ON THE DESIGN OF PEER-ASSISTED VIDEO-ON-DEMAND SYSTEMS

BY

JIAHUA WU

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 JIAHUA WU. ALL RIGHTS RESERVED.
On the Design of Peer-Assisted Video-on-Demand Systems

Master of Applied Science Thesis
Edward S. Rogers Sr. Dept. of Electrical and Computer Engineering
University of Toronto

by Jiahua Wu
September 2009

Abstract

Peer-assisted Video-on-Demand (VoD) systems have not only received substantial recent research attention, but also been implemented and deployed with success in large-scale real-world streaming systems. Despite the remarkable popularity in real-world systems, the design of such systems are not well understood. In this thesis, we seek to address two design problems in peer-assisted VoD systems. First, we focus on the design of cache replacement algorithms. We construct an analytical framework based on dynamic programming, to help us form an in-depth understanding of optimal strategies to design cache replacement algorithms. Second, we shift our attention to the surplus upload bandwidth allocation problem in multi-channel systems. Through theoretical analysis and realistic simulations, we conclude that surplus upload bandwidth from peers can be utilized more efficiently than conventional prefetching strategies when it is devoted to redistributing content to those channels in deficit state.
To my parents and Xi
Acknowledgments

First of all, I would like to express my sincere attitude to my supervisor, Professor Baochun Li, for his continued guidance and support throughout the process of completing this thesis. His invaluable insights and patient guidance continued to inspire and motivate me to push the research frontiers to a new level.

To my thesis committee, Professor Shahrokh Valaee, Professor Cristiana Amza and Professor Ashish Khisti. Thank you for all the time and effort spent on reviewing this manuscript. Your valuable feedback and comments greatly improve the quality of this thesis.

I would also like to acknowledge the constant assistance and encouragements from everyone in the iQua research group, Chen Feng, Yuan Feng, Jin Jin, Elias Kehdi, Yunfeng Lin, Zimu Liu, Di Niu, Hassan Shojania, Mea Wang, Hui Wang, Chuan Wu, Henry Xu, Junqi Yu, Xinyu Zhang. Hours of stimulating discussions and friendly debates during the past years generated many brilliant ideas, which substantially contributed to the development of my thesis work.

Last but not least, I must express my attitude to my parents, and my girlfriend Xi Yin. Their unconditional love and support are the most important power that kept my perseverance throughout the past years. To them I dedicate this thesis.
Contents

Abstract ii

Acknowledgments iii

List of Tables vii

List of Figures xi

1 Introduction 1
   1.1 Cache Replacement Algorithm Design 3
   1.2 Surplus Upload Resource Allocation 4
   1.3 Structure of the Thesis 5

2 Related Work 7
   2.1 Cache Management 8
   2.2 Bandwidth Allocation 9

3 Cache Replacement Algorithm Design 12
   3.1 System Model 13
   3.2 Analysis of Cache Replacement Algorithms 16
3.2.1 Optimal Cache Replacement Algorithm in the System with the Knowledge of Segment Popularity .......................... 17
3.2.2 Serving Probability Matrix $P(t)$ ..................................... 23
3.2.3 Properties of the Optimal Control Process .............................. 25
3.2.4 Cache Replacement Algorithms in the System without the Knowledge of Segment Popularity .............................. 27
3.3 Practical Cache Replacement Algorithm Design ......................... 29
3.4 Performance Evaluation .................................................. 32
  3.4.1 Effects of Neighborhood Sizes and Cache Sizes ..................... 34
  3.4.2 Performance in Small Scale Systems ................................. 36
  3.4.3 Effects of System Sizes ............................................. 36
  3.4.4 Effects of Optimization Frequency .................................. 37
  3.4.5 Effects of Peer Churn .............................................. 38
3.5 Summary ........................................................................ 39

4 Upload Bandwidth Allocation across Multiple Channels .................. 42
  4.1 An Overview of Bandwidth Allocation Problem .......................... 43
    4.1.1 An Illustrative Example ............................................. 43
    4.1.2 System Model and Notations ........................................ 46
  4.2 User Perceptual Quality .................................................. 47
    4.2.1 Fraction of Late Packets ............................................ 47
    4.2.2 Link Between User Perceptual Quality and Fraction of Late Packets 50
  4.3 Optimization Framework .................................................. 51
  4.4 Algorithms for Multi-Objective Programming ............................ 54
    4.4.1 Elitist Non-Dominated Sorting Genetic Algorithm ............... 56
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.2</td>
<td>Greedy Heuristic Algorithm</td>
<td>59</td>
</tr>
<tr>
<td>4.5</td>
<td>Performance Evaluation</td>
<td>63</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Fraction of Late Packets vs. Server Upload Capacities</td>
<td>64</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Effects of Clustering Coefficients</td>
<td>66</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Effects of Neighborhood Sizes</td>
<td>66</td>
</tr>
<tr>
<td>4.6</td>
<td>Summary</td>
<td>68</td>
</tr>
<tr>
<td>5</td>
<td>Concluding Remarks and Future Work</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>73</td>
</tr>
</tbody>
</table>
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>List of Variables and Notations</td>
<td>16</td>
</tr>
<tr>
<td>3.2</td>
<td>Optimization Based Cache Replacement Algorithm</td>
<td>31</td>
</tr>
<tr>
<td>4.1</td>
<td>Key Notations in the System Model</td>
<td>46</td>
</tr>
</tbody>
</table>
# List of Figures

1.1 An illustrative example of peer-assisted VoD systems. ........................................... 2

3.1 An illustrative example of LRU algorithm. Segments in the local cache are sorted by the most recent access times. The segment pointed by the arrow line is to be replaced. ................................................................. 33

3.2 An illustrative example of LFU algorithm. A history of access events is maintained, and the number of access events for each segment in this history is computed. The segment pointed by the arrow line is to be replaced. 33

3.3 Average server load over the total demand under various neighborhood sizes and cache sizes, in large systems with 10000 peers. We set $B/M$ to 10% in (a) and $k$ to 50 in (b). ................................................................. 35

3.4 Average server load over the total demand under various cache sizes, in small systems with 50 peers. We set $B/M$ to 10% and $k$ to 50. ............... 37

3.5 Average server load over the total demand under various system sizes. We set $B/M$ to 5% and $k$ to 50. ................................................................. 38

3.6 Average server load over the total demand under various optimization frequencies, in large systems with 10000 peers. We set $B/M$ to 10% and $k$ to 50. ................................................................. 39
3.7 Average server load over the total demand under various churn rates, in large systems with 10000 peers initially. We set $B/M$ to 10% and $k$ to 50.

4.1 Serving media files within the same channel. Peers within each channel are circled by dashed lines. Superscripts of the server and peers are their normalized upload capacity (upload capacity divided by streaming rate). Capacity of each link is also marked next to the link.

4.2 Serving media files across multiple channels. Peers within each channel are circled by dashed lines. Superscripts of the server and peers are their normalized upload capacity (upload capacity divided by streaming rate). Capacity of each link is also marked next to the link.

4.3 A refined example of serving media files across multiple channels. Normalized upload capacities of peers are the same as those in the previous two examples, and thus omitted in this figure. Capacity of each link is marked next to the link.

4.4 An example illustrates the way of deriving the relation between user perceptual quality and peer’s download bandwidth through an intermediate factor, the fraction of late packets. The top left figure shows the relation between download bandwidth and the fraction of late packets, while the top right figure presents the relation between user perceptual quality and the fraction of late packets.
4.5 The tradeoff between normalized user perceptual quality and server upload capacity. $|\mathcal{C}|$ is set to 4, and each channel consists of 250 peers. The parameters for the system are initialized as the following. We set $\alpha$ to 1, $\gamma$ to 2, $U_{\text{max}}$ to 5, clustering coefficient for each channel to 0.2, 0.3, 0.5, 0.5 and mean of the normalized upload capacities for each channel to 3, 1, 0.75, 0.5, respectively. 63

4.6 Average fraction of late packets under various server upload capacities. $|\mathcal{C}|$ is set to 3, and each channel consists of 2000 peers on average. We set $k$ to 100, $k_c$ to 12, clustering coefficient to 0.1 and mean of the normalized upload capacities for each channel to 0.5, 1, 2, respectively. Please refer to Fig. 4.7 for the legend. 65

4.7 Average fraction of late packets under various clustering coefficients. $|\mathcal{C}|$ is set to 3, and each channel consists of 2000 peers on average. We set $k$ to 100, $k_c$ to 12, normalized server upload capacity to 600 and mean of the normalized upload capacities for each channel to 0.5, 1, 2, respectively. 67

4.8 Average fraction of late packets under various number of neighbors within the same channel. $|\mathcal{C}|$ is set to 3, and each channel consists of 2000 peers on average. We set $k_c$ to 12, clustering coefficient to 0.1, normalized server upload capacity to 600 and mean of the normalized upload capacities for each channel to 0.5, 1, 2, respectively. Please refer to Fig. 4.7 for the legend. 68
4.9 Average fraction of late packets under various number of neighbors from other channels. $|\mathcal{C}|$ is set to 3, and each channel consists of 2000 peers on average. We set $k$ to 100, clustering coefficient to 0.1, normalized server upload capacity to 600 and mean of the normalized upload capacities for each channel to 0.5, 1, 2, respectively. Please refer to Fig. 4.7 for the legend. 69
Chapter 1

Introduction

As multimedia content becomes prevalent in the Internet, multimedia Video-on-Demand systems traditionally rely on dedicated servers — prominent examples include YouTube and Google Video — often incurring prohibitive server bandwidth costs. Conventional wisdom calls for the use of proxy caching servers at the AS-level gateways to mitigate such bandwidth costs at dedicated servers. However, a recent measurement study [17] shows that the popularity of media content follows the Stretched Exponential (SE) distribution, which makes it inefficient and infeasible to deploy proxy cache servers.

Peer-assisted Video-on-Demand systems, on the other hand, have not only received substantial recent research attention [21], but also been implemented and deployed with success in large-scale real-world streaming systems, such as PPLive [23]. The essential advantage of peer-assisted VoD is to dramatically increase the number of peers a streaming channel may sustain with dedicated servers. Intuitively, as participating peers contribute their upload bandwidth capacities to serve other peers in the same channel, the load on dedicated servers is significantly mitigated.
The roots of peer-assisted VoD systems can be traced back to BitTorrent, a peer-to-peer file sharing protocol used for distributing large amounts of data. With BitTorrent, when multiple peers are downloading the same file simultaneously, they upload pieces of the file to each other. This approach redistributes the cost of upload to peers in the system, thus making hosting a file with a potentially unlimited number of peers.

Unlike the file-sharing application in BitTorrent, VoD systems deliver the content by streaming, and peers can watch different parts of a video at the same time. The asynchronization access from users dilute their ability to help each other and offload the server. To compensate, each user contributes a small amount of storage, and stores media files recently downloaded in this local cache. This additional resource opens up opportunities for arranging suitable patterns of content replication to meet diverse user demands. Essentially, peer-assisted VoD system is a P2P replication system, plus a scheduling mechanism for directing peers to help each other in real time. An illustrative example of peer-assisted VoD systems is shown in Fig. 1.1.

Figure 1.1: An illustrative example of peer-assisted VoD systems.
In peer-assisted VoD systems, two types of resources, \textit{i.e.}, local cache size and upload bandwidth, from participating peers are limited, which restricts their ability to serve each other. Consequently, it is of vital importance to efficiently utilize these precious resources in order to improve the overall system performance. Two questions, how to manage the media content stored in peer’s local cache, and how to allocate surplus upload bandwidth from peers, naturally arise. In this thesis, we seek to unravel these two questions in the peer-assisted VoD system design.

\section*{1.1 Cache Replacement Algorithm Design}

For peers in the system, each of them uses a pre-determined and fixed amount of disk space as a \textit{cache} of recently downloaded media segments. In the PPLive VoD system [23], for example, the pre-determined size of such a media cache may be in the order of 1 GB. As peers download media segments from servers or other peers for on-demand playback, they are stored in the cache to potentially serve other peers, hence reducing bandwidth costs on servers. Peers in PPLive, however, do not choose to \textit{actively prefetch} any media segment that it has not requested for playback, including later segments from the same on-demand media stream. One of the reasons against the use of active prefetching in PPLive is that downloading activities on most real-world DSL peers may affect upload bandwidth contributions [23].

As the size of such a cache on each peer is limited, it is imperative that an appropriate cache replacement algorithm is designed. Such an algorithm is critically important, as it determines the availability of media segments in the entire system. Such segment availability affects server bandwidth costs, as peers will download segments directly from servers if they become unavailable from other peers.
Consider peer-assisted VoD systems with a limited pool of *passive* media cache space, where no active prefetching is used and the media cache content is not actively managed by centrally controlled servers. We have good news and bad news in this case. The good news is that there exists a tremendous level of flexibility in the design space of such cache replacement algorithms, including the simplest alternatives such as Least Recently Used (LRU) and Least Frequently Used (LFU). Which algorithm is the best to minimize server bandwidth costs, so that when peers need a media segment, it is most likely available from caches of other peers? The bad news is that such a question is arguably non-trivial to answer, as both the demand and supply of media segments are stochastic in nature in the system.

