greater diversity, which in turn further reduces the redundancy.

4.3 Comparison between CODING and PULL

To further evaluate the performance of streaming protocols with network coding (CODING) in a comparison study with traditional pull-based mesh protocols (PULL), we have implemented an event-driven simulation tool in C to conduct a series of large-scale simulations. The simulation tool models many important peer characteristics and strategies, such as peer joining and leaving, segment selection, peer selection, and buffer map exchanging. For the sake of scalability to a large scale, we intentionally choose not to model P2P streaming systems in the finest details. Instead, our simulation tool captures a carefully selected set of important properties and strategies that we focus on in this thesis, such as peer arrivals and departures, heterogeneous peer upload capacities, and buffer-map exchanges. Its internal data structures are specifically tailored to scale well to a large number of peers within a reasonable simulation time, and it is used throughout this thesis for the purpose of simulations.

We use a random graph to represent the mesh structure formed by participating peers, and the average size of neighborhood is set to 50. The duration of each segment in CODING is set to 5 seconds, which is further divided into 100 blocks, while the duration of each block in PULL is set to 1 second. To simulate the heterogeneity of peer upload capacities, we use three types of peers, whose capacities are 3 Mbps, 384 Kbps and 128 Kbps, respectively. The streaming rate is set to 300 Kbps. By adjusting the fractions of different types of the peers, we obtain several different average upload capacities.
4.3. COMPARISON BETWEEN CODING AND PULL

For instance, a combination of 10% 3 Mbps peers, 38% 384 Kbps peers, and 52% 128 Kbps peers leads to an relative average peer capacity of 1.05. Unless otherwise specified, the number of participating peers is set to 10,000. We also scale to 100,000 in some simulation scenarios.

![Figure 4.7: A comparison of playback quality between CODING and PULL under different peer dynamic scenarios. In (a), we set the relative server capacity $u_s$ to 50 and the relative average peer capacity $u_p$ to 1.08; In (b), we set $u_s$ to 60 and $u_p$ to 1.05.](image_url)

We compare the performance of CODING and PULL in terms of playback quality and initial buffering delays, under several different scenarios of peer dynamics. We first consider the flash crowd scenario. As shown in Fig. 4.7(a), the playback quality in CODING degrades very slowly with the scale of a flash crowd. Moreover, perfect playback quality is achieved if the scale of a flash crowd is less than 70,000. In contrast, the playback quality in PULL is much lower than that in CODING. Further, it deteriorates significantly as the scale of a flash crowd increases. This is because newly joined peers in PULL are not able to utilize their upload capacities very effectively. Before a newly joined peer serves a block, it has to inform its neighboring peers by exchanging buffer maps and to wait for the explicit requests. Moreover, a certain amount of blocks may arrive after
their playback deadlines due to these interactions, resulting in degraded playback quality.

We now turn to the highly dynamic scenario. To decouple the effect of system scale, we let the arrivals of new peers approximately match the departures of existing peers in each time slot. Fig. 4.7(b) shows the playback quality of both CODING and PULL under different peer churn rates (i.e., the ratio of the number of departures in each time slot to the system scale, as defined in Sec. 4.2.2). As expected, peer churn has little impact on the playback quality in CODING, since the average peer capacity is not significantly affected. However, PULL suffers from such peer churn due to the difficulty of finding appropriate upstream peers.

Table 4.5: Initial buffering delay (in seconds) in PULL using the same configuration as in Fig. 5.1(a) (denoted as flash) and in Fig. 5.1(b) (denoted as churn). In CODING, the initial buffering delay strictly ranges from 5 to 10 seconds regardless of peer dynamics.

|         | flash 43 44 45 45 46 46 46 46 | churn 19 22 23 26 27 27 29 32 33 35 |

As to the initial buffering delays, we present the simulation results of PULL in Table 4.5, which use the same configuration as in Fig. 4.7. Specifically, we vary the scale of a flash crowd from 10,000 to 100,000 in the flash crowd scenario and the churn rate from 0.01 to 0.10 in the highly dynamic scenario. Recall that the initial buffering delay in CODING is strictly ranging from 5 to 10 seconds, regardless of peer dynamics. In Table 4.5, we observe that the initial buffering delay increases slowly in PULL with the scale of a flash crowd, and is substantially larger than that in CODING. In addition, peer churn affects the initial buffering delay significantly in PULL, as it usually takes much more effort for a newly joined peer to find appropriate upstream peers under a higher churn rate.

