An Optimized Video-on-Demand System: Theory, Design and Implementation

by

Hao Zhang

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering – Electrical Engineering and Computer Sciences
and the Designated Emphasis
in
Communication, Computation and Statistics

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Fall 2012

An Optimized Video-on-Demand System: Theory, Design and Implementation

Abstract

An Optimized Video-on-Demand System: Theory, Design and Implementation

by

Hao Zhang
Doctor of Philosophy in Engineering – Electrical Engineering and Computer Sciences
and the Designated Emphasis
in
Communication, Computation and Statistics

University of California, Berkeley

Professor Kannan Ramchandran, Chair

Internet on-demand video traffic has seen explosive growth in recent years. This brings a number of challenges in deploying video-on-demand services at large scale. The first challenge has to do with the enormity of the video catalog size. Traditionally, content providers replicate the entire video library in different locations, but this is wasteful and non-scalable as the video catalog size expands. The second challenge comes with the increase in video quality, which calls for efficient utilization of the scarce network bandwidth resources that continue to be economically expensive to expand. The emergence of different device modalities, including smart-phones, high-definition and 3D TV, tablets and etc poses another challenge in designing efficient systems and algorithms that cater to all device characteristics and user needs.

In this dissertation, we aim to present a general approach to designing, optimizing and architecting a video-on-demand system. Our approach considers the practical constraints of disk space, network link bandwidth, and node connection degree bound. In general, the joint optimization problem is combinatorially difficult. To tackle this, we first design a simple fractional storage architecture, which uses a class of regeneration codes that fluidifies the content, thereby enabling a distributed content placement and link rate allocation algorithm. We show that by storing only a fractional of the entire catalog everywhere, the system is able to fully support user demand at large scale. Second, we develop a Markov approximation technique to solve the problem of topology selection under node degree bound using a simple distributed algorithm. We prove that our algorithm achieves close-to-optimal solution, which we verify using extensive realworld trace simulations.

On the system side, we show extensive results to test the algorithm's scalability and robustness to changes in user dynamics and demand patterns. We show that our solution achieves high utilization of cache nodes storage and bandwidth resources, and automatically learns and caches the video according to the demand patterns. We observe that there exists a complex interplay between disk space, network bandwidth and node degree bound. We also present guidelines to important practical design choices including caching update intervals, demand prediction and provisioning. We also demonstrate the feasibility and efficiency of our design choice by building and experimenting a prototype system at Berkeley.

I dedicate this dissertation to my family, particularly to my parents Liyan Zhang and Wenhua Ye who believe in diligence, science, art, and the pursuit of academic excellence; to my wife Jian Wang for continuously supporting our family and giving me tremendous care and encouragement; and to our newborn son Lucas Zhang, who brings endless joy to our lives. I must also thank my in-laws who have helped so much with taking care of Lucas and have given me their fullest support.

Contents

List of Figures

List of Tables

Acknowledgments

It is a pleasure to thank the many people who made this thesis possible.

It is difficult to overstate my gratitude to my Ph.D. supervisor, Dr. Kannan Ramchandran. With his enthusiasm, his inspiration, and his great patience to explain things clearly and simply, he helped to make graduate study fun for me. Throughout the years, he not only provided lots of good ideas but also supported me with great personal encouragement and care. I feel greatly thankful to his company.

I also thank my co-advisor Professor Abhay Parekh for providing me with insightful ideas and guide to designing and building our prototype system. I am also grateful for others who have helped with my dissertation including Professor Avideh Zakhor, Professor Jean Walrand and Professor John Chuang, for their kind assistance with writing letters, giving wise advice, helping with my qual-exam, and so on.

I want to thank Professor Minghua Chen from the Chinese University of Hong Kong (CUHK) for sharing with me many interesting ideas, many of which helped build the very foundation of this thesis. I want to thank Ziyu Shao from CUHK for generously sharing his meticulous research and insights that supported and expanded my own work. I am thankful to Kangwook Lee from UC Berkeley for his implementation of the automated system simulator, and his hard work is also a great inspiration for me.

I am indebted to my many colleagues for providing a stimulating and fun environment in which to learn and grow. I am especially grateful to Salim El Rouayheb, Longbo Huang, Sameer Pawar, Nebojsa Milosavljevic, Kangwook Lee and Giulia Fanti from UC Berkeley. I am also thankful to Ermin Kozica who was a research visitor from KTH Royal Institute of Technology. I really enjoyed the discussions with them of both research and life, which have helped make my graduate study full of colors and joy.

I am grateful to the secretaries and assistants in the graduation division of the EECS department of UC Berkeley for helping the departments to run smoothly. I give my special thanks to Ruth Gjerde for assisting me in many different ways throughout my years at Berkeley.

Lastly, and most importantly, I wish to thank my family for all the sacrifices they made to make whom I am today. They supported me, cared me, and loved me. To them I dedicate this thesis.

Chapter 1

Introduction

1.1 Motivation

There is no doubt that Internet video traffic is dominating our network highway. The global IP traffic has increased eightfold over the past 5 years, expecting to cross the annual zettabyte (1 zettabyte $= 10^{21}$ bytes) line by year 2015 [3]. Among these, more than 60% is Internet video traffic, equaling to 3 trillion minutes (6 million years) of video content crossing the Internet each month in 2015. The explosive growth motivates the design and optimization of a scalable and robust video delivery system.

While video traffic takes a number of different formats, e.g., on-demand video, live streaming, video surveillance, video conferencing and video gaming etc., it is dominated by on-demand videos that accounts for 70% of the total video traffic. These systems provide users with a large catalog of videos to choose from, and allow them to watch any videos at any time at any place.

There are a number of challenges to deploy VoD services at a large scale. First, the popularity of videos exhibits long-tail properties, i.e., a large number of videos are requested infrequently. For example, while the number of videos on YouTube has reached 1 trillion, only a few thousand videos are “hot-hits”. Traditionally, VoD providers replicate the entire video library in different locations to circumvent problems such as unknown content popularity and system overload due to flash crowds. However, this is truly wasteful and avoidable. Despite the reduction in the cost of disk space, the rate of creation of content and the increasing demand for higher quality content will eventually outpace the ability of providers to scalably and economically add storage and replicate the entire catalog everywhere. This challenge motivates us to design optimization strategies that minimize the video storage while satisfying user demand.

The second challenge is to provide the best video quality to users located at different end-points with different bandwidth capacities and network conditions. Currently, HD video-on-demand has already surpassed standard definition (SD) at the end of 2011. By 2015, HD Internet videos, which typically stream at $3 - 10$Mpbs, will comprise 77% of the entire VoD traffic. However, the network capacities across different regions vary significantly, and the median worldwide download speed merely passes 600Kbps. It then becomes crucial to design algorithms that are “bandwidth-agnostic”, and that optimize streaming quality given arbitrary capacities of the underlay network.

The third dimension of growth (the first two being the number of videos and the quality of videos) is the number of modalities of video delivery. This is a new challenge not seen before

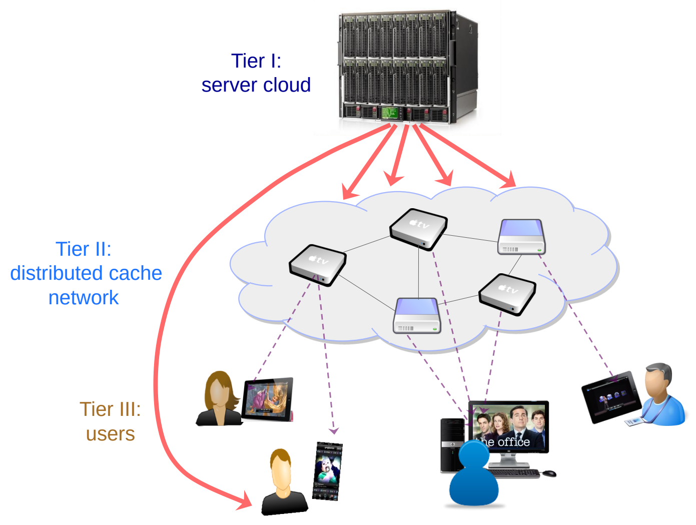


Figure 1.1: The three-tier architecture of the VoD system. The top tier is an oracle server cloud which serves as a vault for a potentially massive catalog of video content of demand to a large user base having an unknown demand distribution (bottom tier). The scalability requirement is addressed through a critical middle tier of distributed caches that sits in between the server cloud and the users, and helps distribute the content.

the recent blossom of video to mobile devices. While PC-originated traffic will grow at a rate of 33%, TVs, tablets, smart phones, and machine-to-machine traffic will have growth rates of 101%, 216%, 144% and 258%. To meet the demand at large scale, distributed approaches emerge, and it becomes crucial to operate within the resource capacities of individual video servers while optimizing delivery at a global scale. This brings new challenges and opportunities to different aspects of VoD systems design including video storage, video coding, traffic provisioning and load balancing.

1.2 System Model

To design a scalable and robust VoD content delivery solution that optimizes for storage usage, network bandwidth utilization and , we propose and design in this dissertation a VoD system based on distributed caching, which we have architected and built at Berkeley. The three-tiered system model is shown in Figure 1.1. The top tier is an oracle server cloud which serves as a vault for a potentially massive catalog of video content of demand to a large user base having an unknown demand distribution (bottom tier). The scalability requirement is addressed through a critical middle tier of distributed caches that sits in between the server cloud and the users, and helps distribute the content in a highly scalable and adaptive manner.

There are three major advantages of the above design abstraction.

1. The architecture is very general and it captures the essence of many concurrent systems. For example, in a content-distribution network (CDN), the cache layer represents the edge-servers

that are deployed close to the end-users. In a peer-to-peer system, the cache nodes are peers who have free system resources and are willing to share. In a wireless scenario, users can come from various modalities including smart phones, tablets and PCs. The cache nodes can take forms of base towers, femtocells and Wi-Fi access points.

2. The model allows us to model any arbitrary physical network that sits in between the cache nodes and the users. While the caches and users are often connected via a complete (or fully connected) graph in the Internet application layer, the underlying physical layer can be arbitrary and unknown. This model allows us to capture the physical network layer constraints between the caches and the users. The details are described in Chapter 3 and Chapter 4.

3. The architecture also eases the mathematical modeling of the problem. In practice, the physical incarnation of the top server tier nodes can take various forms, e.g. it can be a single backup server or a server farm (distributed archive servers). The existence of the top server tier ensures that the video streaming requirement be met to the users. The modeling details are presented in Chapter 3.

1.3 Contributions

Our approach to the design of the above VoD system is to first model it as an optimization problem and then propose distributed algorithms. We corroborate our theoretical analysis with the implementation of a realworld prototype system, which bridges the gap between theory and practice that is present in many existing works. Our modeling considers three practical constraints of storage capacity, network link capacity and node degree capacity, the joint optimization of which is missing in the literature. Specifically, the modeling is aimed to capture the three aforementioned challenges.