**1.2 Surplus Upload Resource Allocation**

In addition to local cache size, upload bandwidth from each participating peer is limited as well, and it is imperative to utilize this precious resource efficiently. Consider the circumstance when the instantaneous aggregate upload capacity outweighs aggregate demand within a channel, conventional wisdom calls for the use of prefetching of future content for peers in this channel. Under this scenario, surplus upload capacity is used to distribute future content to the peers, creating a reservoir of prefetched content that can be exploited when the system shifts into deficit state. Efficient prefetching schemes can also be used to facilitate the random seek functionality in VoD systems based on statistics of user behavior. A substantial number of schemes [21, 19, 43] were proposed in VoD systems to enjoy the benefit of prefetching. However, as user behavior can not be accurately predicted, a considerable amount of the content prefetched may never have the chance to be used, thus leading to a waste of upload bandwidth.
A recent study [5] reveals interesting social networks with strong clustering among videos with their long-term measurement over five million YouTube videos. The high overlap in peers’ interest introduces the potential in allocating upload bandwidth from peers across multiple channels. Consider a peer-assisted VoD system serving two media files, where the aggregate upload capacity outweighs aggregate demand in one channel, while demand exceeds supply in the other channel. As a subset of peers in the upload bandwidth over-provisioned channel might have the other media file stored in their local cache, surplus capacity from this channel can be utilized to help redistribute content in the other one. However, it is arguably non-trivial to answer the question that how we optimally allocate upload capacity from peers across these two channels, so that the aggregate system performance, in terms of user perceptual quality, is maximized.

In addition, server bandwidth cost is another factor that should be taken into consideration in this problem. From the aspect of content providers, the ultimate goal is to mitigate the server bandwidth costs while maintaining a satisfactory system performance. However, the goal of improving the user perceptual quality usually conflicts with that of reducing server upload bandwidth. How to solve these two conflicting objectives concurrently is another challenging issue to deal with. Furthermore, the allocation of server upload bandwidth is directly determined by the bandwidth resource allocation strategy from peers. In other words, different allocation strategies for the peers may lead to different strategies for the servers.

1.3 Structure of the Thesis

The remainder of the thesis is structured as follows. In Chapter 2, we highlight our original contributions in the context of related work.
1.3. STRUCTURE OF THE THESIS

In Chapter 3, despite the challenges imposed by both stochastic demand and stochastic supplies of media segments, we seek to construct a tractable analytical framework based on optimal control theory and dynamic programming, to help us form an in-depth understanding of optimal strategies to design cache replacement algorithms. With complementary simulation studies, we show that the performance margin enjoyed by a practical algorithm design of optimal strategies over the simplest algorithms is not substantial, with respect to reducing server bandwidth costs. In most cases, the simplest choices are good enough as cache replacement algorithms in peer-assisted VoD systems.

In Chapter 4, we consider the two conflicting objectives concurrently, and form a unifying multi-objective optimization framework to optimize user perceptual quality and server bandwidth cost simultaneously. Under this framework, we seek to practically allocate server capacity and surplus capacity from peers across multiple channels to achieve optimal aggregated user perceptual quality. We first derive the relation between peer download bandwidth and user perceptual quality, based on queueing theory. Then, we construct a tractable framework based on multi-objective optimization theory, to help us form an in-depth understanding of the trade-offs between server upload capacity and user perceptual quality. We propose a greedy heuristic algorithm to unravel this problem, and compare its performance with the conventional evolutionary algorithm, i.e., NSGA-II, via numerical analysis. With complementary simulation studies, we show that our greedy heuristic enjoys consistently better performance than the prefetching scheme in the isolated channel design.

Finally, Chapter 5 concludes the thesis and presents future work.
Chapter 2

Related Work

Peer-assisted Video-on-Demand systems have received substantial research attention recently. Two of the most representative work is [21] and [23]. Huang et al. [21] assessed what the server bandwidth costs would have been if a peer-assisted employment had been utilized by using a trace from a client-server VoD deployment for MSN Video. They showed that peer-assistance can dramatically reduce server bandwidth costs, especially if peers prefetch content when there is spare upload capacity in the system. In [23], the authors discussed the challenges and the architectural design issues of a large-scale P2P-VoD system based on the experiences of a real system deployed by PPLive. Later on, many researchers devote their enthusiasm and time to the design of peer-assisted VoD systems. In this chapter, we present and discuss existing literatures about cache management and upload bandwidth allocation in VoD systems.
2.1 Cache Management

The use of proxy and client-side caching has long been discussed in VoD systems in order to reduce bandwidth costs on servers and access latencies for media content. Significant research efforts have been devoted to their improvement of system performance [36, 24, 16]. Alan et al. [24] proposed a COPACC protocol, which leveraged the client-side caching to amplify the aggregated cache space. Also in the proxy caching system, Lei et al. [16] deployed the storage and upload resources from peers to improve the quality of on-demand media streaming. However, media segments stored in both proxies’ and clients’ cache are actively managed by centrally controlled proxies or distributed algorithms, rather than passively managed caching considered in our thesis, which is a more realistic setting in VoD systems over Internet. The stochastic nature of passively managed media cache space makes the availability of media segments random, which leaves the problem considered in our thesis more complex than those in previous work.

The application of cache management was also comprehensively investigated in the VoD services over cable networks [1, 33, 3]. In cable networks, content subscribers appear to be more cooperative because the facilities, such as home gateways or set-top boxes, are under the control of a network or content provider. Comprehensive theoretical analyses on peer-assisted video-on-demand on cable networks were conducted in [33, 3]. However, the emphasis of our thesis differs from those work in that we focus on the cache replacement algorithm in a system where the cache is passively, rather than actively, managed.

Allen et al. [1] did a comprehensive trace-driven evaluation in cache replacement algorithms on cable networks. Although similar conclusions were drawn, our work distinguishes from [1] in two aspects. First, the conclusion drawn in [1] completely relies on
2.2. BANDWIDTH ALLOCATION

simulation, rather than theoretical aspects considered in our work. Second, the placement of content in [1] is also dictated completely by the index servers, which is the same as other works on cable networks.

Recently, in the Internet, there also emerged a number of application-level tree-based or mesh-based P2P VoD protocols. However, only simple cache replacement algorithms were utilized in these proposals, \textit{e.g.}, FIFO [11], LRU [19] or simply replacing pieces earlier than the current file position [7]. Parvez et al. [29] did a comprehensive theoretical analysis on the performance of on-demand stored media content delivery using BitTorrent-like protocols. However, the local cache of the peers were assumed to be unlimited in their analysis.

A popularity-based storage space replacement algorithm was shown in [41]. The segments stored in peers’ local cache is directly determined by the algorithm, which is quite similar to the case considered in [1, 33, 3] over cable networks. As mentioned in preceding paragraphs, our work presents the theoretical analysis on peer-assisted VoD systems considering passive peer caching. Passive caching is a more realistic setting, since peers only download and store the content that interest them in their local cache. From the system level, the aggregated content cached in the system is random, which is determined not only by the cache replacement algorithm, but also users’ interests.

2.2 Bandwidth Allocation

The efficiency in utilizing resources, such as upload bandwidth, from participating peers is playing a pivotal role in improving system performance in peer-assisted systems. Under the circumstance where aggregate upload capacity outweighs aggregate demand, one of the most representative approach is \textit{prefetching} in previous literature. Two crucial
problems, which part of the content to prefetch [22, 43, 19], and how to allocate surplus upload capacity from peers [32, 37, 21], have been comprehensively investigated. With respect to the first question, data mining techniques [22], user behavior statistics collected through logs on the server [43], or gossips among peers [19] were applied in determining the content to prefetch.

Allocation of instantaneous surplus upload capacity among peers in the system is the other problem in designing prefetching schemes. Shen et al. [32] investigated the prefetching problem in the system where each media content is encoded into multiple low bit-rate substreams, and assignment of prefetching rates to each substream is governed by their importance. In [21], two policies, *i.e.*, the water-leveling policy and greedy policy, were proposed to allocate the surplus upload capacity in peer-assisted VoD systems. These policies were proved to be near optimal by comparing their performance with the theoretical bound. Our work distinguishes from the previous literature in two aspects. First, our thesis considers the system where peers are able to upload media files to peers in other channels, as long as this media file is stored in its local cache. The content peers upload is directly determined by requests from peers, either from the same channel or different channels, rather than estimation in [22, 43, 19]. Second, when taking multiple channels into consideration, surplus upload capacity from peers are first allocated among channels, and then peers in each individual channel. This problem is not involved in the design of prefetching schemes in the isolated channel design.

Recently, the design of multi-channel peer-assisted systems attracted significant attention. There emerged a number of application-level mesh-based P2P protocols, which leveraged the benefit of multi-channel design [6, 38, 39]. Xu et al. [6] proposed a peer-assisted delivering framework that explores the clustering in social networks for short
2.2. BANDWIDTH ALLOCATION

video sharing. In their framework, peers prefetch video prefixes using their surplus upload capacity in order to achieve fast and smooth transitions between video files. In [38, 39], the authors proposed a view-upload decoupling (VUD) framework, which strictly decouples what a peer uploads from what it views. The VUD design brought stability to multi-channel systems, and enjoyed better performance compared with the isolated channel design. However, the content to prefetch in [6] or upload in [38, 39] are actively managed by centrally controlled servers or distributed algorithms, rather than passively requested by peers considered in our paper, which is a more realistic setting in VoD systems over Internet. Consequently, no surplus upload capacity is wasted in downloading media content that might never be used by peers in our approach.
Chapter 3

Cache Replacement Algorithm Design

Peer-assisted VoD systems are designed to take full advantage of peer upload bandwidth contributions with a cache on each peer. Since the size of such a cache on each peer is limited, it is imperative that an appropriate cache replacement algorithm is designed. There exists a tremendous level of flexibility in the design space of such cache replacement algorithms, including the simplest alternatives such as Least Recently Used (LRU). Which algorithm is the best to minimize server bandwidth costs, so that when peers need a media segment, it is most likely available from caches of other peers? Such a question, however, is arguably non-trivial to answer, as both the demand and supply of media segments are stochastic in nature. In this Chapter, we seek to construct an analytical framework based on optimal control theory and dynamic programming, to help us form an in-depth understanding of optimal strategies to design cache replacement algorithms. With such analytical insights, we have shown with extensive simulations that, the performance margin enjoyed by optimal strategies over the simplest algorithms is
not substantial, when it comes to reducing server bandwidth costs. In most cases, the simplest choices are good enough as cache replacement algorithms in peer-assisted VoD systems.

3.1 System Model

We first present our theoretical model for the cache replacement problem in P2P Video-on-Demand systems. Our model applies to both problems with and without the knowledge of segment popularity. While conducting the analysis with the knowledge of segment popularity, the distribution of segment popularity is known \textit{a priori}. However, instead of probability distributions, the demand for each segment belongs to a given set, which is bounded by the number of peers, in the analysis without the knowledge of segment popularity. We first define the notations and assumptions in the theoretical analysis.

Let there be $N$ peers and a dedicated server in the system. We consider the server with a repository of some media files. The length of the files may vary. However, each of them is composed of some constant-length segments and the total number of segments stored in the server is of size $M$. In real-world systems, segment is defined and used for the purpose of advertising to neighbors what parts of a movie a peer holds. The typical size of a segment is around several MBs. However, unlike the situation in real world, we assume all the segments are independent in our analysis.

We assume the access probabilities of all the segments are known \textit{a priori} in the analysis with the knowledge of segment popularity. Let $f_i$ be the access probability of the segment $i$. The sum of access probability $f_i$ satisfies the condition $\sum_{i=1}^{M} f_i = 1$. Without loss of generality, we assume $f_i \geq f_j$, $\forall$ $i < j$. By assigning identical popularity for all segments within the same media file, we can easily generalize our model to the
3.1. SYSTEM MODEL

system with the knowledge of probability distribution for each file.

Each peer in the system maintains a limited size of local cache, which is able to store up to \( B \) segments. We assume that the peers in the system are homogeneous, which indicates that all the peers have the same upload capacity and local cache size. We analyze the system performance within the context of time intervals or time slots. The time intervals and time slots have the same meaning in our analysis, hence we will use these two terms interchangeably in the rest of the paper. The duration of a time interval corresponds to the amount of time for a peer to upload a single segment.

By making the aforementioned assumptions, the behavior of the peers can be characterized in a time-slotted fashion. Let vector \( \mathbf{X}(t) = (x_1(t), x_2(t), \ldots, x_M(t))^T \) denote the state of the peers’ cache in the system. Each entry \( x_i(t) \) stands for the number of peers that hold segment \( i \) in its local cache at time interval \( t \). The local cache of all peers will eventually be completely filled as time progresses. Without loss of generality, we assume the cache of all the peers are filled up at time interval 0. The sum of all the entries in the state vector is equal to the storage space from all the peers, namely \( \sum_{i=1}^{M} x_i(t) = BN, \forall t \). In fact, our analysis can be extended to more general situations in which the extent of fill in peers’ local cache are divergent. In this case, we only need to take those peers whose cache is filled to the cache limit into consideration, and derive the cache replacement algorithm for these peers.