We also investigate the change of playback quality over time in both CODING and
4.4. SUMMARY

In Fig. 4.8, we see that the playback quality in CODING remains perfectly above 0.97 all the time and nearly 1 at most times, while PULL maintains a much lower and varied playback quality. These results show that CODING also enjoy excellent stability under peer dynamics.

![Playback quality graphs for CODING and PULL](image)

Figure 4.8: The change of playback quality over time in CODING and PULL under a typical flash crowd scenario and a highly dynamic scenario. We set the relative server capacity $u_s$ to 60 and relative average peer capacity $u_p$ to 1.05 in both scenarios. The comparison shows that CODING has much better stability under peer dynamics than PULL.

4.4 Summary

In this chapter, we have analytically investigated the performance of streaming systems with network coding that follow our design principles. The use of network coding not only eliminates some mathematical difficulties associated with previous theoretical models, but also leads to simple and effective streaming protocols. In particular, we have demonstrated that any streaming protocol using our design principles is sufficient to achieve provably good performance with respect to many important metrics, such as
playback quality, initial buffering delay, resilience to peer dynamics, as well as bandwidth cost on dedicated streaming servers. The simplicity in the core functionalities of our design principles further allows us to use a more realistic system model, taking into account many of the essential features of P2P streaming, such as peer dynamics (peer joining and leaving) and heterogeneous peer upload capacities. With extensive large-scale simulations, we validate our analytical results and demonstrate clear advantages of network network-coding-based protocols over traditional pull-based streaming protocols.
Chapter 5

Concluding Remarks

In this thesis, we have first investigated the performance of traditional pull-based protocols. The unique strength of pull-based protocols is the simplicity, which makes them the choice of many real-world streaming systems. The essential features of pull-based protocols include periodic buffer-map exchanges and lack of centralized scheduling. These features contribute most to the simplicity of pull-based protocols, but at the same time, lead to a performance gap between the fundamental limits and the actual performance. We have analytically explored such a performance gap in Chapter 3 by examining the effects of periodic buffer-map exchanges and lack of centralized scheduling. We show that there exists a significant performance gap between the fundamental limits and the actual performance of pull-based protocols. Moreover, periodic buffer-map exchanges account for most of the gap that separates the actual and optimal performance. In contrast, the lack of centralized scheduling only results in a small degree of performance loss.

These analytical results on the performance gap, which are also of independent interest, motivate our design principles for streaming systems with network coding. They
suggest that to significantly reduce the performance gap and reach the fundamental limits, one has to eliminate the need for periodic buffer-map exchanges, which is exactly the essence of our design principles proposed in Chapter 4.

The use of network coding not only leads to simple and effective streaming protocols, but also eliminates some mathematical difficulties associated with previous theoretical studies. In particular, we have demonstrated in Chapter 4 that any streaming protocol using our design principles is sufficient to achieve provably good performance with respect to many important metrics, such as playback quality, initial buffering delay, resilience to peer dynamics, as well as bandwidth cost on dedicated streaming servers. The simplicity in the core functionalities of our design principles further allows us to use a more realistic system model, taking into account many of the essential features of P2P streaming, such as peer dynamics (peer joining and leaving) and heterogeneous peer upload capacities. With extensive large-scale simulations, we validate our analytical results and demonstrate clear advantages of network-coding-based protocols over traditional pull-based streaming protocols.
Bibliography