- First, we assume no single cache node can store more than a tiny fraction of the catalog content, as is motivated by the size of the catalog.
- Second, no single cache node has the bandwidth to satisfy the streaming demands of more than a small subset of all users. The physical network may have arbitrary and unknown link capacities, and the link rate allocation algorithm should be adaptable to any such constraints.
- Third, motivated by the increasing number of users and devices that may access the video delivery network, we assume each cache node has only a limited (the degree bound of) number of connections they can open up to the users. Conversely, each user is able to connect to only a limited number of cache nodes to retrieve his content. This is motivated by the fact that a single server/cache node can accept up to only a certain number of incoming requests at a time due to limits in its disk read/write speed, CPU capacity and network I/O bound.

Given the system architecture and the above modeling constraints, our system goal is to effectively use the collective resources of the cache network in minimizing the load on the top tier server, which has to bear the burden of covering the deficit between the supply (by the collective cache network) and the demand (by the users). We would like to emphasize that minimizing load on the oracle server is only a modeling choice that enables easy mathematical formulations. In practice, the objective can vary depending on the usage scenario.

Concretely, our problem is to minimize the server load while satisfying the users' streaming demands for their respective content by optimizing (a) what content should be stored on each cache node (while respecting the storage constraints)? (b) which users should each cache connect to (while respecting the connection degree bounds)? and (c) how should each cache node allocate its bandwidth among the users it connects to (while respecting the link rate capacity constraints in the physical network)? Further, in order to scale efficiently and have practical impact, we would like to solve this optimization problem in a distributed manner.

The distributed optimization problem has two difficulties, which we tackle one by one. The first difficulty is the content placement problem of deciding which movies should each cache node store to be of optimal system help. This problem is of a combinatorially explosive nature when there are a lot of choices of movies and cache nodes. We resolve this challenge by proposing a fractional storage architecture and the use of appropriately designed network codes. This helps convert this combinatorial non-multicast VoD problem into a tractable one, and enable a fully distributed algorithm that jointly optimizes the cache content placement and link rate allocation problem. This algorithm efficiently utilizes network resources and automatically learns the video demand distribution. The details are given in Chapter 3.

The second difficulty is the graph topology selection problem of deciding which users each cache should connect to (while satisfying the connection degree bound constraints). The graph topology selection problem is also a combinatorially difficult problem. To tackle this, we apply a Markov approximation technique by converting the topology selection problem into a Markov chain design problem, which allows us to design a simple algorithm that achieves close-to-optimal performance. In Chapter 4 we present the algorithm and provide its performance analysis. The topology selection results are of general interest and can be applied to other network system design problems of combinatorial nature.

At a high level, the joint design of appropriate network codes, together with topology selection, content placement and link rate allocation strategies, allows not only for a tractable solution to a hard combinatorial problem, but even admits a fully distributed algorithmic solution with provably close-to-optimal performance. In Chapter 5, we give theoretical proof that our solution achieves close-to-optimal solution and that the optimality gap diminishes when the number of users becomes large. We close the gap between theory and practice by building a prototype system at Berkeley, which we illustrate in Chapter 6. We describe in detail the system architecture, the selection of network codes, the system implementation and its experiments design.

While our theoretical analysis is based on a "static" setting where the demand and supply resources are fixed, we also show in our large-scale simulations that our algorithm works well even for the more interesting "dynamic" cases. We use realworld traces and show that the the system is scalable in the number of caches, users and the video catalog size, and that it is robust to different dynamic situations including user fluctuation and demand distribution changes. We observe via simulations the the complex interplay between disk space, network bandwidth and node degree bound, whose further theoretical study is of profound interest. We also give experimental results given by our prototype system to prove the feasibility of our architecture and algorithms, and present guidelines to practical design choices including caching update intervals, demand prediction and provisioning. The detailed experiment results are presented in Chapter 7.

We conclude in Chapter 8 and give concrete examples to discuss applications, extensions and future research directions to the work presented in this dissertation.

Chapter 2

Background

This chapter presents the problem formulation and discusses its challenges through a simple example. It then explores related work of the optimization and design of VoD systems.

2.1 Problem Formulation

The design of our VoD system begins with formulating it as a server load minimization problem by jointly optimizing over content placement, network link rate allocation and topology graph selection, which we present here through a simple example illustrated in Figure 2.1(a).

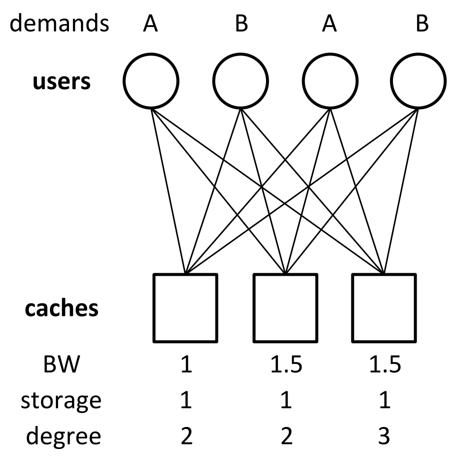
2.1.1 A Simple Toy Example

In this example, the system has two videos, A and B, each of size 1 GB, and a streaming rate of 1 Mbps. There are 4 users in the system, two requesting video A and the other two requesting video B. The cache cloud consists of 3 nodes. The network between the cache nodes and the user nodes form a fully connected graph. However, the node degree bound on each cache node allows them to connect to only a limited number of users simultaneously. For simplicity, we assume each cache has limited upload bandwidth resources[1] and limited storage capacities. The resource constraints for each cache node are listed below them. For example, cache node 1 has an upload bandwidth of 1Mbps, a storage capacity of 1GB and a degree bound of 2.

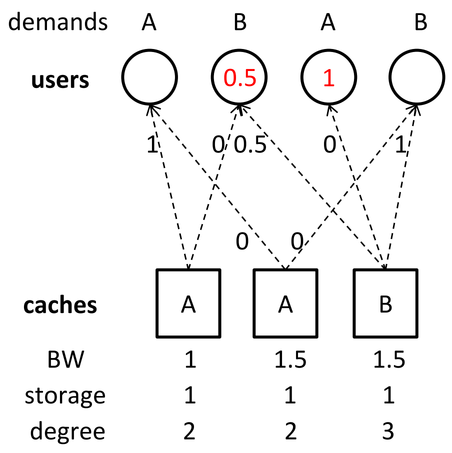
For each cache node, we have to decide which video it should store, which users it should connect to, and how it should allocate its upload bandwidth among these users. Figure 2.1 shows that these problems are interdependent. If we choose a "bad" topology and a "bad" content placement strategy such as the one in Figure 2.1(b), one user is missing an entire video and another use is missing half of a video (which needs to be filled in by the oracle server, as is indicated in the user circle). In Figure 2.1(c), a "good" content placement strategy is chosen, and only one video needs to be filled in by the server tier. In Figure 2.1(d), a "good" content placement strategy and a "good" topology is chosen, and only half of a video is missing from the cache network.

In summary, we have the following problems in the design of VoD systems:

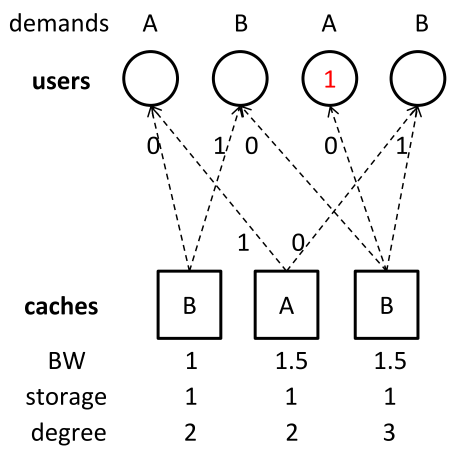
[1]In general, the network constraints can exist arbitrarily in any links in the network. We discuss this in the subsection 2.1.2



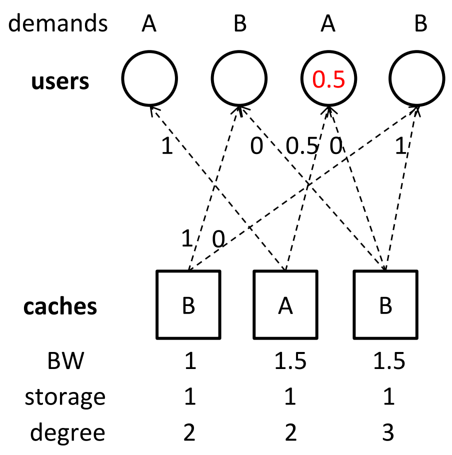
(a)



(b)



(c)



(d)

Figure 2.1: A simple example of VoD caching problem. The system has two videos of size 1 GB and rate 1 Mbps and there are 2 users requesting each video. The system employs 3 cache nodes with constraints on storage, bandwidth and out-degree as shown in (a). The problem is to decide, for each cache, which videos to store, which users to connect to, and how much bandwidth to allocate for each user. These questions are coupled. The connections between the cache nodes and users in (b) form a "bad" topology, and the content placement is non-optimal. In (b), one user is in deficit of an entire video from the cache network and another user is in deficit of half of video, both of which need to be filled in by the oracle server. The content placement in (c) is a "good" strategy, where one user is in deficit of one video from the cache cloud. In (d), the topology is a "good" one, and with the same content placement strategy in (c), one user is in deficit of only half of a video. In general, finding the "best" storage, bandwidth and topology combination is a combinatorially-hard problem.

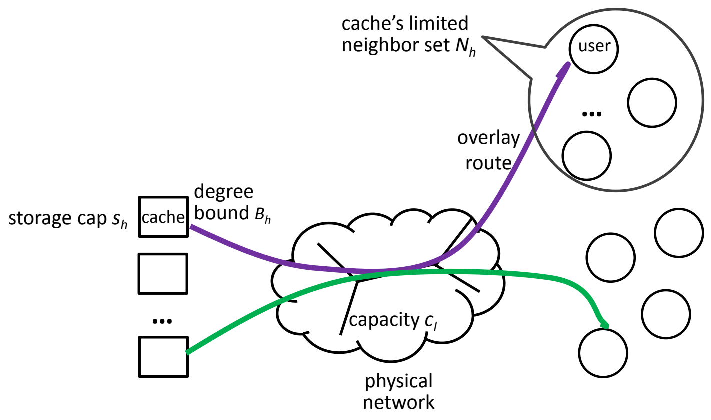


Figure 2.2: Caches and users connected by a physical network. The link capacity constraints can exist arbitrarily anywhere in the network.

(Q1) Content placement and link rate allocation: Which videos should each cache store, while respecting the storage constraint? Which videos and to which users should each cache node serve its content, while respecting the link rate constraints?

(Q2) Topology: Which users among its neighbors should each cache connect to, while respecting the node-degree constraint?

One can imagine that as the number of videos, number of caches and number of users increase, Q1 and Q2 will each involve an exponential number of choices. Moreover, these questions cannot be solved independently, as hinted by the example in Figure 2.1. The result is a hard combinatorial problem that is in general intractable.

In this dissertation, we tackle the first challenge by designing a class of network codes, called DRESS codes, to convert the combinatorial problem of content placement into a tractable one, and we tackle the second challenge by using a Markov approximation technique.

2.1.2 A General Formulation

We mathematically formulate the problems of Q1 and Q2 in a general setting in the following. Let us begin by introducing some notation listed in Table 2.1.

Denote by H the set of all cache nodes and by U the set of all users. There are M videos, each video m having a constant streaming rate γ_m and size (MB) of β_m, and being watched by a subset of users $U_m \subseteq U$.