At the beginning of every time slot, each peer will request a segment it will playback in the near future from the peers that have this specific segment in their local cache. Each peer sends out requests to all the peers holding this specific segment in their local cache. Once this request is satisfied by one of the target peers, the peer will download the entire segment during this time slot. The remaining unsatisfied requests are redirected
3.1. SYSTEM MODEL

to the dedicated server and all requests sent to the server are assumed to be satisfied within the time slot. At each time slot, it is natural for a peer to receive several requests from other peers, but it is able to respond to only one request at each slot and it selects the request uniformly at random. For simplicity of exposition, we ignore the request and acknowledgment latency at the beginning of each time slot.

After introducing the working mechanism of the system, we can now model it in a more rigorous way. At each time slot, the requests sent from the peers can be characterized by a random vector \( \mathbf{W}(t) = (w_1(t), w_2(t), \ldots, w_M(t))^T \), where \( w_i(t) \) denotes the number of requests for segment \( i \) at time slot \( t \). In the problem with the knowledge of segment popularity, the random vector \( \mathbf{W}(t) \) should meet the following conditions:

\[
\sum_{i=1}^{M} w_i(t) = N, \quad \forall \ t \\
\Pr\{\mathbf{W}(t) = \mathbf{A}\} = \frac{N!}{a_1!a_2!\cdots a_M!} f_1^{a_1} f_2^{a_2} \cdots f_M^{a_M}, \quad \forall \ t
\]

where \( \mathbf{A} = (a_1, a_2, \ldots, a_M)^T \) is any vector satisfying \( \sum_{i=1}^{M} a_i = N \) and \( a_i \in \mathbb{N}, \forall \ i \). We will discuss the constraints that the random vector should meet in the problem without the knowledge of segment popularity in Sec. 3.2.4.

At the same time, each peer downloads a new segment and replaces one stored in its local cache at every time slot. Like the random demand vector, the segments being replaced can also be described by a control vector \( \mathbf{U}(t) = (u_1(t), u_2(t), \ldots, u_M(t))^T \), where \( u_i(t) \) denotes the number of segment \( i \) being replaced at time slot \( t \). This control vector is the key parameter in our system, since the selection of segments to be replaced at each time slot is directly determined by the cache replacement algorithm. Different replacement strategies may impact the availability of media segments severely. It is easy
Table 3.1: List of Variables and Notations

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>The number of peers in the system</td>
</tr>
<tr>
<td>$M$</td>
<td>The total number of segments in the system</td>
</tr>
<tr>
<td>$B$</td>
<td>The number of segments a peer can store in its local cache</td>
</tr>
<tr>
<td>$f_i$</td>
<td>The access probability of segment $i$</td>
</tr>
<tr>
<td>$X(t)$</td>
<td>The state vector, with each entry $x_i(t)$ indicating the number of peers hold segment $i$ in its local cache at time $t$</td>
</tr>
<tr>
<td>$W(t)$</td>
<td>The random demand vector, with each entry $w_i(t)$ indicating the number of requests for segment $i$ at time $t$</td>
</tr>
<tr>
<td>$U(t)$</td>
<td>The control vector, with each entry $u_i(t)$ indicating the number of segment $i$ being replaced at time $t$</td>
</tr>
<tr>
<td>$P(t)$</td>
<td>The diagonal serving probability matrix, with each entry $p_i(t)$ on the diagonal indicating the average probability of the peers currently holding segment $i$ in its local cache to respond to a request for this segment</td>
</tr>
</tbody>
</table>

To see that the control vector should obey the following rules:

$$\sum_{i=1}^{M} u_i(t) = N, \quad \forall \ t$$  \hspace{1cm} (3.1)

$$0 \leq u_i(t) \leq x_i(t), \ i = 1, 2, \ldots, M, \quad \forall \ t$$  \hspace{1cm} (3.2)

$$u_i(t) \in \mathbb{N}, \ i = 1, 2, \ldots, M, \quad \forall \ t$$  \hspace{1cm} (3.3)

However, the control vector is a macro-parameter of the system and it may sometimes cause conflict for its allocation to individual peers. We will take a close look at this problem in Sec. 3.3.

### 3.2 Analysis of Cache Replacement Algorithms

The model introduced in the previous section set the stage for us to deduce the optimal cache replacement algorithm. In this section, we begin with the problem having the knowledge of segment popularity.
3.2.1 Optimal Cache Replacement Algorithm in the System with the Knowledge of Segment Popularity

The state that the system evolves to after the $t$th time interval is

$$X(t + 1) = X(t) + W(t) - U(t)$$

(3.4)

where $X(0)$ is the initial state of the system. This state is determined by the time that the cache of all the peers are filled up. Equation (3.4) characterizes the state evolution of our system.

The purpose of a cache replacement algorithm is to vary the availability of different segments so as to fully utilize the storage and upload resources from peers. In each time slot, we are willing to see that most of the requests are able to be satisfied by other peers, rather than directly from the server. The optimality of the cache replacement algorithm can be measured by the percentage of requests satisfied by the server directly. The mathematical formulation of this problem is as follows:

$$\min_{U(t)} \mathbb{E}\{\sum_{t=0}^{T-1} 1^T \cdot \max(0, W(t) - P(t)X(t))\}$$

(3.5)

where $1$ is a $M \times 1$ vector with all the entries equal to 1. $P(t) = \text{diag}(\bar{p}_1(t), \bar{p}_2(t), \ldots, \bar{p}_M(t))$ is a diagonal matrix. Each entry $\bar{p}_i(t)$ on the diagonal indicates the average probability of the peers currently holding segment $i$ to respond to the requests for segment $i$. To rigorously define $\bar{p}_i(t)$, we first define $A_i = \{\text{peer } j \mid \text{ peer } j \text{ has segment } i \text{ in its local cache}\}$, the set of peers holding segment $i$ in their local cache. Hence $\bar{p}_i(t) = \frac{\sum_{j \in A_i} p_{ij}(t)}{x_i}$,
where \( p_{ij}(t) \) denotes the probability of peer \( j \) selecting a request for segment \( i \) at time interval \( t \). This serving probability reflects the peer selection algorithm, segment selection algorithm and some other working mechanisms in the system, thus it is an important parameter in the system design. The properties of this serving probability matrix will be carefully examined in the next subsection.

\( W(t) \) in Equation (3.5) represents the random demand for each segment at time interval \( t \), while \( P(t)X(t) \) stands for the average supply from the peers. Although the contribution peers make to serve others at each time interval is random as well, we take the mean to approximate the random supply in the objective function. In a large-scale system, this approximation is quite accurate according to the law of large numbers. The difference of these two quantities indicates the amount of requests need to be satisfied by the server. The main objective of the cache replacement algorithm is to minimize the burden of the server, which in this case is equivalent to minimizing the sum of the difference over \( T \) time slots.

This is a stochastic optimal control problem, and we resort to Dynamic Programming (DP) algorithms [2] to solve it. First, we need to define some notations. Let

\[
\begin{align*}
g_T(X(T)) &= 0 \\
g_t(X(t), W(t)) &= 1^T \cdot \max (0, W(t) - P(t)X(t)), \\
& \quad t = 0, 1, \ldots, T - 1
\end{align*}
\]

where \( g_t(X(t), W(t)) \) denotes the sum of requests satisfied by the servers at time slot \( t \). The DP algorithm for our problem is illustrated as follows. For every initial state \( X(0) \), the optimal cost \( J^*(X(0)) \) of the problem is equal to \( J_0(X(0)) \), given by the last step of
3.2. ANALYSIS OF CACHE REPLACEMENT ALGORITHMS

the following algorithm, which proceeds backwards in time from period $T - 1$ to period 0:

$$
J_T(X(T)) = g_T(X(T)),
$$

$$
J_t(X(t)) = \min_{U_t \in U(X(t))} E\{g_t(X(t), W(t)) + J_{t+1}(X(t+1))\},
$$

$$
t = 0, 1, \ldots, T - 1
$$

where the expectation is taken with respect to the probability distribution of $W(t)$. $U(X(t))$ is the set of $U(t)$ satisfying Equations (3.1) – (3.3). If $U^*(t) = \mu^*_t(X(t))$ minimizes the right side of Equation (3.6) for each $X(t)$ and $t$, the policy $\pi^* = \{\mu^*_0, \ldots, \mu^*_{T-1}\}$ is considered to be optimal.

The basic idea behind this algorithm is the principle of optimality argument: “The tail portion of an optimal policy is optimal for the tail subproblem.” In other words, the optimal policy for the problem consisting of the last few stages is the portion of the optimal policy for the original problem. The power of this algorithm is to decompose the original problem into $T$ subproblems and each of the subproblem is a simpler optimization problem.

However, it is impossible to obtain an optimal policy in the closed form for our problem, and a numerical solution is therefore necessary. The computational requirements to obtain this solution are overwhelming, as they increase exponentially as the size of the problem increases, commonly referred to as “the curse of dimensionality” [2]. Fortunately, for this specific problem, it is possible to suppress the temporal component of the model and determine a one time interval optimum that is also optimal for the problem with the planning horizon $T$. In order to formally prove this argument, we have
3.2. ANALYSIS OF CACHE REPLACEMENT ALGORITHMS

to introduce two Lemmas first.

Lemma 3.1 (Convexity Preservation under Minimization) If $X$ is a convex set, $Y(x)$ is a nonempty set for every $x \in X$, the set $C := \{(x, y) \mid x \in X, y \in Y(x)\}$ is a convex set, $g(x, y)$ is a convex function on $C$,

$$f(x) := \inf_{y \in Y(x)} g(x, y),$$

and $f(x) > -\infty$ for every $x \in X$, then $f$ is a convex function on $X$.

Lemma 3.2 For any one time interval optimal policy, if $J_{t+1}(X(t+1))$ is convex, and the following conditions hold:

- $g_t(X(t))$ is convex;
- $J_t(X(t))$ is convex.

Then, the policy is optimal in each period of a finite-horizon problem.

The rigorous proofs of these Lemmas are omitted here. The proof of Lemma 1 can be found in conventional textbooks, e.g., [20]. For Lemma 2, interested readers can refer to [30]. Lemma 2 prepares the groundwork for showing that a myopic policy is optimal in every time slot. However, Lemma 2 is only suitable for the inventory control problems, the category that our problem incidentally falls in. Having introduced the Lemmas above, we now derive the optimal cache replacement algorithm.

Theorem 3.1 A myopic policy (one time interval optimum) is optimal for the problem described by Equations (3.1) – (3.5) with a planning horizon $T$. 

Proof: We first note that $J_T(X(T)) = g_T(X(T))$ is the zero function, so it is convex. Suppose $J_{t+1}(X(t+1))$ is convex. $g_t(X(t)) = 1^T \cdot \max (0, W(t) - P(t)X(t))$ is the sum of the maximum of affine functions. The convexity is preserved under maximization and summation [4], thus $g_t(X(t))$ is a convex function.

It remains to show the convexity of $J_t(X(t))$. To this end, define the set of state vectors

$$\mathcal{X} = \{X(t) \mid 0 \preceq X(t) \preceq N1, 1^T X(t) = BN, \forall \ t\}$$

(3.7)

The set $\mathcal{X}$ is convex. In order to show this, suppose $X_1$ and $X_2$ are two vectors in $\mathcal{X}$. Let $Y = \theta X_1 + (1 - \theta) X_2$, where $0 \leq \theta \leq 1$. From (3.7), we have

$$0 \preceq \min(X_1, X_2) \preceq Y \preceq \max(X_1, X_2) \preceq N \cdot 1$$

$$1^T \cdot Y = \theta \cdot 1^T X_1 + (1 - \theta) \cdot 1^T X_2 = BN.$$

This implies that $Y$ also belongs to the set $\mathcal{X}$. According to the definition of convexity, $\mathcal{X}$ is a convex set. By taking a similar approach, we can prove that, for every $X(t) \in \mathcal{X}$, the following set is convex:

$$U(X(t)) = \{U(t) \mid 0 \preceq U(t) \preceq X(t), 1^T \cdot U(t) = N, \forall \ t\}$$

Hence, we have that the set $C := \{(X(t), U(t)) \mid X(t) \in \mathcal{X}, U(t) \in U(X(t))\}$ is a convex set. Let

$$G(X(t), U(t)) = g_t(X(t), W(t)) + J_{t+1}(X(t) + W(t) - U(t))$$

(3.8)
It is the sum of two convex functions, thus is convex itself. Take the expectation of Equation (3.8), which does not change the convexity of the function, we then have that $E\{G(X(t), U(t))\}$ is a convex function on $C$. Following Lemma 3.1, it indicates that $J_t(X(t)) = \min_{U(t) \in U(X(t))} G(X(t), U(t))$ is a convex function. Finally, combining the last relation with the convexity of $g_t(X(t))$ and Lemma 3.2, we obtain the announced result.