To model arbitrary network link capacity constraints, suppose we have a set of overlay links R that connect the users and caches together, and a set of underlay links L that form the physical network. For each overlay link $r \in R$, it involves a set of underlay links, which is a subset of L, denoted by L_r. Each underlay link l may be used by several overlay links, and vice versa. Accordingly, we write $l \in r$ if $l \in L_r$. An overlay link $r = (h, u)$ enables node h to send data to

Table 2.1: Key Notation

Parameters	**Definition**
H	set of caches in tier II
U	set of users in tier III
M	number of videos
U_m	users watching video m
γ_m, β_m	video m's streaming rate and size
R	set of overlay routes
L	set of underlay links. $l \in r$ if link l belongs to route r
c_l	link capacity of l
A_{lr}	routing matrix. $A_{lr} = 1$ if $l \in r$
s_h	storage capacity of cache h
Q_v	connectable neighborhood of v
N_v^g	connected neighborhood of v under topology g
B_v	maximum size of connected neighborhood of v
G	set of all possible topology graphs
Auxiliary Variables	**Definition**
θ_l	shadow price of link l
q_r	$q_r = \sum_{l \in r} \theta_l$ is the shadow price of route r
λ_r	popularity index of route r
ω_h	storage price of cache h
Decision Variables	**Definition**
x_r	route rate of r
W_{hm}	storage of video m on cache h
p_g	probability of each topology graph g

node u in the overlay graph by setting up TCP/UDP connections. Figure 2.2 shows an overlay route can pass through multiple underlay links, and vice versa.

Let link l in the physical network L have a capacity c_l, and let x_r be the rate of overlay link r. Introduce the overlay link rates column vector $x := (x_r, r \in R)$, the link capacity column vector $c := (c_l, l \in L)$, and the routing matrix $A := (A_{lr}, l \in L)$, where $A_{lr} = 1$ if $l \in r$ and 0 otherwise. The physical link capacity constraint can then be formulated as $Ax \leq c$.

To model the storage constraints, let s_h be the storage capacity of cache node h and denote by $s := (s_h, h \in H)$ the column vector of the storage capacities. Let $W := (W_{hm}, h \in H, m \in M)$ be the storage matrix where $W_{hm} \in \{0, 1\}$ represents if video m is stored on cache node h ($W_{hm} = 1$) or not ($W_{hm} = 0$). Denote by $\beta := (\beta_m, m \in M)$ the vector of the sizes (in MB) of all videos. The storage constraints can then be expressed by $W\beta \leq s$.

To model the node degree bound constraints, assume each cache node h (user node u) can only see a subset $Q_h \subseteq U$ ($Q_u \subseteq H$) of users nodes (cache nodes), and can only simultaneously connect to a bounded number B_h (B_u) of users (caches). Denote by g a certain topology graph, by $N_h^g \subseteq Q_h$ ($N_u^g \subseteq Q_u$) the set of connected users of cache node h (user node u) under topology graph g.

Our interest is to minimize the load on the oracle server by jointly optimizing over the cache storage allocated to each video, the overlay link rates and the topology graph selection. Let

$z_u = \sum_{r=(h,u):h \in N_u^g} x_r$ be the total received rate by user u. We formulate this as the following optimization problem:

$$\begin{aligned}
\max_{x \geq 0, W, g} \quad & \sum_{u \in U} V(z_u) && (2.1)\\
\text{s.t.} \quad & x_{r:=(h,u)} \leq W_{hm}\gamma_m \;\; \forall h \in H, u \in U_m \cap N_h^g, && (2.2)\\
& Ax \leq c, && (2.3)\\
& W\beta \leq s, && (2.4)\\
& W \in \{0,1\}^{|H| \times M}, && (2.5)\\
& N_v^g \subseteq Q_v, |N_v^g| \leq B_v \;\; \forall v \in H \cup U, && (2.6)
\end{aligned}$$

where $V(z) = \max \gamma_m, z_u$ is user u's utility function, which says any aggregate received rate that exceeds the required streaming rate yields little usefulness. Maximizing such utility is equal to minimizing the sever load. In general, the $V(z)$ can take any non-decreasing concave functions. We choose this particular function in this dissertation only for easiness of discussions.

2.2 Related Work

The optimization of VoD systems has received wide attention. However, to the best of our knowledge, none of the prior works consider the joint optimization of topology graph selection, content placement and link rate allocation and give provably optimal results. We list the respectively the related works that address each subfield.

2.2.1 Video Content Placement

There are a number of works on content placement [6, 12, 62, 34, 59, 56, 9]. Almeida et al. [6] studied the delivery cost minimization problem under a fixed topology by optimizing over content replication and routing. Boufkhad et al. [12] investigated problem of maximizing the number of videos that can be simultaneously served by a collection of peers. Zhou et al. [62] focused on minimizing the load imbalance of video servers while maximizing the system throughput.

Almeida et al. [6] addressed the problem of replica placement, client request routing, and multicast stream routing in media content distribution systems employing scalable streaming protocols. The authors formulated a simple optimization models for a variety of scalable protocols including hierarchical merging, patching, periodic broadcasts, and scheduled broadcasts. With the aid of additional constraints that must hold in the scalable delivery system solution, they showed that a variety of realistic scenarios can be solved exactly using available optimization software. They also showed that using the optimal conventional unicast content distribution system for scalable delivery results in network costs. However, their setup is different because they consider server placement rather than distributed caching. They also use a replication based storage scheme. In our scheme, we use coding and show that the performance is much better. We also consider topology selection in our system which they assume is fixed.

Recently, Tan and Massoulie [56] studied the problem of optimal content placement in P2P networks. Their goal is to maximize the utilization of peers' uplink bandwidth resources. Optimal content placement strategies are identified in a particular scenario of limited content catalog under the framework of loss networks. Their work assumes that the peers' storage capacity grows unboundedly with system size. In contrast, our work does not make any assumption on the storage capacities. We also take into account the combinatorial aspect of the node degree constraints that was not studied in [56].

Applegate and et al. [9] formulated the problem of content placement into a mixed integer program that takes into account constraints such as disk space and link bandwidth. However, they assume the knowledge of the content popularity and a fixed topology is given. The content placement problem is solved approximately given the constraint that a video is either stored in its entity or not stored at all. In our work, we use a class of network codes that enables fractional storage, which helps convert an NP-hard problem to a convex problem that can be solved exactly in a distributed manner. We also do not assume the topology is fixed, and instead also optimize over the topology selections. Our scheme does not require any prior knowledge of the video demand distribution.

2.2.2 Optimal Bandwidth Utilization

With regard to network resource utilization, Borst et al. [11] solved a link bandwidth utilization problem assuming a tree structure with limited depth. An LP is formulated and under the assumption of symmetric link bandwidth, demand, and cache size, a simple local greedy algorithm is designed to find a close-to-optimal solution. Valancius et al. [57] propose an LP-based heuristic to calculate the number of video copies placed at customer home gateways. Both works assume a tree network structure. In contrast, the network topology in our work is not constrained to be a tree, and the video request patterns can be arbitrary in different network areas. Zhou and Xu [61] aimed to minimize the load imbalance among servers subject to disk space and network bandwidth constraints. However, they only consider egress link capacity from servers. In contrast, our formulation allows the link capacity constraints that may exist anywhere in the network.

Kulkarni et al. [32] proposes a heuristic algorithm for VoD systems that reduces the bandwidth requirement. Ho et al. [23] proposed a transmission policy for peer-to-peer systems to efficiently deliver video data by exploiting the multicast capability of the network. To avoid the disruption of services, the fault tolerance and recovery mechanism is also developed. They proposed a mathematical model to evaluate the performance of their policies. However, the only consider upload bandwidth bottleneck. Our algorithm assumes arbitrary link capacity constraints in the network, and provide theoretical guarantee of optimality.

2.2.3 Topology Graph Selection

Topology building is also an important design dimension and has been studied in various works [42, 33, 4]. While most works focus on enforcing locality-awareness and/or improving ISP-friendliness, they make the simple assumption that the graph is fully connected, i.e., no node-degree-bound is taken into consideration. Zhang et al. solve the problem of optimal P2P streaming under node degree constraints [60]. However their topology selection algorithm depends on a global statistics which is only easily accessible under a live-streaming scenario. In VoD, different users have individual demand on different videos. Directly applying their technique requires global statistics of all users' utility functions, which can create enormous overhead. Our work proposes a simple distributed algorithm that requires knowledge of only local information. We also give bounds on the algorithm performance and provide mixing time results.

There also exist a number works of topology graph selection using differential-equation based macroscopic analysis [50, 44, 40, 20, 45]. In [50], a refined fluid model of BitTorrent is proposed and the high efficiency of BitTorrent is shown. However, this model assumes node selection based on global knowledge of all nodes in the session, as well as uniform distribution of pieces. In contrast, our work does not assume global knowledge and each node has only a limited local view of networks. In [44], a coupon replication model is proposed by considering nodes with only limited upload. It is argued that overall system performance does not depend critically on either altruistic node behavior or the rarest-first piece selection strategy. In [40], an extend coupon replication model is proposed by considering nodes with limited upload and download capacity. With the same access link bottleneck assumption, in [20], a model is proposed to capture the trade-off between performance and fairness. In contrast, our model assume that the bottleneck link can be anywhere, which is more realistic. In [45], an improved piece selection strategy is proposed and analyzed. In contrast, in our work, we can characterize system trade-offs by both analytical modeling and simulation. Further, our results access properties which are hard to analyze before.

Another line of work is based on game-theoretic analysis [50, 21, 30, 48, 46, 38]. These works follow an economic flavor, including Tit-for-Tat (TFT) strategy analysis, feasibility of selfish behavior (free-riding), incentive compatibility, and auction analysis. Major parts of these studies are characterizing the existence, uniqueness, stability and other key properties of Nash equilibria. Our work is orthogonal to these studies.

Other approaches focus on real measurements [28, 10, 7, 49, 37, 36]. These measurements usually lasted for several months, either collecting tracker logs obtained from the trackers or collecting event logs by joining an ongoing torrent with a modified client. The track logs enable us to have the global view of BitTorrent performance whereas event logs enable us to observe the individual behavior of peers. Observations based on real measurements indeed give us some ad-hoc design heuristics. In contrast, our work enables us to have a systematic design, whose feasibility we prove by building a realworld prototype.

2.2.4 System Design and Performance Measurements

There are also a number of works on practical VoD system design. Huang et al. studied [27] the challenges and the architectural design issues of a large-scale VoD system based on the experiences of a real system deployed by PPLive [1]. Such challenges include coordinating content storage distribution, content discovery, and peer scheduling. Wang et al. studied the ISP-friendliness aspect of VoD systems, and proposed a solution to minimize the weighted sum of server load and non-ISP-friendly traffic [58] under the single video scenario. Liu et al. presented the design of the first production deployment of random network coding in VoD systems, and showed in-depth trace-driven analysis based on the collected 200 Gigabytes worth of real-world traces throughout the 17-day 2008 Olympic Games [43]. There are also a number measurement studies of practical VoD systems [22, 26].

Annapureddy et al. proposed a VoD system called Redcarpet [8]. The authors proposed an efficient video block dissemination algorithm in a mesh-based VoD system, and showed that prefetching and network coding techniques can greatly improve system performance. While we use a coding based framework to tackle the combinatorics of content placement, our problem formulation and scope is very different, and we are further able to provide theoretically provable performance guarantees.