Theorem 1 tells us that an optimal policy in any time interval $t$ is optimal over the planning horizon $T$. It remains to derive a one time interval optimum for our problem. To this end, consider the time interval $T-1$. It should be mentioned that the replacement of content occurs at the end of each time slot, thus in order to minimize the cost function at time interval $T-1$, we should optimize the control vector $U(t-2)$. If we denote $x^+ \triangleq \max\{x, 0\}$, the cost function at the $(T-1)$th time interval $J_{T-1}(X(T-1))$ is equal to

$$
\min_{U(T-2)} E\{1^T(W(T-1) - P(T-1)X(T-1))^+ + J_T(X(T))\}.
$$

Plugging (3.4) into the cost function and replacing $J_T(X(T))$ with 0, $J_{T-1}(X(T-1))$ can be expressed as follows:

$$
\min_{U(T-2)} E\{1^T(W(T-1) - P(T-1)(X(T-2) + W(T-2) - U(T-2)))^+\}.
$$

Changing the order of expectation and summation in the last formula, the optimal control problem at time interval $T-1$ can be obtained:

$$
\min_{U(T-2)} 1^T\{(I - P(T-1))\tilde{W} - P(T-1)X(T-2) + P(T-1)U(T-2)\}^+.
$$
subject to the constraints

$$1^T \cdot U(T - 2) = N$$
$$0 \leq U(T - 2) \leq X(T - 2)$$

where \(I\) is an \(M \times M\) identity matrix and \(\overline{W} = \{\overline{w}_1, \ldots, \overline{w}_M\}\) is the expected value of the random demand vector \(W\), which is equal to \(\{f_1N, f_2N, \ldots, f_M N\}^T\). We can introduce an \(M \times 1\) vector \(K\), which satisfies the following constraints:

$$K \succeq 0$$ (3.9)

$$(I - P(t + 1))\overline{W} - P(t + 1)X(t) + P(t + 1)U(t) \preceq K$$ (3.10)

Thus, the optimal cache replacement policy \(U^*(t)\) at each time slot \(t\) can be concluded by the following theorem.

**Theorem 3.2** The optimal cache replacement strategy is \(\pi^* = \{U^*(0), U^*(1), \ldots, U^*(T - 1)\}\). For each \(U^*(t)\), it is the solution to the following mixed-integer programming problem:

$$\min_{K, U(t)} \ 1^T \cdot K$$ (3.11)

subject to the constraints (3.1) – (3.3), (3.9) and (3.10).

### 3.2.2 Serving Probability Matrix \(P(t)\)

In this subsection, we examine the properties of the serving probability matrix \(P(t)\) carefully. The value of \(\bar{p}_i(t)\) on the diagonal of matrix \(P(t)\) is directly determined by the
probability of peer \( j \in \mathcal{A}_i \) choosing a request for segment \( i \) in each time interval. Hence, we first derive the value of \( p_{ij}(t) \). Denote \( K_j \) and \( N_j \), the number of requests for segment \( i \) that peer \( j \) receives and the total number of requests that peer \( j \) receives, respectively. \( \mathcal{B}_j \) indicates the set of segments that peer \( j \) has in its local cache.

\[
P\{\text{peer } j \text{ selects a request for segment } i\} = \sum_{n=0}^{N-1} \sum_{k=0}^{n} P\{\text{peer } j \text{ selects a request for segment } i \mid k, n\} \cdot P\{K_j = k \mid N_j = n\} \cdot P\{N_j = n\}
\]

\[
= \sum_{n=0}^{N-1} \sum_{k=0}^{n} \frac{k}{n} \cdot P\{K_j = k \mid N_j = n\} \cdot P\{N_j = n\}
\]

\[
= \sum_{n=0}^{N-1} \frac{1}{n} \cdot P\{N_j = n\} \cdot E\{K_j \mid N_j = n\}
\]

\[
= \sum_{n=0}^{N-1} \frac{1}{n} \cdot P\{N_j = n\} \cdot \frac{f_i}{\sum_{k \in \mathcal{B}_j} f_k}
\]

We then have

\[
\bar{p}_i(t) = \sum_{j \in \mathcal{A}_i} p_{ij}(t)/x_i = \frac{1}{x_i} \sum_{j \in \mathcal{A}_i} \left( \frac{f_i}{\sum_{k \in \mathcal{B}_j} f_k} \right) \tag{3.12}
\]

Since \( f_i \geq f_j \), \( \forall \ i < j \), we then obtain the relation \( \bar{p}_i(t) \in \left[ \frac{f_i}{f_i \sum_{j \in \mathcal{A}_i} f_j}, \frac{f_i}{B_j f_i} \right] \). Recall that, as we mentioned in Sec. 3.2.1, the serving probability matrix \( \mathbf{P}(t) \) is an important parameter, since it reflects the peer selection algorithm, segment selection algorithm and some other mechanisms in the system design. For instance, if each peer sends out only one request in every time interval, the serving probability is \( \bar{p}_i(t) = \frac{1}{x_i} \sum_{j \in \mathcal{A}_i} \left( \frac{f_i}{\sum_{k \in \mathcal{B}_j} f_k} \right) \). In the
general case, however, it is hard to derive a closed form of the serving probability.

### 3.2.3 Properties of the Optimal Control Process

In this subsection, we extend the analysis in Sec. 3.2.1 and derive the conditions for all requests to be exclusively satisfied by peers in the steady state. Let us first define $Y(T-2) = X(T-2) - U(T-2)$, which describes the state of the system at the end of time interval $T-2$. Hence, the cost function at $(T-1)^{th}$ time interval can be rewritten as

$$J_{T-1}(X(T-1)) = \min_{U(T-2)} 1^T \cdot \{(I - P(T-1)) \cdot W - P(T-1) \cdot Y(T-2)\}^+ \quad (3.13)$$

The objective of the cache replacement algorithm is to minimize the requests served by the dedicated server. Then it naturally raises the question whether it is possible to serve all the requests by peers only. We next derive the necessary condition for this argument.

**Theorem 3.3** It is possible to satisfy all requests exclusively by peers in the steady state, only if the total storage space from the peers is larger than or equal to $(MBf_1 - 1) \cdot N$, where $f_1$ is the maximal access probability for the segments.

**Proof:** Let us first define $\beta_{\text{sum}} = \max_t \{1^T \cdot P(t+1)^{-1}(I - P(t+1))W\}$. From (3.13), we can see that the cost function can be minimized to zero in the steady state if the following condition holds:

$$Y(t) \geq P(t+1)^{-1}(I - P(t+1))W \quad (3.14)$$
This relation will definitely be satisfied by the effects of the cache replacement algorithm if the total storage space from peers is larger than $\beta_{\text{sum}}$. Hence, we only need to focus on the case of $Y(t) = P^{(t+1)^{-1}}(I - P(t+1))W$ and derive the upper bound for it. This equation can be expressed in a more explicit way as follows:

$$Y(t) = \begin{pmatrix}
\frac{1-\bar{p}_1(t+1)}{\bar{p}_1(t+1)} & 0 & \ldots & 0 \\
0 & \frac{1-\bar{p}_2(t+1)}{\bar{p}_2(t+1)} & \ddots & \vdots \\
\vdots & \ddots & \ddots & 0 \\
0 & \ldots & 0 & \frac{1-\bar{p}_M(t+1)}{\bar{p}_M(t+1)}
\end{pmatrix} W \quad (3.15)$$

Replacing $\bar{p}_i(t+1)$ with the bounded value derived in Sec. 3.2.2, the upper and lower bound of each entry in the distribution of segments can be written as the following:

$$(Bf_M - f_i)N \leq y_i(t) \leq (Bf_1 - f_i)N$$

The upper bound of the total storage space from all the peers is equal to $\sum_{i=1}^{M}(Bf_1 - f_i) \cdot N$, and thus the announced result follows.

**Corollary 3.1** The maximal segment access probability must be less than or equal to $\frac{B+1}{MB}$, if all the requests are to be exclusively satisfied by peers.

**Proof:** The total storage space from peers in the model is equal to $BN$. Thus, according to Theorem 3.3, if all the requests are to be served by peers only, then it should satisfy the following condition:

$$(MBf_1 - 1) \cdot N \leq BN$$

$$f_1 \leq \frac{B+1}{MB}. $$
Theorem 3.3 and Corollary 3.1 tell us that a sufficiently large local cache on the peers can not guarantee that all the requests are guaranteed to be responded by peers exclusively. Beyond large local caches, segment popularity is another important factor. Based on Corollary 3.1, the popularity for all segments would be almost identical in the system, if all the requests are to be satisfied by peers only.

3.2.4 Cache Replacement Algorithms in the System without the Knowledge of Segment Popularity

In Sec. 3.2.1, we derived the optimal cache replacement algorithm in the system with the knowledge of segment popularity. However, it is usually hard to predict the exact probability distribution of segment popularity, due to the size of the system and evolution of popularity over time. Thus, the design of the cache replacement algorithm in the system without the knowledge of segment popularity is also of vital importance in some cases. In this subsection, we will briefly investigate this problem.

The system model we analyzed in this subsection is the same as that in Sec. 3.2.1. The system function is also represented by Equation (3.4). However, instead of the probability distribution, we only know that \( W(t) \) belongs to a given set \( \mathcal{W} \). For each \( W(t) \in \mathcal{W} \), it meets the following conditions.

\[
0 \preceq W(t) \preceq N1 \quad \forall t \tag{3.16}
\]

\[
1^T W(t) = N \quad \forall t \tag{3.17}
\]

\[
w_i(t) \in \mathbb{N}, i = 1, 2, \ldots, M, \quad \forall t \tag{3.18}
\]
3.2. ANALYSIS OF CACHE REPLACEMENT ALGORITHMS

Under this circumstance, we may use the minimax approach instead of the expected cost approach, which we used to analyze the system with the probability distribution of segment popularity. Thus, the objective is

$$\min_{U(t) \in U(X(t))} \max_{W(t) \in W} \left\{ \sum_{t=0}^{T-1} 1^T \max(0, W(t) - P(t)X(t)) \right\}$$

This problem can also be solved by the dynamic programming algorithm, which proceeds backwards in time from period $T - 1$ to period 0. However, for this minimax problem, there does not exist the myopic theory as that we utilized in Sec. 3.2.1. In order to keep the computation requirement under control, we propose a sub-optimal control algorithm which adopts the limited lookahead policy and utilizes a heuristic cost function approximation.

An effective way to reduce the computation required by dynamic programming is to truncate the time horizon and use, at each stage, a decision based on lookahead of a small number of stages. For this problem, we resort to a one-step lookahead policy whereby at stage $t$ and state $X(t)$, one uses the control vector $U(t - 1)$, which attains the minimum in the cost function:

$$J_t(X(t)) = \min_{U(t-1)} \max_{W(t)} \left\{ 1^T \cdot (W(t) - P(t)X(t))^+ + \tilde{J}_{t+1}(X(t+1)) \right\}$$

where $\tilde{J}_{t+1}$ is the one step approximation of the true cost function $J_{t+1}$, with $\tilde{J}_T(X(T)) = J_T(X(T)) = 0$. Given the approximations $\tilde{J}_t$ to the optimal cost functions, the computational savings of the limited lookahead approach are evident. For the one-step lookahead policy, only a single minimization problem has to be solved per stage.

The cost function approximation approach we take in our problem is based on the
feature extraction and state aggregation. The grouping of states that incur similar costs into subsets can greatly simplify the cost function. Let $\lambda_{\text{min}}(t)$ denote the minimal entry of the vector $P(t) \cdot X(t+1)$. At time slot $t+1$, the maximal cost caused by $W(t+1)$ is equal to $N - \lambda_{\text{min}}(t)$. The state $X(t+1)$ with the same minimal entry will incur the same maximal cost at time slot $t+1$, thus we can group the states by their minimal entry and design the approximation function for the one-step lookahead approach. Consider the finite horizon problem with a total of $T$ time intervals, $\tilde{J}_{t+1}$ approximates the cost over the last $T - t - 1$ intervals. The maximal cost incurred over these time slots is approximately $(T - t - 1) \cdot (N - \lambda_{\text{min}}(t))$, which we will take as the approximation cost function $\tilde{J}_t$.

The sub-optimal control for the system without the knowledge of segment popularity at time slot $t$ can be formulated as the following optimization problem:

$$
\min_{U(t-1) \in U(X(t-1))} \max_{W(t) \in W} \left\{ 1^T(W(t) - P(t)X(t))^+ + (T - t - 1)(N - \lambda_{\text{min}}(t)) \right\} \quad (3.19)
$$

subject to (3.1) – (3.3) and (3.16) – (3.18).

### 3.3 Practical Cache Replacement Algorithm Design

In Sec. 3.2, we analyzed optimal and sub-optimal cache replacement strategies in systems with and without the knowledge of segment popularity, respectively. We now proceed to the design of practical cache replacement algorithms, based on insights from our theoretical analysis. Such practical algorithms can serve as a “benchmark” of the best possible algorithms that may be realizable, to which simpler alternative heuristics can be compared.
Unfortunately, we understand that only sub-optimal policies may be proposed in the case without the knowledge of segment popularity. Realistically, however, segment popularity can be estimated with system-level mechanisms, such as the availability of a centralized tracking server [23], rank based estimations [17], as well as various decentralized algorithms [41]. As a result, we concentrate our efforts on designing optimal cache replacement algorithms with the knowledge of segment popularity.