Chapter 3

Content Placement

Solving the two combinatorial problems in (2.1) in bundle is a challenging task. In this Chapter, we first fix the topology and study how to achieve optimal content placement under arbitrary network link capacity constraints. The content placement problem is in general NP-hard. We tackle this difficulty by introducing a fractional storage idea. We show that with appropriate choice of network storage codes, fractional coding not only reduces the problem into a convex problem, but also has the potential of reducing the server load even further by allowing more flexibility in the system. We then show that the converted convex problem has an easy-to-implement distributed solution, whose convergence is verified using simulation results.

3.1 Convex Relaxation

By removing constraints (2.6), we have the following optimization problem:

$$
\begin{aligned}
\max_{x\geq 0, W} \quad & \sum_{u\in U} V(z_u) && (3.1)\\
\text{s.t.} \quad & x_{r:=(h,u)} \leq W_{hm}\gamma_m \;\; \forall h \in H, u \in U_m \cap N_h^g, && (3.2)\\
& Ax \leq c, && (3.3)\\
& W\beta \leq s, && (3.4)\\
& W \in \{0,1\}^{|H|\times M}, && (3.5)
\end{aligned}
$$

The above optimization problem is still NP-hard in general, as the storage matrix W can only take binary values, which makes the problem difficult to solve. To mitigate this, consider relaxing constraint (3.5) $W \in \{0,1\}^{|H|\times M}$ to $0 \leq W \leq 1$ with piecewise inequality as follows.

$$
\begin{aligned}
\max_{x\geq 0, W} \quad & \sum_{u\in U} V(z_u) && (3.6)\\
\text{s.t.} \quad & x_{r:=(h,u)} \leq W_{hm}\gamma_m \;\; \forall h \in H, u \in U_m \cap N_h^g, && (3.7)\\
& Ax \leq c, && (3.8)\\
& W\beta \leq s, && (3.9)\\
& 0 \leq W \leq 1, && (3.10)
\end{aligned}
$$

which becomes a convex problem. However, the following questions arise.

- How to achieve fractional $0 \leq W \leq 1$ storage of videos? What does it mean in practice?
- Constraint (3.8) with fractional storage $0 \leq W \leq 1$ adds an additional requirement that the fractional storage of the same video by different caches be "orthogonal" to the users. In other words, users requesting video packets from different caches should be able to decode the video as long as they enough number of the full set of video packets regardless of the actual instances of the packets. This is a difficult problem because we need to carefully allocate packets to guarantee that none of the packets received by the users have duplicates. This problem does not exist if W contains only binary values of 0 and 1, because each cache has either does not have the video or has the full copy of the video, in which case the problem of duplicates can always be avoided with sufficient communication between the nodes.

3.2 Fractional Storage

To resolve the above problems, let us revisit our simple example in Figure 2.1(c) shown again in Figure 3.1(a). In Figure 3.1(b), we fix the same topology but adopt a fractional storage mechanism. In particular, videos A and B are broken into halves (A_1, A_2) and (B_1, B_2). Interestingly in this case, all users are fully satisfied, and cache 1 has to use only half of his storage capacity.

This example illustrates the interesting fact that relaxing the constraint that a cache node stores only full (or none) copies of the videos improves rather than hurts the system performance, provided that the allocation of the packets are carefully chosen.

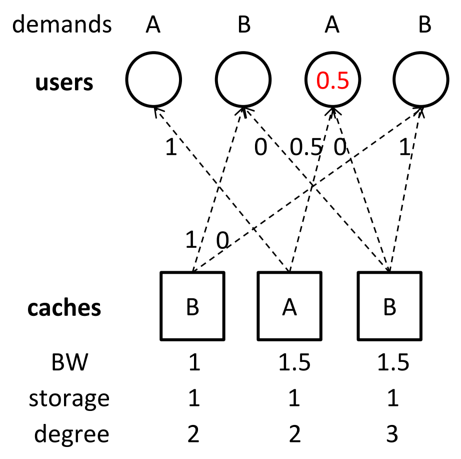
3.3 Role of Codes

However, although fractional storage may yield superior solutions, it does not make the problem convex yet. The problem is still combinatorial because one needs now to answer the even more difficult question of "which parts of which video" to store on each cache instead of simply "which video" to store on each cache.

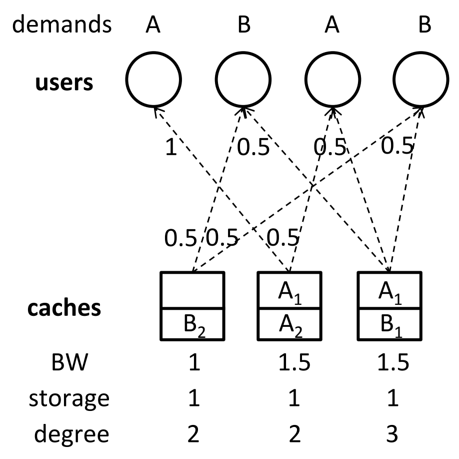
In fact, the problem as formulated above remains hard even if we restrict it to the special case of multicast demands, i.e., when all the users want to watch the same movie. However, it is known that the multicast problem is tractable and the key idea for solving it is to use codes. In this case, each cache stores coded packets of videos. The intuition for the multicast case, as explained in the Fountain coding [14] and network coding literature [5, 25, 29], is that codes transform the content into a fluid where coded packets can be thought of as "drops". A user has to collect enough drops (per time unit), and does not need to keep track of the identity of the packets, to be able to decode and play the content. Therefore, solving the problem reduces to ensuring that the network can sustain a flow equal to the movie rate to each user[1].

The details of the code design to approximate the fluid-limit constraint in (3.10) are given in Chapter 6. Here, we present a high-level illustration of what happens when we design a perfect code. In Figure 3.1(c), we replace the packets, i.e., (A_1, A_2) and (B_1, B_2), of videos A and B by

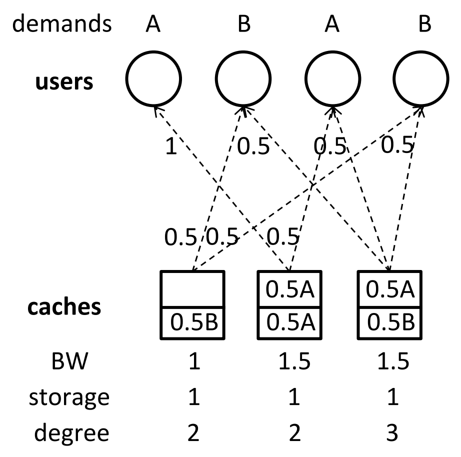
[1]Practical codes can come close to this fluid abstraction.



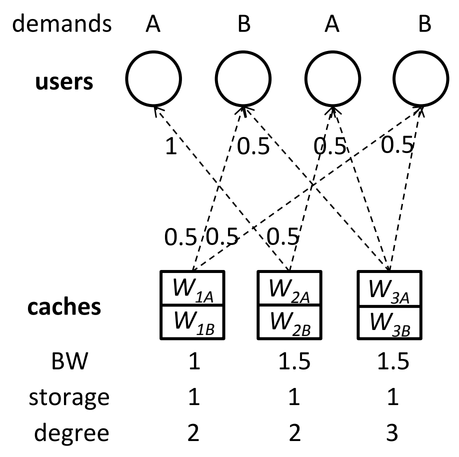
(a)



(b)



(c)



(d)

Figure 3.1: A simple example demonstrating the usefulness of fractional storage. The system has the same setup as that in Figure 2.1. In (a), the videos are stored in their entirety. Even with the optimal topology, the oracle server still needs to fill in the gap of half of a video. In (b), videos A and B are broken into halves (A_1, A_2) and (B_1, B_2) and are stored on the caches. In this case, the users are fully satisfied and cache 1 has filled only half of its storage. In (c), the fluid limit version of (b) is given, where the video packets are replaced by video "drops", and it is enough that users obtain enough number of "drops" to meet their streaming requirements. In general, the caches can store a fraction (between $[0, 1]$) of a video, as is indicated in (d).

video "drops", represented by $0.5A$ and $0.5B$. The users only need to receive enough amount of "drops" to meet their streaming requirements instead of worrying about the identity of the packets. While we will discuss how to design a code to satisfy such requirements in Chapter 6, for the time being to easy discussions, let us assume that we have designed a perfect code such that this is true unless stated otherwise. Therefore, by using the fractional storage concept, and by the appropriate design of codes, we can make valid of the fluid-limit constraint in (3.10), i.e., we allow caches to store a fraction between $[0, 1]$ of a video, as is indicated in Figure 3.1(d).

As a final note, it is worth noting that although we also use codes to tackle the combinatorial difficulty of the content placement problem, the problem we address is more general than the problem of classic multicast over a known network. The differences are three-fold. First, we do not have a fixed network, rather we have to find the optimal topology within the design constraints of maximum connectivity degree. Secondly, we have a content-placement problem of deciding what content to place where in the cache cloud. Thirdly, we have a non-multicast setup where different users demand different content.

3.4 Distributed Solution

3.4.1 Algorithm Optimality

Now that the problem becomes convex, we can solve it using a primal-dual algorithm, which we state in the following theorem.

Theorem 1. *The problem* (3.6) *can be solved by the following three-stage primal-dual algorithm that converges to the optimal solution.*

1) Step 1: update the upload rate.

$$\dot{x}_r = [\delta_r(V_{x_r}(z_u) - \lambda_r - q_r)]^+_{x_r} \tag{3.11}$$

where $V_{x_r}(z_u)$ is the derivative of function $V(z_u)$ with respect to x_r. $q_r = \sum_{l:l\in r} \theta_l$ is the aggregate route price, where θ_l is the single link price on link l and is updated by:

$$\dot{\theta}_l = [\eta_l(\sum_{r:l\in r} x_r - c_l)]^+_{\theta_l}. \tag{3.12}$$

where $\delta_r, \eta_l > 0$ are adaptation parameters. λ_r is explained in the following.

2) Step 2: Update the demand index. The parameter λ_r is updated via:

$$\dot{\lambda}_r = [\kappa_r(x_r - W_{hm}\gamma_m)]^+_{\lambda_r} \tag{3.13}$$

where step size κ_r is a positive constant, and m is the index of the video watched by user u. This parameter captures the relative demand, i.e., absolute demand minus supply, of the video delivered on route r, hence its name demand index.

3) Step 3: Update cache storage

$$\begin{cases} \dot{W}_{hm} = [\iota_{hm}(\Lambda_{hm} - \beta_m\omega_h)]^{[0,1]}_{W_{hm}} \\ \dot{\omega}_h = [\nu_h(\sum_{m\in M} W_{hm}\beta_m - s_h)]^+_{\omega_h} \end{cases} \tag{3.14}$$

where the Lagrangian variable ω_h is interpreted as the storage price for the storage constraint, and $\Lambda_{hm} = \gamma_m \cdot (\sum_{r=(h,u):u\in U_m, h\in N^g_u} \lambda_r)$ is the aggregate demand index and β_m the size of video m.