Our basic idea is to develop the algorithm based on the optimal cache replacement strategy, mixed-integer optimization problem (3.11), derived in Sec. 3.2. However, there exist three challenges in utilizing this optimization formulation. First, the mixed-integer optimization problem is NP-hard in general [28]. In other words, it is unlikely to be solved in polynomial time. Second, future knowledge of the serving probability matrix is needed to derive the optimal solution. Third, even if we can obtain optimal integral solutions, there might exist problems to allocate the control vector to individual peers. Recall that in Sec. 3.1, we mentioned that the control vector is a macro-parameter, and it may sometimes cause conflicts. For example, at time slot $i$, only peer $k$ holds segment 1 and 2 in its local cache. The optimal solution may dictate that both $u_1(t)$ and $u_2(t)$ equal to 1, i.e., both segment 1 and 2 should be replaced at time slot $i$, which is in conflict with the fact that each peer only replaces one segment at each time slot. Thus, the control vector is not feasible at all times.

Approximation algorithms should be designed to address these challenges. For the mixed-integer optimization problem in the first challenge, it can be relaxed to a corresponding linear programming problem while discarding the integer constraints. In fact, this relaxation does not undermine the optimality of the solution. To respond to the
3.3. PRACTICAL CACHE REPLACEMENT ALGORITHM DESIGN

Table 3.2: Optimization Based Cache Replacement Algorithm

| 1. **Data Collection.** The tracking server collects the cache availability map of each peer periodically, with which it maintains up-to-date segment probabilities. |
| 2. For those peers whose local cache is full, repeat the following operations every time interval $\Delta t$: |
| 3. **Solve the main framework** (3.11). |
| (a) **Update Parameters.** Update the state vector $X(t)$ by calculating the number of each segment in the peers’ local cache. Update the expectation of demand vector $\bar{W}$ with $\bar{w}_i = N f_i = x_i(t)/b$. Update the serving probability matrix $P(t) = \text{diag}(\bar{p}_1(t), \bar{p}_2(t), \ldots, \bar{p}_M(t))$ according to (3.12). |
| (b) **Solve the linear programming problem.** |
| 4. Update the segment replacement probability vector and distribute it to individual peers. |

second challenge and the lack of future knowledge, we can simply use the serving probability matrix that is available at the current time. With respect to the third challenge, the conflict of allocation of the control vector can be solved by the following probabilistic replacement strategy.

Rather than selecting the segment for replacing peer by peer, we can set the probability of replacement for segment $i$ to be $q_i(t) = u_i^*(t)/N$ and distribute these probabilities to all peers. Each peer makes its decision according to the replacement probability distribution individually. The benefit of this scheme is two fold. First, by letting the peers make their own decisions individually, the control overhead is quite low in our algorithm. The control messages only occur when the optimization is re-solved. Second, this algorithm copes better with the situation that peers replace segments asynchronously. In the asynchronous case, it is not feasible to allocate the control vector to individual peers in
a centralized manner.

To estimate the segment popularity, our analysis in the previous section shows that it is best to have the knowledge of the cache availability maps of each peer, in order to obtain an accurate value of the serving probability matrix \( P(t) \). As such, we choose to utilize a centralized tracking server — typically used in real-world peer-assisted VoD systems — to centrally keep track of cache availability maps in all peers. Due to latencies of collecting such information, estimates of the segment popularity on the tracking server may not be accurate, but it is already the best one can do in practice.

With these measures, we can solve the optimization problem as a linear programming problem in a centralized fashion, known to be solvable in polynomial time. The algorithm is solved by a centralized tracking server, and it will distribute replacement probability vectors to individual peers. As demonstrated in simulations, increasing the optimization period will not result in a severe degradation for the performance of our algorithm. Consequently, the load of the control packets will not be overwhelming. Our optimization based cache replacement algorithm is summarized in Table 3.2.

### 3.4 Performance Evaluation

One of our research objectives is to design a practical optimal cache replacement algorithm as a “benchmark” to evaluate simple heuristic cache replacement strategies. In this section, via complementary simulation studies, we evaluate two commonly used candidates: Least Recently Used (LRU) and Least Frequently Used (LFU), in comparison with the optimization-based algorithm that we have designed. In the LRU strategy, a queue of segments is maintained sorted by the most recent access times. A segment is moved to the front of the queue when it is accessed. When the cache is full, the segment
at the end of the queue is discarded. In the LFU strategy, instead, a history of access
events that occur in all previous time slots is maintained, and the number of access events
for each segment in this history is computed. When the cache is full, the segment that
is accessed with the least frequency is discarded. Illustrative examples of LRU and LFU
are presented in Fig. 3.1 and Fig. 3.2, respectively.

![Local Cache:](image1)

Figure 3.1: An illustrative example of LRU algorithm. Segments in the local cache are
sorted by the most recent access times. The segment pointed by the arrow line is to be
replaced.

![Local Cache:](image2)

Figure 3.2: An illustrative example of LFU algorithm. A history of access events is
maintained, and the number of access events for each segment in this history is computed.
The segment pointed by the arrow line is to be replaced.

We conduct simulations to evaluate both LRU and LFU heuristics, as candidates of
simple heuristic designs, to our optimization-based algorithm, arguably the best achiev-
able strategy in practice. Unless otherwise specified, we simulate systems that consist
of random topologies of 10000 peers, with each peer connected with a varying number
of neighbors. Denote the mean of the number of neighbors to be $k$. Recall that $B$ and
3.4. PERFORMANCE EVALUATION

$M$ denote the number of segments a peer is able to store in its local cache and the total number of segments in the system, respectively. $B/M$, the local cache size from each peer over the total number of segments in the system, is set to be 10% in most simulations. For the optimization based algorithm, the default optimization interval is set to be 2 time slots.

Since our consistent objective is to mitigate the server bandwidth cost, the main performance metric of concern in our comparisons is the average server load over the total demand, i.e., the portion of requests served by the server. A smaller portion represents higher performance. Unless otherwise specified, the Y-axis in the subsequent figures indicate the average server load over the total demand. We further assume the segment popularity follows a stretched exponential distribution (SE), which was found in recent work to be the access pattern for most media workloads in Internet [17]. Hence, it is an appropriate approximation of the segment popularity distribution in our simulation.

3.4.1 Effects of Neighborhood Sizes and Cache Sizes

Fig. 3.3 shows the effects of neighborhood sizes and cache sizes on the server load reduction. We vary the values of the neighborhood size and cache size, and then compute the average server load over 500 time slots, the results of which are shown in the figure. We observe that the server load decreases dramatically with the increase of the neighborhood size and cache size. The curves in these two figures show similar trends, since the increase of the neighborhood size is equivalent to the increase of the cache size per peer.

Further, it is most revealing that the differences among the three cache replacement strategies are notably small. Though our optimization based algorithm shows consistently better performance than much simpler heuristics, the improvement is insignificant. The
Figure 3.3: Average server load over the total demand under various neighborhood sizes and cache sizes, in large systems with 10000 peers. We set $B/M$ to 10% in (a) and $k$ to 50 in (b).
differences are slightly more pronounced in small caches. As the cache size increases, there is little, if any, difference when the three algorithms are compared with one another.

### 3.4.2 Performance in Small Scale Systems

In order to further verify the performance of the algorithms in small caches, we set the total number of peers in the system to 50 in our simulations in this phase of our experiments. The topology of the system is set to be a complete graph. We vary the average cache size in the system and the results are shown in Fig. 3.4. The difference among these algorithms are more significant under this setting, which is consistent with our previous observation.

An interesting finding is that the LRU algorithm is more sensitive to the size of cache and the size of the system. In large scale systems, the LRU algorithm performs consistently better than the LFU algorithm, while its performance is much worse than other algorithms in the system with only 50 nodes. However, the difference becomes insignificant as the cache size increases.

### 3.4.3 Effects of System Sizes

After investigating large systems with 10000 peers and small systems with 50 peers, we are now interested on the effects of varying system sizes on the performance of cache replacement algorithms. Fig. 3.5 presents the average server load over the total demand while varying the system size from 1000 peers to 10000 peers. From Fig. 3.5, we can clearly see that the average server load over the total demand remains constant with the change of the system size in all these three algorithms. This demonstrates a superior degree of scalability of peer-assisted VoD systems, where new peers contribute more caching
Figure 3.4: Average server load over the total demand under various cache sizes, in small systems with 50 peers. We set $B/M$ to 10% and $k$ to 50.

storage space to the system. The bad news is that, even with such stellar scalability, a linear increase of server bandwidth costs need to be incurred to accommodate a linear increase of the system scale, and using even the practically optimal cache replacement algorithm does not make a significant difference.

### 3.4.4 Effects of Optimization Frequency

Although the optimization is conducted at every time slot in the analysis, it is worthwhile to investigate the effects of optimization frequency on the system performance in the simulation. Fig. 3.6 illustrates the average server load over the total demand while varying the optimization frequency from 2 to 20 time slots. The curve shows that the performance of optimization based algorithm will deteriorate sharply while the optimization interval
3.4. PERFORMANCE EVALUATION

Figure 3.5: Average server load over the total demand under various system sizes. We set \( B/M \) to 5\% and \( k \) to 50.

is over 5 time slots.

3.4.5 Effects of Peer Churn

Finally, although all previous analyses and simulations are performed in the static setting, we are also interested in the system performance under the effects of peer churn, including both peer arrivals and departures. The performance of all three algorithms in terms of the average server load over the total demand is shown in Fig. 3.7. The average server load over the total demand increases dramatically as the churn rate increases. The performance of cache replacement algorithms are similar to that in the static case when the churn rate is small. However, in the case of high churn rates, there exist no substantial performance gain when the optimal algorithm is used rather than its simpler alternatives.
When the peer lifetime is short with a high churn rate, a substantial percentage of peers are not able to fill up their local cache before they leave the system, making it futile to optimize the design of the cache replacement algorithm.

3.5 Summary

Real-world peer-assisted VoD systems commonly use peer caches that are limited in size and passively managed. What, then, is the best possible way to manage these peer caches? It is non-trivial to answer this question, since the passively managed nature of peer caching makes both demand and supply of media segments stochastic in nature, and as such more challenging than actively controlled caching. In fact, in passively managed peer caches, the only opportunity of exerting an influence on its contents — and thus
Figure 3.7: Average server load over the total demand under various churn rates, in large systems with 10000 peers initially. We set $B/M$ to 10% and $k$ to 50.

potentially mitigating server bandwidth costs by increasing peer contributions — is when the peer cache is full and existing segments need to be selected and replaced.

In this Chapter, we seek to answer this question with rigorous theoretical analysis and complementary simulation studies. Using optimal control theory and dynamic programming, we construct a tractable analytical framework and derive the optimal and sub-optimal strategies in systems with and without the knowledge of segment popularity, respectively. With our analysis, we are able to gain new insights on how an optimal algorithm should be designed; with our simulations, we are able to compare the performance of our optimal algorithm with the simplest heuristics, and draw the surprising conclusion that simple solutions perform as well as the optimal algorithm, with very insignificant differences. In most cases, simple heuristics are good enough to be used as cache replacement algorithms in peer-assisted VoD systems. The results in this Chapter
were published in [40].
Chapter 4

Upload Bandwidth Allocation across Multiple Channels

The essential advantage of peer-assisted Video-on-Demand (VoD) systems is to dramatically increase the number of peers a streaming channel may sustain with dedicated servers by exploiting resources from volatile peers. Consequently, the allocation of surplus upload bandwidth from peers is of vital importance in the VoD system design. Conventional wisdom calls for the use of prefetching of future content for peers within the same channel to improve user perceptual quality. However, such method suffers from the risk that a substantial amount of content prefetched may never be used.

Inspired by the observation of strong clustering among videos [5], we seek to solve this problem in a multi-channel system design, where the barriers between different channels are removed. Under this scenario, surplus upload bandwidth from one channel is steered to those channels in deficit state. In this paper, we formulate this problem as a multi-objective optimization problem, in which two objectives of our consideration are to improve user perceptual quality and to mitigate server bandwidth costs. Besides the
conventional evolutionary algorithm, NSGA-II, we propose a greedy heuristic algorithm, and show that it performs as well as — if not better than — NSGA-II for this specific problem. Through complementary simulations, we evaluate the performance of the greedy heuristic algorithm, and conclude that it enjoys consistently better performance than the prefetching strategy in the isolated channel design.

4.1 An Overview of Bandwidth Allocation Problem

Before we venture into theoretical analysis, we first introduce an illustrative example to show the benefit of bandwidth allocation across multiple channels.

4.1.1 An Illustrative Example

Figure 4.1: Serving media files within the same channel. Peers within each channel are circled by dashed lines. Superscripts of the server and peers are their normalized upload capacity (upload capacity divided by streaming rate). Capacity of each link is also marked next to the link.
4.1. AN OVERVIEW OF BANDWIDTH ALLOCATION PROBLEM

Figure 4.2: Serving media files across multiple channels. Peers within each channel are circled by dashed lines. Superscripts of the server and peers are their normalized upload capacity (upload capacity divided by streaming rate). Capacity of each link is also marked next to the link.

Consider the simple scenario in Fig. 4.1, where three media files are being served concurrently. A common practice in peer-assisted VoD systems today is to organize peers viewing the same channel into a swarm, with peers in the same swarm redistributing media chunks exclusively to each other. In this simple example, we assume users can have a satisfactory playback quality if the download bandwidth exceeds the streaming rate of the media file. As shown in Fig. 4.1, no matter how complicated a scheduling algorithm is designed, it is impossible for peers from all three channels to enjoy satisfactory performance for the following reasons. For each individual channel, the downloading rate from the server should exceed 1, no matter how peers within each channel is organized, in order to provide decent performance. However, by doing so, peers in channel b fall into a deficit state: the aggregate upload bandwidth is less than the aggregate supply, and quality of service can not be guaranteed.
4.1. AN OVERVIEW OF BANDWIDTH ALLOCATION PROBLEM

Figure 4.3: A refined example of serving media files across multiple channels. Normalized upload capacities of peers are the same as those in the previous two examples, and thus omitted in this figure. Capacity of each link is marked next to the link.

An elegant solution to the above problem can be easily observed if we remove the barrier between different channels. In our example, the only peer in channel a has already downloaded media files b and c, and can help to redistribute these files. For example, the peer in channel a can choose any peer in channel b and upload the media file b to it, while letting the server to upload the file to the other peer. A sample solution is illustrated in Fig. 4.2. For this problem, the main challenge lies in how to optimally allocate upload capacity from all the peers in order to maximize the aggregate system performance. This challenge is two fold. First, media content stored in peers' local cache limits the ability for peers to help each other. In other words, a peer can only relay content it has downloaded before and still stores in its local cache. Second, unlike the simple settings in this example, the relation between download bandwidth and system performance, in terms of the user perceptual quality, is unclear. How the download
bandwidth will alleviate the impact of delay jitter and packet loss in the user perceptual quality is another problem worth investigating.

However, this is not the whole story. By allowing peers to interact across multiple channels, it introduces the potential to save the bandwidth costs at the server. Those peers who help to redistribute media content in other channels act as seeds, and can thus alleviate the burden of the server without sacrificing system performance. In our example, two out of the five peers have media files — other than the file they are downloading — stored in their local cache, and a sample solution in order to fully utilize their upload capacity is shown in Fig. 4.3. In this case, 30% of server upload capacity can be saved compared with the original scenario.

<table>
<thead>
<tr>
<th>$\mathcal{C}$</th>
<th>The set of concurrent channels in the system.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{N}_c$</td>
<td>The set of peers in channel $c$.</td>
</tr>
<tr>
<td>$p^n_c$</td>
<td>The $n_{th}$ peer in channel $c$.</td>
</tr>
<tr>
<td>$V^n_c$</td>
<td>Upload capacity of peer $p^n_c$.</td>
</tr>
<tr>
<td>$R$</td>
<td>Streaming rate.</td>
</tr>
<tr>
<td>$v^n_c$</td>
<td>Normalized upload capacity of peer $p^n_c$.</td>
</tr>
<tr>
<td>$v^s_c$</td>
<td>Normalized server upload capacity for channel $c$.</td>
</tr>
<tr>
<td>$d^n_c$</td>
<td>Normalized download bandwidth of peer $p^n_c$.</td>
</tr>
<tr>
<td>$k$</td>
<td>Average number of neighbors within the same channel.</td>
</tr>
<tr>
<td>$k_c$</td>
<td>Average number of neighbors from different channels.</td>
</tr>
</tbody>
</table>

### 4.1.2 System Model and Notations

In the following sections, we consider a peer-assisted VoD system with a total of $N$ peers and a dedicated server. The total number of channels in the system is identical to the number of media files stored in the server, represented as a set of $\mathcal{C}$. The server is regarded
as a repository of $|\mathcal{C}|$ constant-bit-rate media files, and the peers downloading the same media file form a channel. Let $\mathcal{N}_c$ denote the set of peers in channel $c$. The $n^{th}$ peer in the channel $c$ is marked as $p^n_c$.

In accordance with measurement studies of existing peer-assisted systems (e.g., [15]), we assume that peer upload capacities are the only bottlenecks in the system, commonly referred to the upload-constrained scenario [12]. Denote $V^n_c$ and $v^n_c$ the upload capacity and the normalized upload capacity (divided by the streaming rate $R$) for the $n^{th}$ peer in channel $c$, respectively. Similar to the upload capacity, the normalized download bandwidth for the $n^{th}$ peer in channel $c$ and the normalized server upload capacity for channel $c$ are denoted as $d^n_c$ and $v^n_s$, respectively. For ease of reference, key notations introduced are summarized in Table 4.1.

4.2 User Perceptual Quality

The model introduced in the previous section sets the stage for us to analyze the problem quantitatively. As our consistent goal is to improve the aggregate user perceptual quality, we seek to find out how user perceptual quality is coupled with peer download bandwidth. In order to bridge these two quantities, an imperative intermediate factor, fraction of late packets, is introduced.

4.2.1 Fraction of Late Packets

Consider a pair of upstream and downstream peers in the system, where the sender transmits the media content to the receiver at the streaming rate $R$. Assume a packet is the unit for transmission in the system. In the absence of delay jitter and packet loss, media
4.2. USER PERCEPTUAL QUALITY

packets can be played as they are received, resulting in a smooth playback. However, in
the presence of delay jitter, inter-arrival times will vary and lead to late packets. Late
packets are those packets that arrive at the receiver later than their playback deadline.
In the presence of packet loss, some of the packets will not even arrive at the downstream
peer and need to be retransmitted. In the following analysis, we will only consider delay
jitter, as the measurement study [27] shows that 98.9% of late packets are due to delay
jitter, according to their large-scale study of Internet dynamics and its effect on video
streaming. Implementation specific issues such as video encoding and decoding are not
discussed.

Intuitively, higher download bandwidth of a peer may mitigate the impact of delay
jitter, resulting in a smoother playback. We seek to quantitively derive this relation as
follows. For the brevity of presentation, the aggregate rate that upstream peers transmit
to a specific peer \( p \) is denoted as \( \lambda \). The spacing \( T \) between consecutive packets at this
specific peer \( p \) is given by the inverse of the rate, \( i.e., T = 1/\lambda \). Previous studies have used
the Poisson process or the interrupted Poisson process, with mean \( \lambda \), for the modeling of
packet arrivals. The exponentially (hyperexponentially) distributed inter-arrivals, that
are implied by the Poisson (interrupted Poisson) process, have some properties that limit
their value as models of the real inter-arrivals of periodic streams that have been shaped
by delay jitter. First, the exponential distribution is not symmetrical around its mean,
which contradicts with many measurement results, \( i.e., [25] \). Second, the exponential
distribution is much more variable than measured inter-arrival distributions.

In order to better characterize packet inter-arrivals, a more regular distribution, \( k \)-
Erlang distribution, is introduced and verified in [26]. A \( k \)-Erlang random variable \( X \)
with mean \( T \) is given by \( X = \sum_{i=1}^{k} Y_i \), where \( Y_i \), for \( 1 \leq i \leq k \), is an exponentially
4.2. USER PERCEPTUAL QUALITY

distributed random variable with mean $T/k$. For the random variables $X$ and $Y$ we have

$$
E\{X\} = k \cdot E\{Y\} = T \\
Var\{X\} = k \cdot Var\{Y\} = \frac{T^2}{k}
$$

We apply the $k$-Erlang distribution to characterize packet inter-arrivals in our analysis. However, the intrinsic dynamic nature of P2P systems leaves the connections between peers less stable compared with client-server systems. The amount of time peers searching for new upstream peers is non-negligible, and should be taken into consideration in the packet inter-arrival model. Let $\alpha = \frac{\tau}{T_s}$, where $\tau$ denotes the average amount of time, during which peers have no valid connections and search for new upstream peers. $T_s$ denotes the average duration of downloading peers. As a result, the average packet inter-arrival time equals to $(1 + \alpha) \cdot T$ when taking peer dynamics into consideration. Then, we have the following:

**Proposition 4.1** For any specific peer $p^n_c$, the fraction of late packets $q$ during its entire downloading session is given by

$$q = \begin{cases} 
1 - \frac{d^n_c}{(1+\alpha)} & 0 \leq d^n_c < (1 + \alpha) \\
0 & d^n_c \geq (1 + \alpha)
\end{cases} \quad (4.1)
$$

where $d^n_c$ is the normalized download bandwidth of peer $p^n_c$.

**Proof:** The local cache of any specific peer $p^n_c$ can be regarded as a queue, where media packets are stored in it. Denote $D^n_c$ the downloading rate for peer $p^n_c$. Follow the discussions in the preceding several paragraphs, the input of this queue conforms to the $k$-Erlang distribution, with average packet inter-arrivals $1/D^n_c \cdot (1 + \alpha)$, while media
packets are drained from the queue at the constant streaming rate $R$. Additionally, each peer uses a pre-determined and fixed amount of disk space to cache recently downloaded media segments in VoD systems. For example, in the PPLive system, the size of such a media cache may be in the order of 1 GB, which exceeds the size of most media files, and can be regarded as $\infty$ for the queue. Hence, peer $p^n$’s local cache can be modeled as an $E_k/D/1/\infty$ queueing system. According to Theorem 2 in [34], the probability that the server is idle equals $1 - \lambda \beta$ when $\lambda \beta < 1$, and 0 otherwise. $\lambda$ and $\beta$ denote the average arrival rate and average service time, respectively. When the server is idle, it implies that packets arrive late than their playback time, and the announced result is obtained.  

4.2.2 Link Between User Perceptual Quality and Fraction of Late Packets

We now proceed to find out how the fraction of late packets is related to user perceptual quality. User’s perception, understood as information assimilation, is a rather subjective object, and it tends to be difficult to analyze quantitatively. However, measurement studies [8, 14], based on experiments and performance evaluation metrics, like eye tracking or subjective quality opinion score, reveal some crucial characteristics how these two quantities are linked. Some points are summarized as follows. First, the presence of even low amounts of delay jitter results in severe degradation in perceptual quality. However, higher amounts of delay jitter do not degrade perceptual quality proportionally. Second, the perceived quality of low temporal aspect video is not impacted in the presence of delay jitter as much as high temporal aspect video. Third, a strong correlation exists between the average number of quality degradation events and the average user quality rating recorded.
4.3. **OPTIMIZATION FRAMEWORK**

Although no strict correlation between the fraction of late packets and user perceptual quality is drawn, observations made from these measurement studies are sufficient for us to design a rough fitting function to characterize their relations. According to the points raised in [8, 14], we propose the function as the following:

\[
U(q) = U_{max} \exp\left(-\frac{q}{\gamma}\right), \quad 0 \leq q \leq 1
\]  

(4.2)

where \(U\) and \(q\) denote the user perceptual quality and fraction of late packets, respectively. \(\gamma\) indicates the impact of late packets on the user perceptual quality. According to the second point summarized in the paragraphs above, \(\gamma\) is much larger in low temporal aspect videos, and smaller in high temporal aspect videos. \(U_{max}\) denotes the maximal value of the user perceptual quality.

Combining Equations (4.1) and (4.2), we have the relation between peer’s download bandwidth and user perceptual quality as follows.

\[
U(d) = \begin{cases} 
U_{max} \exp\left(-\frac{1}{\gamma} + \frac{d}{(1+\alpha)\gamma}\right) & 0 \leq d < (1 + \alpha) \\
U_{max} & d \geq (1 + \alpha) 
\end{cases}
\]

This is also one of the utility functions we seek to maximize in the optimization framework formulated in the next section. An illustrative example on how to derive this function via Equations (4.1) and (4.2) is shown in Fig. 4.4.

### 4.3 Optimization Framework

We present our optimization framework in this section. First, we consider the scenario in which a fix amount of upload capacity for each channel is allocated from the server,
4.3. OPTIMIZATION FRAMEWORK

Figure 4.4: An example illustrates the way of deriving the relation between user perceptual quality and peer’s download bandwidth through an intermediate factor, the fraction of late packets. The top left figure shows the relation between download bandwidth and the fraction of late packets, while the top right figure presents the relation between user perceptual quality and the fraction of late packets.

namely $v^c$ is predetermined and constant, $\forall c$. The objective we seek to maximize is the aggregate average user perceptual quality across all channels, which is

$$\max \sum_{c \in C} U(d_c)$$

(4.3)

where $d_c$ denotes the average download bandwidth in channel $c$, that is,

$$d_c = \frac{1}{|N_c|} \sum_{n \in N_c} d^n$$

Within each channel, we only consider the average performance among all the peers. The bandwidth allocation and scheduling problem within each channel, which already received a substantial amount of research focus, is out of the scope of this paper. With carefully designed peer selection and segment selection algorithms, the upload bandwidth from both peers and the dedicated server is proved to be efficiently utilized [35, 42].