Proof. Given a topology graph g, the problem in (3.6) has concave objective functions and linear constraints, therefore the Slater constraint qualification conditions are satisfied [13] and strong duality holds. Relaxing the first set of constraints in (3.7), we obtain its partial Lagrangian:

$$L(x, W, \lambda) = \sum_{u \in U} V(z_u) + \sum_{m \in M, u \in U_m} \sum_{r=(h,u): h \in N_u^g} \lambda_r (W_{hm}\gamma_m - x_r),$$

where $\lambda := (\lambda_r, r \in R)$ is the column vector of Lagrange multipliers of constraints (3.7). Then we can solve an equivalent dual problem:

$$\begin{aligned} &\min_{\lambda \geq 0} \max_{x \geq 0, 0 \leq W \leq 1} L(x, W, \lambda) \\ &s.t \qquad (3.8) - (3.10). \end{aligned}$$

Given λ, we have two separated subproblems. One subproblem is cache storage allocation for each cache node $h \in H$:

$$\begin{aligned} \max_{W_h} \quad & \sum_{m \in M} \Lambda_{hm} W_{hm} \\ \text{s.t.} \quad & \sum_{m \in M} W_{hm}\beta_m \leq s_h, \\ & 0 \leq W_{hm} \leq 1 \ \ \forall m \in M, \end{aligned}$$

where $\Lambda_{hm} = \gamma_m \cdot (\sum_{r=(h,u): u \in U_m, h \in N_u^g} \lambda_r)$. This is a linear programming problem that maximizes the weighted summation of W_{hm} subject to box constraints, which can be solved by a primal dual algorithm:

$$\begin{cases} \dot{W}_{hm} = [\iota_{hm}(\Lambda_{hm} - \beta_m \omega_h)]_{W_{hm}}^{[0,1]} \\ \dot{\omega}_h = [\nu_h(\sum_{m \in M} W_{hm}\beta_m - s_h)]_{\omega_h}^{+} \end{cases}$$

The remaining subproblem is link rate allocation:

$$\begin{aligned} \max_{x \geq 0} \quad & \sum_{m \in M, u \in U_m} \left[V(z_u) - \sum_{r=(h,u): h \in N_u^g} \lambda_r \cdot x_r \right] \\ \text{s.t.} \quad & Ax \leq c. \end{aligned}$$

Assign Lagrangian multipliers θ_l to the capacity constraint for link l. This multiplier can be thought of as the shadow price of the underlay link [55, 41], which summarizes the link congestion information. Let $q_r := \sum_{l \in r} \theta_l$ be the overlay link r price aggregate over all the underlay links that connect the overlay link.

Introduce the underlay link price column vector $\theta := (\theta_l, l \in L)$ and the overlay price column vector $q := (q_r, r \in R)$. Then we have $q = A^T\theta$. A primal dual algorithm can then be used to solve the link rate allocation as follows. The route rate can be adjusted by:

$$\dot{x}_r = [\delta_r(V_{x_r}(z_u) - \lambda_r - q_r)]_{x_r}^{+},$$

where $V_{x_r}(z_u)$ is the derivative of function $V(z_u)$ with regard to x_r. The link shadow prices can be updated via:

$$\dot{\theta}_l = [\eta_l(\sum_{r:l\in r} x_r - c_l)]^+_{\theta_l},$$

where $\delta_r, \eta_l > 0$ are adaptation parameters.

After solving the above two subproblems, we obtain x and W. Then we update Lagrange multipliers λ by the following sub-gradient method:

$$\dot{\lambda}_r = [\kappa_r(x_r - W_{hm}\gamma_m)]^+_{\lambda_r}$$

where step size κ_r are positive constants.

Overall, the solution is given by the following set of update equations.

$$\begin{cases} \dot{x}_r = [\delta_r(V_{x_r}(z_u) - \lambda_r - q_r)]^+_{x_r} \\ \dot{\theta}_l = [\eta_l(\sum_{r:l\in r} x_r - c_l)]^+_{\theta_l} \\ \dot{\lambda}_r = [\kappa_r(x_r - W_{hm}\gamma_m)]^+_{\lambda_r} \\ \dot{W}_{hm} = [\iota_{hm}(\Lambda_{hm} - \beta_m\omega_h)]^{[0,1]}_{W_{hm}} \\ \dot{\omega}_h = [\nu_h(\sum_{m\in M} W_{hm}\beta_m - s_h)]^+_{\omega_h} \end{cases}$$

To prove its optimality, we observe that at optimal, the following KKT conditions [13] of the overall Lagrangian should hold:

$$\begin{cases} [V_{x_r}(z_u^*) - \lambda_r^* - q_r^*]^+_{x_r^*} = 0 \\ \theta_l^*(\sum_{r:l\in r} x_r^* - c_l) = 0 \\ \lambda_r^*(x_r^* - W_{hm}^*\gamma_m) = 0 \\ (\Lambda_{hm}^* - \beta_m\omega_h^*)^{[0,1]}_{W_{hm}^*} = 0 \\ \omega_h^*(\sum_{m\in M} W_{hm}^*\beta_m - s_h) = 0 \end{cases}$$

where $\Lambda_{hm}^* = \gamma_m \cdot (\sum_{r=(h,u):u\in U_m, h\in N_u^g} \lambda_r^*)$, and $*$ denotes the optimal value of the corresponding variables. Let x^* and W^* be the primal optimal, and θ^*, ω^* and λ^* are the dual optimal. Denote by $y = (x, W, \theta, \omega, \lambda)$ and by $y^* = (x^*, W^*, \theta^*, \omega^*, \lambda^*)$. To prove that $y \to y^*$, we propose the following generalized energy function:

$$\begin{aligned} E(y) &= \frac{1}{2\delta}\|x - x^*\|^2 + \frac{1}{2\iota}\|W - W^*\|^2 + \frac{1}{2\eta}\|\theta - \theta^*\|^2 \\ &\quad + \frac{1}{2\kappa}\|\lambda - \lambda^*\|^2 + \frac{1}{2\nu}\|\omega - \omega^*\|^2, \end{aligned}$$

and show that (a) $E(y) > 0\ \forall y \neq y^*$ and $V(y^*) = 0$; (b) $\dot{E}(y) \leq 0\ \forall y$ and $\dot{E}(y^*) = 0$.

(a) is obvious since $E(y)$ is summation of quadratic terms centered at y^*. To show (b), we derive $\dot{E}(y)$:

$$\begin{aligned}\dot{E}(y) \;=\; & \sum(x_r - x_r^*)(V_{x_r}(z_u) - (q_r + \lambda_r))^+_{x_r} \\ & + \sum(W_{hm} - W^*_{hm})(\Lambda_{hm} - \beta_m\omega_h)^{[0,1]}_{W_{hm}} \\ & + \sum(\theta_l - \theta_l^*)(\sum_{r:l\in r} x_r - c_l)^+_{\theta_l} \\ & + \sum(\lambda_r - \lambda_r^*)(x_r - W_{hm}\gamma_m)^+_{\lambda_r} \\ & + \sum(\omega_h - \omega_h^*)(\sum_{m\in M} W_{hm}\beta_m - s_h)^+_{\omega_h}\end{aligned}$$

by applying partial derivatives and plugging in the dynamic system equations. For simplicity, we have omitted the sets over which the terms are summed up. It is easy to see that $\dot{E}(y^*) = 0$. Now we can upper-bound $\dot{E}(y)$ as follows:

$$\begin{aligned}\dot{E}(y) \;\leq\; & \sum(x_r - x_r^*)(V_{x_r}(z_u) - (q_r + \lambda_r)) \\ & + \sum(W_{hm} - W^*_{hm})(\Lambda_{hm} - \beta_m\omega_h) \\ & + \sum(\theta_l - \theta_l^*)(\sum_{r:l\in r} x_r - c_l) \\ & + \sum(\lambda_r - \lambda_r^*)(x_r - W_{hm}\gamma_m) \\ & + \sum(\omega_h - \omega_h^*)(\sum_{m\in M} W_{hm}\beta_m - s_h) \\ = \;& \sum(x_r - x_r^*)(V_{x_r}(z_u) - V_{x_r}(z_u^*)) \\ & + \sum(x_r - x_r^*)(V_{x_r}(z_u^*) - (q_r^* + \lambda_r^*)) \\ & + \sum(W_{hm} - W^*_{hm})(\Lambda^*_{hm} - \beta_m\omega_h^*) \\ & + \sum(\theta_l - \theta_l^*)(\sum x_r^* - c_l) \\ & + \sum(\lambda_r - \lambda_r^*)(x_r^* - W^*_{hm}\gamma_m) \\ & + \sum(\omega_h - \omega_h^*)(\sum W^*_{hm}\beta_m - s_h) \\ \leq \;& \sum(x_r - x_r^*)(V_{x_r}(z_u) - V_{x_r}(z_u^*)) \\ & +0 + 0 + 0 + 0 + 0 \leq 0\end{aligned}$$

where the first inequality is obtained by dropping the $^{[a,b]}_{x}$ terms, the second equality is obtained by canceling opposite terms, and the third inequality is obtained by applying the set of KKT conditions. The last set inequality holds due to the concavity [13] of the function $\min\left(\sum_{r:l\in r} x_r, \gamma_m\right)$ over x_r.

Using (a) and (b), it follows from Krasovskii-LaSalle principle [35] that y converges to the set $\{y|\dot{E}(y) = 0\}$, and that such a set contains no trajectories other than $\{y = y^*\}$. This concludes the proof. □

The update equations are intuitive, which we illustrate by an example in the next subsection. The upload rate increases linearly with the marginal utility function and decreases linearly

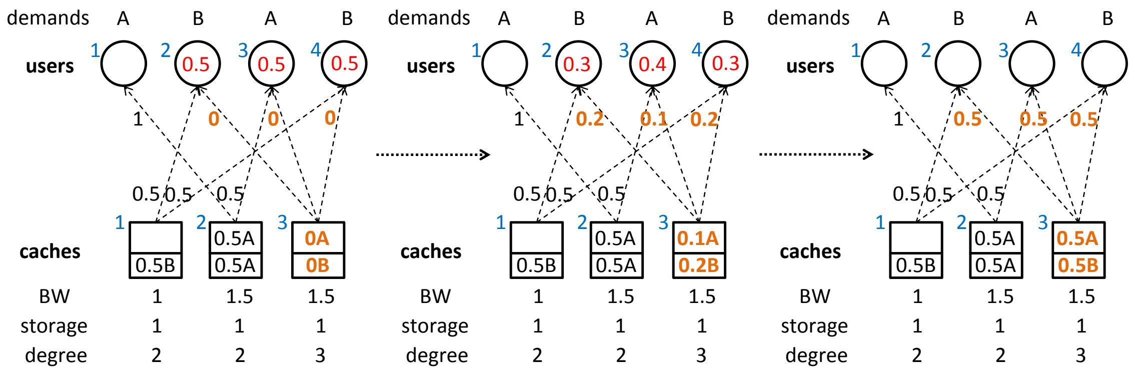


Figure 3.2: An illustrative example of the content placement and link rate allocation algorithm.

with the demand index and link shadow price. The demand index increases if the desired link rate exceeds what the stored video can offer, and vice versa. Cache node h increases storage of video m if there is more demand than supply, i.e., when the popularity index is larger than the storage price, and vice versa.

3.4.2 Algorithm Illustration

The above algorithms can be implemented in a fully distributed way. Figure 3.2 shows an example of the three update steps described above. Consider cache node 3 in this example. At the beginning, all the parameters $x_{3u}, u \in \{2, 3, 4\}$, $W_{3m}, m \in \{A, B\}$ and the Lagrangian multipliers are set to zero. The first step is to update the desired link rate. Take user 3 for example. Since user 1 only receives 50% of the required streaming rate, the derivative of its utility function is one. By step 1, $\Delta x_{33} = \delta_3[1 - 0 - 0]_0^+$. Take $\delta_3 = 0.1$, we have the overall update equation $x_{33} \leftarrow x_{33} + 0.1 = 0.1$. Similarly, $x_{32} \leftarrow 0.1$ and $x_{34} \leftarrow 0.1$. Since $\sum_u x_{3u} = 0.3 < 1.5$, the route price q_3 remains zero.

The second step is to update the demand index. Since $W_{3m} = 0, \forall m \in \{A, B\}$, by step 2 we know that $\Delta\lambda_{33} = [\kappa_3(0.1-0)]_0^+$. Set $\kappa_3 = 1$ for example, and we have $\lambda_{33} \leftarrow \lambda_{33}+0.1 = 0.1$. Similarly, $\lambda_{32} \leftarrow 0.1$ and $\lambda_{34} \leftarrow 0.1$.

The third step is to update the storage. Since $\Lambda_{33} = \lambda_{33} = 0.1$, by the equations in step 3 we have $W_{3A} \leftarrow W_{3A} + 0.1 = 0.1$ when we set $\iota_{33} = 1$. Similarly, $W_{3B} \leftarrow W_{3B} + 0.2 = 0.2$. Therefore, cache node 3 now stores 10% of video A and 20% of video B. The above result is illustrated in the second sub-figure of Figure 3.2. The iteration process then repeats, until either the storage or the bandwidth saturates, as is illustrated in the third sub-figure in Figure 3.2.

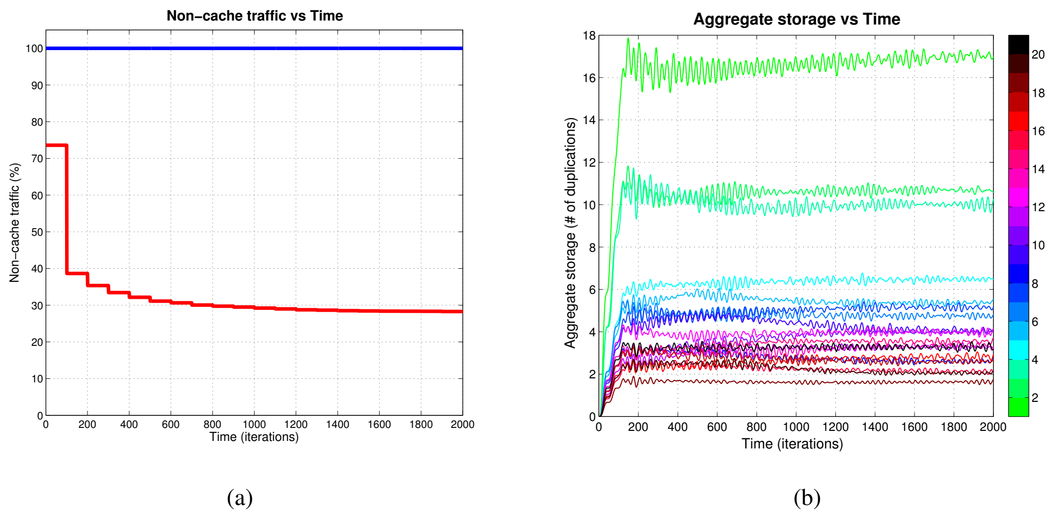


Figure 3.3: Demonstration of the convergence behavior of the content placement and link rate allocation problem. (a) non-cache traffic versus time. We will show in the next section that with topology graph selection the non-cache traffic can eventually reach zero. (b) Total storage amount of each video in the cache network. The algorithm stabilizes in approximately 200 steps.

3.5 Convergence Behavior

We verify the convergence behavior of the algorithms via a simple experiment. This example is a mini-P2P scenario with 20 videos, 200 users and 100 caches[2]. The videos have a flat streaming rate of 2Mbps. Users select videos according to Zipf's law, i.e, they choose video m with probability proportionally to $1/m^s$. We choose $s = 0.8$ in this case. Each cache can store only up to one video. Caches' bandwidth capacity is modeled by a mixtures of two Gaussian distributions $N(2, 0.3)$ and $N(6, 0.3)$ of equal weight. Note that the average bandwidth resources is just enough to satisfy all users demand. A fixed initial topology is given where each cache node randomly connects to 6 user nodes. Figure 3.3(a) shows the traffic (in percentage) from non-cache servers. Figure 3.3(b) shows the aggregate storage for each video among all caches, where each color represents a different video. We can see that the algorithm stabilizes in a few hundred steps. We show in the next section that a fixed random topology is not able to achieve zero non-cache traffic, which can be achieved using our topology graph selection algorithm.

[2]One can think of this as 50% of the peer users are willing to share their storage and bandwidth resources

Chapter 4

Topology Graph Selection

In this chapter, we move forward to solve for the problem of topology graph selection under node degree bound. In general, this problem itself is NP-hard. We apply a Markov approximation technique that solves the problem in a distributed manner and achieves close-to-optimal solution.

4.1 Heuristic Graph Selection Algorithm

Let us motivate the algorithm design by a simple toy example in Figure 4.1. In this example, we will use the following heuristic topology selection algorithm: each cache node waits for some time T, chokes his worst uplink neighbor[1], and randomly connects to another neighbor. For simplicity in the toy example, assume T is large enough such that the content placement and link rate allocation algorithm has fully converged before the next choking happens[2]. Since each node chokes his worst neighbor, we call this the worst-neighbor-choking algorithm.

Figure 4.1 shows two iteration steps of this algorithm. At the beginning, the cache nodes collectively still cannot support user 3's 50% of his streaming rate. After some time T, cache node $1, 2$ and 3 respectively chokes their worst neighbor user, and connects to a new neighbor user. For example, cache node 1 chokes user 1 and connects to user 3. The new topology graph is shown in the middle sub-figure. For clear illustration purpose, we use a different color to show the changes in the topology links and link rates for each cache node. In the second step, all cache nodes also choke their worst neighbor and every user is fully satisfied.

The worst-neighbor-choking algorithm makes intuitive sense: each node keeps getting rid of worst neighbors and exploiting new neighbors. The algorithm is also fully distributed and easy to implement. Fortunately as we will show in the following, by implementing a slight variation of the of it, which we call the soft-worst-neighbor-choking algorithm, one can show it achieves a provably close-to-optimal solution. By soft choking, it means that each node chokes their neighbors according to a certain probability distribution with the highest probability landing on the worst neighbor. We will start the algorithm design by formulating the topology selection algorithm first.

[1]If there are multiple worst uplinks, choke one randomly.

[2]We will discuss a more general case without such an assumption later in the paper.

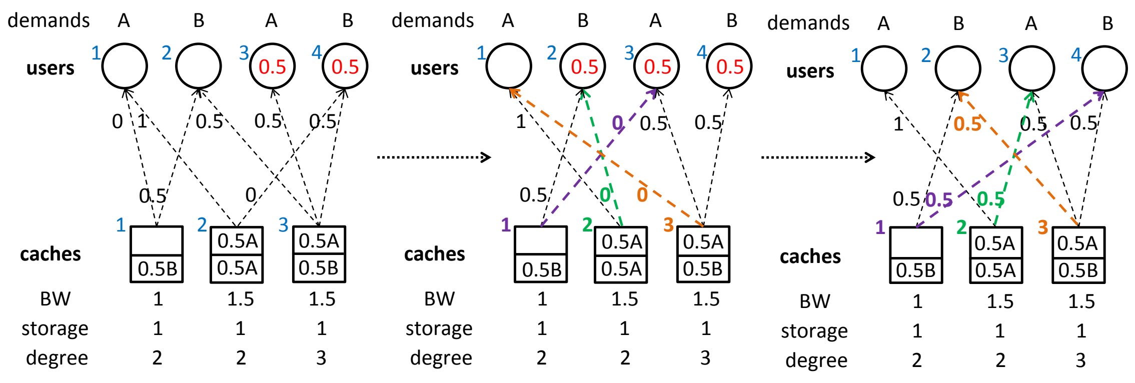


Figure 4.1: An illustrative example of the worst-neighbor-choking topology graph selection algorithm.

4.2 Problem Formulation

4.2.1 Utility Maximization

Let Φ_g be the optimal solution to (3.6) under a fixed graph g, which we can solve using the algorithms shown in Chapter 3. For simplicity of discussion, we assume here that Φ_g has converged to its optimal value under topology graph g. However, similar results will still hold even if it is not the case, as we will show in the performance analysis in Chapter 5.

To model the node degree bound constraints, assume each cache node h (user node u) can only see a subset $Q_h \subseteq U$ ($Q_u \subseteq H$) of users nodes (cache nodes), and can only simultaneously connect to a bounded number B_h (B_u) of users (caches). Denote by g a certain topology graph, by $N_h^g \subseteq Q_h$ ($N_u^g \subseteq Q_u$) the set of connected users of cache node h (user node u) under topology graph g. Our goal is to maximize the overall system utility while satisfying all the degree bound constraints on all the nodes:

$$\begin{aligned} \max_{g} \quad & \Phi_g \\ \text{s.t.} \quad & N_v^g \subseteq Q_v, |N_v^g| \leq B_v \quad \forall v \in H \cup U. \end{aligned} \tag{4.1}$$

Directly solving for (4.1) is a challenging task because the problem is non-convex and there exists combinatorial number of choices. Even for the small toy example in Figure 2.1(a), the total number of topology graphs is $6 \times 6 \times 4 = 144$. To convert the problem into a solvable (convex) one with distributed solutions, we design a Markov approximation technique introduced in [16]. Details follow.

4.2.2 Entropy Approximation

First, we rewrite the above formulation with some minor tweak. Denote by G the set of all possible topology graphs that satisfy all the node degree constraints. The topology building (**TB**) problem can then be re-written in the following way:

$$\begin{aligned} \mathbf{TB}: \quad & \max_{p \geq 0} \sum_{g \in G} p_g \Phi_g, \\ \text{s.t.} \quad & \sum_{g \in G} p_g = 1, \end{aligned} \tag{4.2}$$

where p_g is the probability associated with topology graph $g \in G$. Its optimal solution is $\max_{g \in G} \Phi_g$, and is obtained by setting the probability corresponding to one of the "best" topology graphs to be 1 and the rest probabilities to be 0.

To mitigate the problem of exponential number of graphs, consider using

$$\frac{1}{\mu} \log(\sum_{g \in G} \exp(\mu \Phi_g)) \tag{4.3}$$

to approximate $\max_{g \in G} \Phi_g$. The reason is that while solving for $\max_{g \in G}$ is difficult, solving for (4.3) can be made much easier as we will explain later on. We present the following theorem adapted from [16].