Define the set $A^c = \{p^n_{c'} \mid peer p^n_{c'} has complete file c in its local cache, \forall c' \in C \setminus c\}$ as
the set of peers, excluding peers in channel $c$, holding media file $c$ in their local caches. Then the throughput constraint in channel $c$ can be characterized as follows:

$$\sum_{n \in \mathcal{N}_c} d_n^c \leq v^s_c + \sum_{n \in \mathcal{N}_c} u_{n,c}^c + \sum_{p_{c'}^n \in \mathcal{A}_c} u_{n,c}^{n,c'}, \quad \forall \ c \in \mathcal{C}$$

(4.4)

where $u_{n,c}^{n,c'}$ denotes the amount of bandwidth that peer $p_{c'}^n$ contributes to redistribute media file $c$. For each $u_{n,c}^{n,c'}$, it obeys upload bandwidth constraints as follows:

$$\sum_{c \in \mathcal{C}} u_{n,c}^{n,c'} \leq v_n^{c'}, \quad \forall \ n \in \mathcal{N}_{c'}, \ \forall \ c' \in \mathcal{C}$$

(4.5)

$$0 \leq u_{n,c}^{n,c'} \leq v_n^{c'}, \quad \forall \ n \in \mathcal{N}_{c'}, \ \forall \ c, c' \in \mathcal{C}$$

(4.6)

Consequently, this single objective optimization problem becomes a maximization problem, with the objective (4.3), subject to constraints (4.4), (4.5) and (4.6).

However, the potential solutions that could have obtained through this optimization framework is sub-optimal. This bandwidth allocation problem is based on the assumption that the server allocates a fix amount of upload capacity for each channel. Through the discussion in Sec. 4.1, we observe that server bandwidth can be further optimized without deteriorating system performance. For the consistency of notations, denote $u_s^c$ the upload bandwidth allocated for channel $c$ from the server. Then the second objective involved in this problem is

$$\min_{c \in \mathcal{C}} \sum_{c \in \mathcal{C}} u_s^c$$

(4.7)
with the following constraint

$$0 \leq u_s^c, \quad \forall c \in \mathcal{C} \tag{4.8}$$

Replacing $v_s^c$ with $u_s^c$ in constraint (4.4), and combining the original optimization framework with the second objective (4.7) and the constraint (4.8), we obtain the complete version of this multi-objective optimization problem, which is summarized as follows:

$$\begin{align*}
\max & \quad \sum_{c \in \mathcal{C}} U(d_c) \\
\min & \quad \sum_{c \in \mathcal{C}} u_s^c \\
\text{s.t.} & \quad \sum_{n \in \mathcal{N}_c} d_c^n \leq u_s^c + \sum_{n \in \mathcal{N}_c} u_{n,c} + \sum_{p_{c'} \in \mathcal{A}^c} u_{n,c'}, \quad \forall c \in \mathcal{C} \\
& \quad \sum_{c \in \mathcal{C}} u_{n,c} \leq v_n^c, \quad \forall n \in \mathcal{N}_{c'}, \quad \forall c \in \mathcal{C} \\
& \quad 0 \leq u_{n,c} \leq v_n^c, \quad \forall n \in \mathcal{N}_{c'}, \quad \forall c, c' \in \mathcal{C} \\
& \quad 0 \leq u_s^c, \quad \forall c \in \mathcal{C}
\end{align*}$$

### 4.4 Algorithms for Multi-Objective Programming

Unlike the unique optimal solution in single-objective optimization problems, a number of optimal solutions arise in multi-objective optimization because of trade-offs between conflicting objectives. In order to better understand and design algorithms for multi-objective programming, we need to make several definitions first. Consider a general multi-objective optimization, where $F$ objective functions are defined as $f_i(\cdot), i = 1, \ldots, F$. In order to cover both minimization and maximization of objective functions, we use the operator $\prec$ between two solutions $i$ and $j$ as $i \prec j$ to denote that solution $i$
4.4. ALGORITHMS FOR MULTI-OBJECTIVE PROGRAMMING

is better than solution $j$ on a particular objective.

**Definition 4.1** A solution $x$ is said to dominate the other solution $y$, if both conditions 1 and 2 hold:

1. Solution $x$ is no worse than $y$ in all objectives, namely $f_i(x) \not\preceq f_i(y), \forall i \in \{1, 2, \ldots, F\}$.

2. Solution $x$ is strictly better than $y$ in at least one objective, namely $\exists j \in \{1, 2, \ldots, F\}$ where $f_j(x) \prec f_j(y)$.

**Definition 4.2 (Non-dominated set)** Among a set of solutions $\mathcal{P}$, the non-dominated sets of solutions $\mathcal{P}'$ are those that are not dominated by any member of the set $\mathcal{P}$.

**Definition 4.3 (Globally Pareto-optimal set)** The non-dominated set of the entire feasible search space $\mathcal{S}$ is the globally Pareto-optimal set.

In multi-objective optimization, there are clearly two goals. Progressing towards the Pareto-optimal front is certainly an important goal. However, maintaining a diverse set of solutions in the non-dominated front is also essential. Since both goals are important, an efficient multi-objective optimization algorithm must satisfy both of them. It is important to realize that both of these tasks are somewhat orthogonal to each other. The achievement of one goal does not necessarily achieve the other goal.

Some classical methods, *i.e.*, the weighted sum method, convert multiple objectives into a single objective, and exploit traditional methods to solve it efficiently. Unfortunately, the outcomes of such optimization strategies depend on the fix-ups, such as the chosen weights in the weighted sum method. Multi-objective optimization for finding multiple Pareto-optimal solutions eliminates all such fix-ups and can, in principle, find a
set of optimal solutions corresponding to different weights. Although only one solution is needed for implementation, a knowledge of such multiple optimal solutions may help a designer to compare and choose a compromised optimal solution.

Another problem we want to discuss here is the constraint-handling techniques used with multi-objective programming algorithms. The general constraint-handling approaches fall into four categories, penalty functions, special representations and operators, separation of constraints and objectives, and hybrid methods. Among these approaches, the most representative one is the penalty function approach. The idea of this approach is to transform a constrained optimization problem into an unconstrained one by adding (or subtracting) a certain value to/from the objective function based on the amount of constraint violation present in a certain solution. The modern constraint-handling approaches that use penalty functions include self-adaptive fitness formulation [13], AS-CHEA [18] and stochastic ranking [31]. Unlike penalty functions which combine the value of the objective function and the constraints of a problem to assign fitness, the separation of constraints and objectives approach handle constraints and objective separately. The elitist non-dominated sorting genetic algorithm, we introduces below, applies this approach to handle constraints. The introduction of the other two approaches is omitted here and interested readers are referred to relevant literatures.

4.4.1 Elitist Non-Dominated Sorting Genetic Algorithm

Currently, most multi-objective algorithms build on prior work on evolutionary algorithms, whose roots can be traced back to genetic algorithms and evolutionary programming. Among those evolutionary algorithms, the most popular one is the Elitist Non-Dominated Sorting Genetic Algorithm (NSGA-II) proposed by Deb et al. [10], and
**Algorithm 1 Crowding-Sort Algorithm**

1. Given a set of solutions \( \mathcal{P} \), where \( N_p \) denotes the number of solutions in set \( \mathcal{P} \), namely \( N_p = |\mathcal{P}| \). The number of objective functions is 2.
2. Initialize crowding distance \( d_i = 0 \) for each solution \( i \) in the set.
3. for \( k = 1 \) to 2 do
4. Sort the set in descending order of \( f_k \), or find the sorted indices vector: \( I^k = \text{sort}(f_k, >) \).
5. Assign large values to the boundary solutions
   \[
   d_{I^k_1} = d_{I^k_1} + \infty \\
   d_{I^k_{N_p}} = d_{I^k_{N_p}} + \infty
   \]
6. for \( l = 2 \) to \( N_p - 1 \) do
7. \[
    d_{I^k_l} = d_{I^k_l} + \frac{f_{k(I^k_{l+1})} - f_{k(I^k_{l-1})}}{f_{k_{\max}} - f_{k_{\min}}}
    \]
8. end for
9. end for

we will briefly describe it in this section.

Genetic algorithms (GAs) are search and optimization procedures that are motivated by the principles of natural genetics and natural selection. Some fundamental ideas of genetics are borrowed and used artificially to construct search algorithms that are robust and require minimal problem information.

Basic operations in genetic algorithms include reproduction, crossover and mutation. The primary objective of the reproduction operator is to make duplicates of good solutions, and eliminate bad solutions in a population, while keeping the population size constant. A little thought will indicate that the reproduction operator only makes more copies of good solutions at the expense of not-so-good solutions, and the responsibility of creation of new solutions is carried by crossover and mutation operators. The chosen solutions by the reproduction operator are put into a mating pool. Two solutions are picked from the pool at random and some portion of the solutions are exchanged between
the solutions to create two new ones, which is called crossover. The mutation operator randomly alters part of the solution locally to hopefully create a better solution. Details of these operators are omitted here, and interested readers are referred to [9].

Although all GAs are based on these three basic operators, NSGA-II differentiates from other algorithms in that it utilizes an explicit diversity-preserving mechanism, which is called Crowded Tournament Selection Operator.

**Definition 4.4 (Crowded Tournament Selection Operator)** A solution \( x \) wins a tournament with another solution \( y \) if any of the following conditions is true:

1. Solution \( x \) has a better non-domination rank than solution \( y \).

2. They have the same rank, but solution \( x \) has a better crowding distance than solution \( y \).

The underlying idea here is to give higher priority to those solutions who reside in the less crowded region, if they are in the same non-dominated front. The crowding distance can be derived through the crowding-sort algorithm, as shown in Algorithm 1. One iteration of the NSGA-II algorithm is summarized in Algorithm 2. This algorithm is initialized by randomly generating a set of solutions \( P_0 \), where \( N_p = |P_0| \).

The computational overhead of NSGA-II algorithm is overwhelming in large-scale systems, such as peer-assisted VoD systems. Step 4 requires a non-dominated sorting of population of size \( 2P_n \), where \( P_n \) is the initial population size, and it requires almost \( O(P_n^2) \) computations. Generally, the size of the population is comparable to the number of variables, which is equal to \( |C| \cdot N \). The crowding sorting algorithm in Step 9 requires \( O(P_n \log P_n) \) computations. Thus, the overall complexity of NSGA-II for one single iteration is almost \( O(P_n^2) \), which is equivalent to \( O(|C|^2N^2) \).
**Algorithm 2** NSGA-II Algorithm (One Iteration)
1: Create offspring population $Q_t$ from $P_t$ by using the crowded tournament selection, crossover and mutation operators.
2: Combine parent and offspring populations and create $R_t = P_t \cup Q_t$.
3: Perform a non-dominated sorting to $R_t$ and identify different non-dominated fronts: $F_i$, $i = 1, 2, \ldots, \text{etc.}$
4: Set new population $P_{t+1} = \emptyset$ and a counter $i = 1$.
5: **while** $|P_{t+1}| + |F_i| < N_p$ **do**
6: 
   
   
   $P_{t+1} = P_{t+1} \cup F_i$  
   
   $i = i + 1$

7: **end while**
8: Perform the crowding-sort algorithm described in **Algorithm 1**., and include the most widely spread $(N_p - |P_{t+1}|)$ solutions by using the crowding distance values in the sorted $F_i$ to $P_{t+1}$.

### 4.4.2 Greedy Heuristic Algorithm

In order to overcome the prohibitive computational complexity incurred by NSGA-II, we propose a greedy heuristic algorithm. The underlying idea of this algorithm is that the surplus upload capacity is always distributed to the best under-provisioned channel, that is the channel with minimal difference between the aggregate upload capacity and aggregate minimal demand $(1 + \alpha)R|N_c|$. As the user perceptual quality as a function of download bandwidth is exponential-like, it is improved more significantly when the aggregate upload capacity is closer to the aggregate minimal demand.

This greedy heuristic algorithm is composed of two parts, peers’ part and servers’ part. In the peers’ part of this algorithm, a probability vector $R_i$ is generated and distributed to individual peers in each channel $i$. Peers allocate their upload bandwidth based on the value of $R$. While in the servers’ part, the way of allocating the server upload capacity is determined.
Algorithm 3 \textit{Greedy Heuristic Algorithm – Peer Side}

1: Gather clustering coefficient $p_i$ for each channel $i$, that is the average proportion of peers in channels other than $i$ that have media file $i$ stored in their local cache.

2: Initialize the probability vector $R_i$, for channel $i$, with a $|C|$ by 1 zero vector while the $i$th entry equals to 1 exclusively. Denote $r^j_i$ the $j$th entry in vector $R_i$.

3: Define $\Delta_c$ the difference between the aggregate upload bandwidth from peers and the aggregate minimal demand $(1+\alpha)R|\mathcal{N}_c|$ in channel $c$. Find the sorted index vector: $I = \text{sort}(\Delta, >)$ and the index $\omega$, where $\Delta_{I_i} \geq 0$, $\forall i \leq \omega$ and $\Delta_{I_i} < 0$, otherwise.

4: \textbf{for} $i = 1$ to $\omega$ \textbf{do}

5: \hspace{1em} \textbf{for} $j = \omega + 1$ to $|C|$ \textbf{do}

6: \hspace{2em} $\Delta = \min(p_{I_j} \cdot \Delta_{I_i}, |\Delta_{I_j}|)$
\hspace{2em} $\Delta_{I_i} = \Delta_{I_i} - \Delta$
\hspace{2em} $\Delta_{I_j} = \Delta_{I_j} + \Delta$

7: \hspace{1em} Update probability vector $R_{I_i}$ with
\hspace{2em} $r^I_{I_i} = \frac{\Delta}{\overline{u}_{I_i} |\mathcal{N}_{I_i}|}$
\hspace{2em} $r^I_{I_i} = r^I_{I_i} - \frac{\Delta}{\overline{u}_{I_i} |\mathcal{N}_{I_i}|}$

8: \hspace{1em} \textbf{end for}

9: \hspace{1em} \textbf{end for}

10: Distribute the probability vector $R_i$ to individual peers in channel $i$. 
At the beginning of the peers’ part, a clustering coefficient $p_i$ is initialized for each channel $i$. The clustering coefficient $p_i$ indicates the average proportion of peers in channels other than $i$ that have media file $i$ stored in their local cache. This value can be derived by collecting the cache availability maps of each peer. As such, we choose to utilize a centralized tracking server — typically used in real-world peer-assisted VoD systems — to centrally keep track of cache availability maps in all peers in our simulations. Due to latencies of collecting such information, estimates of clustering coefficient on the tracking server may not be accurate, but it is already the best one can do in practice. In fact, a more accurate setting is to calculate the correlation coefficient for any pair of channels in the system, however, only the homogeneous case is considered here and in the following simulations.

The probability vector $\mathbf{R}$ is the most important parameter in this algorithm. For each channel $i$, a probability vector $\mathbf{R}_i$, with each entry $r_{ij}^i$ indicating the probability for peers in channel $i$ to respond to requests for media file $j$, is distributed to all peers in this channel, and peers make decisions individually. The benefit of this scheme is two fold. First, by letting the peers make their own decisions individually, the control overhead is quite low in our algorithm. The control messages only occur when the optimization is re-solved. Second, this algorithm copes better with the situation that peers download asynchronously. In the asynchronous case, it is infeasible to allocate the control vector to individual peers in a centralized manner.

The way of dynamically allocating servers’ upload capacity is similar to that of peers, which is also based on the idea that the surplus upload capacity is always distributed to the best under-provisioned channel. Due to space constraints, a detailed explanation is omitted here. The peer side and server side of this greedy heuristic algorithm is
4.4. ALGORITHMS FOR MULTI-OBJECTIVE PROGRAMMING

Algorithm 4 Greedy Heuristic Algorithm – Server Side

1: Initialize the amount of server upload bandwidth $\beta u_s$ for each channel in proportion to the number of peers in each channel. $u_s$ indicates the total server upload capacity, and $\beta$ is the aggressive factor, which follows the constraint $0 \leq \beta \leq 1$. Define $u_{\text{residual}} = (1 - \beta)u_s$.

2: Re-sort $\Delta$, updated by the first part of the algorithm. Find the sorted index vector: $I = \text{sort}(\Delta, >)$ and the index $\omega$, where $\Delta_i \geq 0$, $\forall i \leq \omega$ and $\Delta_i < 0$, otherwise.

3: for $i = \omega + 1$ to $M$ do

4: $\Delta = \min(u_{\text{residual}}, |\Delta_i|)$

5: $u_{\text{residual}} = u_{\text{residual}} - \Delta$

6: $\Delta_i = \Delta_i + \Delta$

5: end for

summarized in Algorithm 3 and Algorithm 4, respectively.

We proceed to conduct numerical analysis in order to verify the performance of the greedy heuristic algorithm in Matlab. We generate random topologies of 1000 peers, with each peer connected with a varying number of neighbors. The buffer map for each peer is also randomly generated according to the clustering coefficients. The randomly constructed system can be regarded as a snapshot of the real world system. We run both NSGA-II and the greedy heuristic algorithm on the “static” system, the result of which is presented in Fig. 4.5. At first glance, it might be surprisingly to find that the greedy heuristic performs even better than NSGA-II. However, evolutionary algorithms are categorized as global search heuristics, and there is no guarantee that the Pareto-optimal front generated by the algorithm is globally optimal. We run the tests for several times under different parameter settings, and similar results are obtained. Through numerical analysis, we conclude that the greedy heuristic performs as well as — if not better than — NSGA-II for this specific problem.
4.5 Performance Evaluation

To further evaluate the performance of the greedy heuristic algorithm in multi-channel peer-assisted VoD systems, we conduct comprehensive simulations and present results in this section. We use the prefetching scheme in isolated channel design as a “benchmark”, in which peers keep downloading future content if there exists surplus upload bandwidth from peers within the same channel. In the figures showing simulation results, we use “MC” and “IC” to denote the greedy heuristic in multi-channel (MC) systems, and the prefetching scheme in isolated channel (IC) design, respectively. The numbers next to “MC”s or “IC”s indicate the channel indices.

Unless otherwise specified, we simulate systems that consist of random topologies of 6000 peers, with each peer connected with a varying number of neighbors. The total
number of channels in the system is set to 3, and each channel consists of 2000 peers on average. The mean of the normalized upload capacities for each channel are set to 0.5, 1, 2, respectively. In other words, channel 1 is always running in a deficit state, as the aggregate demand exceeds the aggregate supply from peers, while channel 3 is running in a surplus state. It is worth noting that the performance of channel 2 is the most unpredictable, as the system fluctuates between surplus and deficit state randomly.

Recall that $\alpha$ and $\gamma$ denote the overhead for peers searching and establishing connections with new neighbors during its download sessions, and the impact of late packets on the user perceptual quality, respectively. In our simulations, we set that peers can find new upstream neighbors once they have no valid connections. Thus, the overhead is negligible in our simulations, namely $\alpha = 0$. With respect to the parameter $\gamma$, it should be customized to each individual media file and can only be obtained through statistical experiments. Thus, we use the fraction of late packets, instead of user perceptual quality, as a performance metric in the following simulations.

### 4.5.1 Fraction of Late Packets vs. Server Upload Capacities

Since two of our consistent objectives are to improve user perceptual quality and mitigate the server bandwidth cost, we first investigate the correlations between these two values. The knowledge of such multiple solutions can help us to compare and choose a compromised solution in subsequent simulations, and implementation of systems in real world as well.

Fig. 4.6 shows the effects of server upload capacities on the fraction of late packets in each individual channel. We vary the value of server upload capacity, and then compute the average fraction of late packets over 500 time slots, the results of which are shown
in the figure. Unless otherwise specified, the Y-axis in subsequent figures indicate the fraction of late packets in each individual channel.

We observe that the difference between our greedy heuristic algorithm and prefetching scheme is more pronounced when the upload capacity from servers is limited. As we increase the server upload capacity, the difference becomes insignificant. Furthermore, it is most revealing that the performance of our greedy heuristic algorithm remains almost constant with the increase of the server upload capacity. In multi-channel systems, the greedy heuristic can efficiently steer the surplus upload bandwidth from one channel to those channels in deficit state. As a result, server bandwidth cost can be significantly mitigated compared with prefetching schemes in the isolated channel design.

![Normalized Server Upload Capacity](image)

**Figure 4.6:** Average fraction of late packets under various server upload capacities. $|C|$ is set to 3, and each channel consists of 2000 peers on average. We set $k$ to 100, $k_c$ to 12, clustering coefficient to 0.1 and mean of the normalized upload capacities for each channel to 0.5, 1, 2, respectively. Please refer to Fig. 4.7 for the legend.
4.5.2 Effects of Clustering Coefficients

After investigating the relation between fraction of late packets and server upload capacities, we proceed to inspect the impact of clustering coefficients on the performance of our greedy heuristic algorithm in multi-channel systems. From this point on, the main performance metric of concern in our comparisons is the fraction of late packets. The normalized server upload capacity is set to be 600 in both multi-channel and isolated channel systems in the following simulations.

Fig. 4.7 presents the fraction of late packets while varying the clustering coefficient from 0.1 to 0.6. Without loss of generality, the clustering coefficients are set to be identical for each individual channel. From Fig. 4.7, we can clearly see that the performance of the greedy heuristic algorithm improves significantly as the clustering coefficients increase. When the clustering coefficient is larger than 0.4, the fractions of late packets in all three channels approach 0.

4.5.3 Effects of Neighborhood Sizes

Finally, we shift our interest to the impact of neighborhood sizes on the system performance. Rather than all neighbors belonging to the same channel in the isolated channel design, the neighbors of any individual peer in multi-channel systems are composed of two parts, peers from the same channel and peers from different channels.

Fig. 4.8 presents the fraction of late packets while varying the neighborhood sizes within the same channel. The topology of the system is set to be a random graph, while the average size of the neighborhood within the same channel ranges from 40 to 160. As the peers are able to connect to more peers simultaneously, they are more likely to find useful upstream peers and download segments from those peers. Consequently, with the
4.5. PERFORMANCE EVALUATION

Figure 4.7: Average fraction of late packets under various clustering coefficients. $|C|$ is set to 3, and each channel consists of 2000 peers on average. We set $k$ to 100, $k_c$ to 12, normalized server upload capacity to 600 and mean of the normalized upload capacities for each channel to 0.5, 1, 2, respectively.

increase of size of the neighborhood within the same channel, the fractions of late packets decrease slightly for all channels in both systems.

In order to show the impact of the neighborhood size from different channels, we set the size of the neighborhood from the same channel to 100, vary the value of cross-channel neighborhood size, and compute the average fraction of late packets over 500 time slots, the result of which is shown in Fig. 4.9.

We can see that the performance from different channels presents slightly different behavior in response to a varying number of neighbors. While we increase the neighborhood size from different channels, more peers in channels other than channel 1 are able to contribute their upload resources to peers in channel 1, which is in deficit state. Hence, the average fraction of late packets decreases significantly for peers in channel
As channel 2 fluctuates between surplus and deficit states randomly, the fraction of late packets also fluctuates, and there is no clear trend for the performance of channel 2. With respect to channel 3, its performance is almost constant under different sizes of the neighborhood. Furthermore, the impact of the neighborhood size from different channels is not that significant as the impact of clustering coefficients as shown in Fig. 4.7 for the greedy heuristic in multi-channel systems.

![Graph showing average fraction of late packets under various neighborhood sizes](image)

Figure 4.8: Average fraction of late packets under various number of neighbors within the same channel. $|C|$ is set to 3, and each channel consists of 2000 peers on average. We set $k_c$ to 12, clustering coefficient to 0.1, normalized server upload capacity to 600 and mean of the normalized upload capacities for each channel to 0.5, 1, 2, respectively. Please refer to Fig. 4.7 for the legend.

### 4.6 Summary

In this Chapter, we apply a novel approach — bandwidth allocation across multiple channels — to effectively utilize surplus upload bandwidth capacity from peers in multi-channel peer-assisted VoD systems. Under this approach, surplus upload bandwidth from
one channel can be contributed to redistribute media files in those channels in deficit state. With the objective of optimally allocating upload bandwidth, we form a unifying multi-objective optimization framework to optimize two conflicting objectives, user perceptual quality and server bandwidth cost, concurrently. We propose an approximation algorithm and prove its efficiency by comparing its performance with conventional evolutionary algorithms. Complementary simulations further corroborates the effectiveness of our algorithm.
Chapter 5

Concluding Remarks and Future Work

In this thesis, we answered two questions in the design of peer-assisted VoD systems, with the objective of improving the overall system performance, in terms of server bandwidth cost and user perceptual quality. In Chapter 3, we investigated the cache replacement algorithm design in systems where peer caches are passively managed. With rigorous theoretical analysis and complementary simulations, we showed that simple heuristics are good enough to be used as cache replacement algorithms in peer-assisted VoD systems. We proceeded to inspect the surplus upload bandwidth allocation problem in multi-channel peer-assisted VoD systems in Chapter 4. We formulated a multi-objective optimization framework for this problem and proposed an approximation algorithm to solve it efficiently. With realistic simulations, we presented its performance, and concluded that it enjoyed consistently better performance compared with conventional prefetching schemes in the isolated channel design.

As a matter of fact, there exists a tremendous level of flexibility in the design space
of such peer-assisted VoD systems. Many algorithms and schemes may be refined so as to further improve the system performance. For example, network coding might be a remedy for the scheduling problem in the peer-assisted VoD systems. In real-world peer-assisted VoD systems, a small number of streaming servers are deployed in a geographically distributed fashion, and they collectively serve a large number of media files on demand. Due to the scarcity of server bandwidth, an important challenge is to design new and improved mechanisms to efficiently and fully utilize the upload bandwidth at each of the streaming servers. In traditional peer-assisted VoD protocols, the media content to be served is divided into segments. With each server storing only some of the segments in each media file, the load across different streaming servers can easily go out of balance, especially if the content provider does not have a priori knowledge about the popularity of different segments before deploying them to different servers. If the load is not balanced well across different servers, the upload bandwidth of some of the servers may be underutilized. When network coding is used, each segment is further divided into a considerable number of blocks with finer granularity. Rather than the entire segment, each streaming server stores linear combinations of original blocks within this segment with random coefficients, and these coded blocks in the same segment are of equal importance. Intuitively, since coded blocks from one segment can be stored in multiple servers, requests of this segment can be directed to several servers, which may lead to a more balanced workload across different servers.

Consequently, we argue that the design of peer-assisted VoD systems should be given its deserved research attention. We believe our work marks an important step towards optimization based protocol design for multi-channel peer-assisted VoD systems. However, substantial amount of work remains to be done in order for such kind of systems to
provide satisfactory performance under any demanding circumstances.
Bibliography