Theorem 2. (4.3) *is the optimal value to the following convex optimization problem:*

$$\max_{p \geq 0} \quad \sum_{g \in G} p_g \Phi_g - \frac{1}{\mu} \sum_{g \in G} p_g \log p_g, \tag{4.4}$$

$$\textit{s.t.} \quad \sum_{g \in G} p_g = 1, \tag{4.5}$$

where μ is a positive constant. The optimal solution is given by:

$$p_g^* = \frac{\exp(\mu \Phi_g)}{\sum_{g' \in G} \exp(\mu \Phi_{g'})}, \ \forall g \in G. \tag{4.6}$$

Moreover, the gap between its optimal value (4.3) *and the optimal value of the original problem in* (4.2) *is given by:*

$$0 \leq \frac{1}{\mu} \log(\sum_{g \in G} \exp(\mu \Phi_g)) - \max_{g \in G} \Phi_g \leq \frac{1}{\mu} \log |G|, \tag{4.7}$$

where $|G|$ is the size of G.

Note that $\sum_{g \in G} p_g \log p_g$ is also the entropy $H(p)$ of the distribution $p := (p_g, g \in G)$, and we call this is an "entropy-approximated" formulation. The only difference between the objective function in (4.4) and that in (4.2) is the addition of the weighted entropy term. As $\mu \to +\infty$, $p_{g^*}^* \to 1$ where $g^* = \arg\max_{g \in G} \Phi_g$ and $p_g^* \to 0$ otherwise. Therefore, the optimal value in the entropy-approximated formulation approaches to that of the original problem as μ becomes large. The proof is given as follows.

Proof. The Lagrangian of the problem (4.4) is given by:

$$\begin{aligned} L(p_g, \nu_g, \rho) = \quad & \textstyle\sum_{g\in G} p_g \Phi_g - \frac{1}{\mu}\sum_{g\in G} p_g \log p_g \\ & + \textstyle\sum_{g\in G} \nu_g p_g + \rho(1 - \sum_{g\in G} p_g), \end{aligned}$$

where ν_g and ρ are the Lagrangian variables. At optimal, the following KKT conditions [13] should hold:

$$\Phi_g - \frac{1}{\mu}(\log p_g^* + 1) + \nu_g^* - \rho^* = 0,\ \forall g \in G,$$

Writing p^* as a function of ν_g^* and ρ^* and applying the constraint $\sum_{g\in G} p_g = 1$, we obtain:

$$\rho^* = \frac{1}{\mu}\log\big(\sum_{g\in G}\exp\big(\mu(\Phi_g + \nu_g^*) - 1\big)\big).$$

Plugging ρ^* back into p_g^*, we get:

$$\begin{aligned} p_g^* &= \frac{\exp\big(\mu(\Phi_g + \nu_g^*) - 1\big)}{\exp(\mu\rho^*)} \\ &= \frac{\exp(\mu\Phi_g)}{\sum_{g\in G}\exp(\mu\Phi_g)},\ \forall g \in G. \end{aligned}$$

which is the optimal solution. Plugging it into the optimization problem, we have the optimal value equal to $\frac{1}{\mu}\log(\sum_{g\in G}\exp(\mu\Phi_g))$.

Since p_g^* is the optimal solution, the corresponding optimal value should be greater than or equal to that with any distribution p_g, e.g, $p_g = 1$ when $g = \arg\max_{g\in G}\Phi_g$ and $p_g = 0$ otherwise. The objective function with such a p_g is equal to $\max_{g\in G}\Phi_g$, and therefore:

$$\frac{1}{\mu}\log(\sum_{g\in G}\exp(\mu\Phi_g)) \geq \max_{g\in G}\Phi_g$$

On the other hand, we know that $\sum_{g\in G} p_g\Phi_g \leq \max_{g\in G}\Phi_g$ for any distribution p_g, and that $H(p_g) = -\frac{1}{\mu}\sum_{g\in G} p_g \log p_g \leq \frac{1}{|\mu|}\log G$ for any distribution p_g because of the properties of entropy. Therefore, we also have

$$\max_{p\geq 0}\sum_{g\in G} p_g\Phi_g - \frac{1}{\mu}\sum_{g\in G} p_g\log p_g \leq \max_{g\in G}\Phi_g + \frac{1}{|\mu|}\log G$$

by piecewise inequality, which gives:

$$\frac{1}{\mu}\log(\sum_{g\in G}\exp(\mu\Phi_g)) \leq \max_{g\in G}\Phi_g + \frac{1}{|\mu|}\log G$$

Combining the two results above, we have:

$$0 \leq \frac{1}{\mu}\log(\sum_{g\in G}\exp(\mu\Phi_g)) - \max_{g\in G}\Phi_g \leq \frac{1}{\mu}\log|G|.$$

□

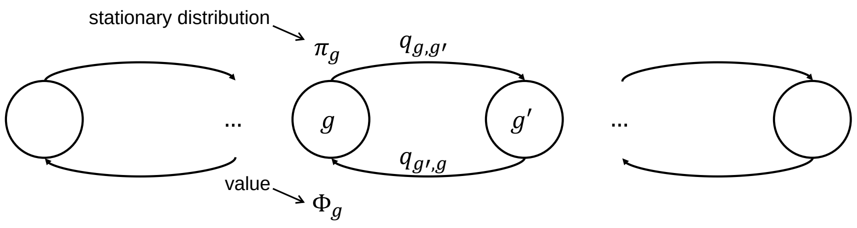


Figure 4.2: Illustration of an Markov chain with state space the topology graphs G. The value Φ_g of each state g is the optimal value of the content placement and link rate allocation scheme. As the Markov chain transits in between states according to designed transition rates $q_{g,g'}$, the Markov chain in its stationary status will achieve an average value of (4.3).

Intuitively, one can see that $\frac{1}{\mu}\log(\sum_{g\in G}\exp(\mu\Phi_g))$ is a good approximation of $\max_{g\in G}\Phi_g$ when μ is large. This is because the term $\exp(\mu\Phi_{g^*})$ will dominate the sum of exponentials $\sum_{g\in G}\exp(\mu\Phi_g)$. When this happens, we have:

$$\frac{1}{\mu}\log(\sum_{g\in G}\exp(\mu\Phi_g)) \to \frac{1}{\mu}\log(\exp(\mu\Phi_{g^*})) = \Phi_{g^*} \tag{4.8}$$

4.3 Algorithm Design

Recall that our goal is to solve for problem (4.2), and we know that one approximation is given by (4.4), but why is this approximation useful? Given the vast number of possibilities, how would we design a distributed algorithm to achieve the optimal topology graph?

The key idea is to construct a Markov chain (MC) on the topology graphs, and design its transition rates such that when the MC is in its stationary status, the topology graphs $g \in G$ are time-shared according to the distribution p_g^* in (4.5). As μ becomes large, the system spends most of the time in the optimal topology graph and the gap between the approximated solution and the optimal solution approaches to zero. If the transition rates can be implemented in a distributed way, we will have effectively solved the entropy-approximated version of the topology graph selection algorithm distributively. Figure 4.2 gives an illustration of this process.

It was shown in [16, 60] that it is possible to design such Markov chain to guide distributed algorithm designs in various domains, including wireless scheduling, channel assignment and etc. However, in our problem, directly applying these known design options in [16, 60] does not result in Markov chains that are distributively implementable. In this section, we aim to design algorithms that can be easily implemented in a distributed way.

Equation (4.5) is of a product form, and can be the stationary distribution of some time-reversible MC with state space the set of all the topology graphs G. To design the Markov chain for our topology selection, we focus on the design of transition rates $q_{g,g'}$ between states g and g'. With slight abuse of notation, let us think of each topology graph g as a set of directed overlay routes that connect the cache nodes and user nodes, i.e., $g := (r = (h, u), r \in R)$. In other words, each

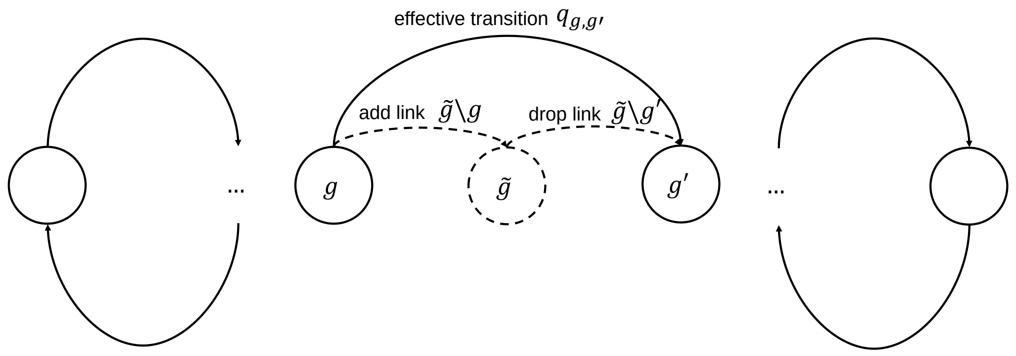

effective transition $q_{g,g'}$
add link $\tilde{g}\backslash g$
drop link $\tilde{g}\backslash g'$
g
$\tilde{g}$
g'
...
...

Figure 4.3: Illustration of an Markov chain design. At state g, a single node v adds a link $\tilde{g} \setminus g$, and the topology graph transits to an intermediate state $\tilde{g}$. The same node v then immediately drops the link $\tilde{g} \setminus g'$. Only such transitions have non-zero rates. All other transitions where multiple nodes drop multiple links have zero transition rates.

topology graph g can be represented uniquely by the set of directed overlay routes, and vice versa. Therefore, we use g and the set of the corresponding directed routes R interchangeably.

To simplify the design, we allow non-zero transition rates from topology graph g to graph g' if and only if they satisfy the following set of conditions, which we name as the "direct-transition condition". Specifically for any topology graphs (or set of overlay routes) g and g', $q_{g,g'}$ is non-zero if and only if for the union of their routes $\tilde{g} = g \cup g'$:

- $|\tilde{g} \setminus g| = 1$ and $|\tilde{g} \setminus g'| = 1$;
- the only overlay route in $\tilde{g} \setminus g$ and that in $\tilde{g} \setminus g'$ should originate from the same node denoted by $v(g, g')$.

In other words, we allow transitions to happen only when a single node adds a not-in-use neighbor and then drops one active neighbor. Figure 4.3 illustrates this idea. $\tilde{g} = g \cup g'$ is a transient state where a node has just added a new neighbor before choking one of the old neighbors[3].

4.3.1 Solution Seed

We begin by constructing a Markov chain that solves (4.4) exactly. This Markov chain does not lead to a fully distributed solution, but serves as a seed to our distributed algorithm to be discussed in later subsections. Recall that with the previous design of direct transition condition, we now need to only design the transition rates.

Proposition 3. *The following transition rates ensures the MC over the topology graph to reach the stationary distribution in* (4.6).

$$q_{g,g'} = \tau^{-1} \cdot \frac{\exp(\mu(\Phi_{g'} - \Phi_{\tilde{g}}))}{1 + \sum_{g'' \in \mathcal{A}_{v(g,g'),\tilde{g}}} \exp(\mu(\Phi_{g''} - \Phi_{\tilde{g}}))}, \tag{4.9}$$

[3]The node can temporarily violate the node degree bound, but the process happens instantaneously and the transient state is almost non-existent. We use it only to simplify our theoretical discussions.

if g and g' satisfy the direct-transition condition, and $q_{g,g'} = 0$ otherwise, where $\tilde{g} = g \cup g'$ is the transient state just after the node adds a new link and before the node drops one of the old links, $\tau > 0$ is a constant, $\mathcal{A}_{v,\tilde{g}}$ is the set of topology graphs derived by node v dropping one of its active links under graph $\tilde{g}$, i.e.,

$$\mathcal{A}_{v,\tilde{g}} = \left\{\hat{g} \in G \mid \hat{g} = \tilde{g} \backslash \left\{(v,u)\right\}, \forall u \in N_v^{\tilde{g}}\right\}, \tag{4.10}$$

where (v,u) is a directed link from node v to node u, and $v(g,g')$ is the node defined in direct-transition condition.

Proof. It is clear that all topology graphs can reach each other within a finite number of transitions, and therefore the constructed Markov chain is irreducible. Further, it is a finite state ergodic Markov chain with a unique stationary distribution. Based on the transition rate specified in (4.9), we see that

$$p_g^* q_{g,g'} = p_{g'}^* q_{g',g}, \forall g, g' \in G,$$

i.e., the detailed balance equations hold. Thus the constructed Markov chain is time-reversible and its stationary distribution is indeed (4.6) according to Theorem 1.3 and Theorem 1.14 in [31]. □

We call this MC an exact MC because it solves the entropy approximated problem (4.4) exactly. To implement it, let us first define some terms. For each node $v \in U \cup H$, we add a virtual neighbor v^{Null} and a directed virtual link from v to v^{Null}. Virtual node v^{Null} is not counted as a node degree for node v.

The introduction of virtual nodes and virtual links are needed because when a node v intends to add a neighbor w, it is possible that node w has already reached his degree bound. In this case, node v will not drop any of its active neighbors. To facilitate the discussions though, we say that in this case it is as if node v drops a virtual neighbor. The implementation is given below.

- The following procedure runs on each individual node independently. We focus on a particular node $v \in U \cup H$.
- **Initialization:** Node v randomly selects B_v neighbors from its neighbor list Q_v and builds connections with these selected neighbors.
- **Step 1:** Denote by g the current topology graph. Node v independently generates an exponentially distributed random number with mean $\tau/(|Q_v| - |N_v^g|)$ and counts down to zero from this number.
- **Step 2:** When the count-down expires, node v random uniformly chooses a new inactive neighbor w from $Q_v \setminus N_v^g$. If adding w does not violate the node degree bound of w or v, then node v builds a connection with w. otherwise, node v does not add any neighbor. The system transits to a temporary topology graph $\tilde{g}$.
- **Step 3**: Node v collects global information $\Phi_{g'} - \Phi_{\tilde{g}}$ for every "neighboring" topology graph $g' \in \mathcal{A}_{v,\tilde{g}}$. Then node v drops an active neighbor u (including the virtual neighbor) with probability

$$\frac{\exp(\mu(\Phi_{g'} - \Phi_{\tilde{g}}))}{1 + \sum\limits_{g'' \in \mathcal{A}_{v(g,g'),\tilde{g}}} \exp(\mu(\Phi_{g''} - \Phi_{\tilde{g}}))}, \tag{4.11}$$

where $g' = \tilde{g}\backslash\{(v,u)\}$. Node v then repeats **Step 1**.

In Step 3, with probability $\frac{1}{1+\sum\limits_{g''\in\mathcal{A}_{v(g,g'),\tilde{g}}}\exp(\beta(\Phi_{g''}-\Phi_{\tilde{g}}))}$ node v will not drop any one of its active neighbors, equivalently to saying that node v drops its virtual neighbor v^{Null} or node v chokes its virtual link $v \to v^{Null}$.

We know that for any direct transition from g to g', there exists a temporary state $\tilde{g} = g \cup g'$, a node v adding a not-in-use neighbor (or its virtual neighbor) and then dropping one active neighbor (or its virtual neighbor). Since the count-down rate for node v in g is $(|Q_v| - |N_v^g|)/\tau$, the probability for node v to choose an inactive neighbor w is $1/(|Q_v| - |N_v^g|)$, and the probability for node v to choke u is

$$\frac{\exp\left(\mu\left(\Phi_{g'} - \Phi_{\tilde{g}}\right)\right)}{1 + \sum\limits_{g''\in\mathcal{A}_{v,\tilde{g}}} \exp\left(\mu\left(\Phi_{g''} - \Phi_{\tilde{g}}\right)\right)}, \tag{4.12}$$

it follows that the transition rate from g to g' is

$$\begin{aligned} q_{g,g'} &= \tau^{-1}(|Q_v| - |N_v^g|) \cdot \frac{1}{|Q_v| - |N_v^g|} \\ &\cdot \frac{\exp\left(\mu\left(\Phi_{g'} - \Phi_{\tilde{g}}\right)\right)}{1 + \sum\limits_{g''\in\mathcal{A}_{v,\tilde{g}}} \exp\left(\mu\left(\Phi_{g''} - \Phi_{\tilde{g}}\right)\right)} \\ &= \tau^{-1} \cdot \frac{\exp\left(\mu\left(\Phi_{g'} - \Phi_{\tilde{g}}\right)\right)}{1 + \sum\limits_{g''\in\mathcal{A}_{v,\tilde{g}}} \exp\left(\mu\left(\Phi_{g''} - \Phi_{\tilde{g}}\right)\right)} \end{aligned} \tag{4.13}$$

Although the above algorithm can achieve the optimal solution of the entropy-approximated problem, it requires every node v to know global statistics $\Phi_{g'} - \Phi_{\tilde{g}}$ for all $g' \in \mathcal{A}_{v,\tilde{g}}$. Recall that $\Phi_{g'} = \sum_{u\in U} V^*(z_u)$ is the optimal solution of the content placement and link rate allocation problem in (3.6), which is the aggregate optimal utility of all users. However, node v will not be able to know this information for all $g' \in \mathcal{A}_{v,\tilde{g}}$ before dropping the corresponding links. Even when it needs to estimate the values of $\Phi_{g'}$ for all $g' \in \mathcal{A}_{v,\tilde{g}}$, it still needs every node in the system to send their utility to some central server, and then let the central server forward it to node v. This is impractical because sending such information back and forth burdens the system with huge overhead. There are also potential issues with the timing and accuracy of each utility reported by each user.

In the following, we design a distributed algorithm to overcome these issues. The key idea is to approximate each $\Phi_{g'} - \Phi_{\tilde{g}}$ with a local statistics. We will call the resulted MC the perturbed MC.

4.3.2 Perturbed Markov Chain

Recall that when $V(z_u) = \min(\gamma_m, z_u)$, Φ_g is equal to the sum of all (converged) utilities from users. In this case, when the overlay link in $\tilde{g}\backslash g'$ is dropped, the performance difference $\Phi_{g'} - \Phi_{\tilde{g}}$ lies in $\left[-\bar{x}_{\tilde{g}\backslash g'}, 0\right]$, where $-\bar{x}_{\tilde{g}\backslash g'}$ is the overlay link rate before it is dropped. If the cache node that drops the link $\tilde{g}\backslash g'$ cannot re-utilize the capacity resource freed up $\tilde{g}\backslash g'$, then the performance difference $\Phi_g - \Phi_{\tilde{g}} = -\bar{x}_{\tilde{g}\backslash g'}$, and this is the worst case that can happen. In the case that the system can fully re-utilize the released capacity resource, $\Phi_g - \Phi_{\tilde{g}} = 0$.

In general, the differences are not zero and follow some distribution, in which case the stationary distribution will be different from that of exact Markov chain. However, we will show that under some minor assumptions, the system will operate within a small gap to the global optimum even when the differences are not zero.

For any direct transition from g to g', define the perturbation error as $\omega_{g,g'} = [\Phi_{g'} - \Phi_{\tilde{g}}] - (-\frac{1}{2}x_{\tilde{g}\backslash g'})$ where $\tilde{g} = g \cup g'$. Recall that $-x_{\tilde{g}\backslash g'} \leq \Phi_{g'} - \Phi_{\tilde{g}} \leq 0$, therefore $-\frac{1}{2}x_{\tilde{g}\backslash g'} \leq \omega_{g,g'} \leq \frac{1}{2}x_{\tilde{g}\backslash g'}$.

Proposed in [2], the quantization error model can be applied to characterize the impacts of one-dimension perturbation errors, where perturbation error ω_g is only dependent on g for any topology graph $g \in G$. However, in this case, perturbation errors are two-dimensional, i.e., perturbation error $\omega_{g,g'}$ depends on both topologies g and g'. Therefore, we cannot apply the quantization error model directly.

Therefore, prior to study the stationary behavior of the perturbed MC, let us first transform the exact MC to a new extended-state MC model. The new extended MC can then easily handle the original two-dimension perturbation errors by effectively transforming them into one-dimension perturbation errors.

4.3.3 Exact Markov Chain with Extended States

To construct the extended-state MC, given an exact Markov chain, for any two different topology graphs $g, g' \in G$ with direct transition rates $q_{g,g'} \neq 0$, we remove the links $g \rightleftharpoons g'$ in state diagram, add an intermediate state $\tilde{g} = g \cup g' \in \tilde{G}$, two extended states $\bar{g}, \bar{g}'$ and four bidirectional links $g \rightleftharpoons \bar{g}, \bar{g} \rightleftharpoons \tilde{g}, \tilde{g} \rightleftharpoons \bar{g}', \bar{g}' \rightleftharpoons g'$. The system utility of the extended states $\bar{g}$ and $\bar{g}'$ are defined as $\Phi_{\bar{g}} = \Phi_g - \Phi_{\tilde{g}}$ and $\Phi_{g'} - \Phi_{\tilde{g}}$ respectively. In this way, any perturbation errors incurred by estimating the utility difference $\Phi_g - \Phi_{\tilde{g}}$ can be regarded as perturbation errors incurred by estimating the utility $\Phi_{\bar{g}}$ of state $\bar{g}$. An extended-state Markov chain is illustrated in Figure 4.4.

We denote the set of all extended states as $\bar{G}$. The new non-zero direct transition rates on extended state space are shown as follows:

$$q_{g,\bar{g}} = 1 \tag{4.14}$$

$$q_{\bar{g},g} = \bar{\alpha} \cdot \exp(\mu \Phi_{\bar{g}}) \tag{4.15}$$

$$q_{\bar{g},\tilde{g}} = 1 \tag{4.16}$$

$$q_{\tilde{g},\bar{g}} = \tilde{\alpha} \frac{1}{1 + \sum_{g'' \in \mathcal{A}_{v(g,g'),\tilde{g}}} \exp\left(\mu\left(\Phi_{\bar{g''}}\right)\right)} \tag{4.17}$$

$$q_{\tilde{g},g'} = \tilde{\alpha} \frac{1}{1 + \sum_{g'' \in \mathcal{A}_{v(g,g'),\tilde{g}}} \exp\left(\mu\left(\Phi_{\bar{g''}}\right)\right)} \tag{4.18}$$

$$q_{\bar{g'},\tilde{g}} = 1 \tag{4.19}$$

$$q_{\bar{g'},g'} = \bar{\alpha} \cdot \exp(\mu \Phi_{\bar{g'}}) \tag{4.20}$$

$$q_{g',\bar{g}} = 1 \tag{4.21}$$

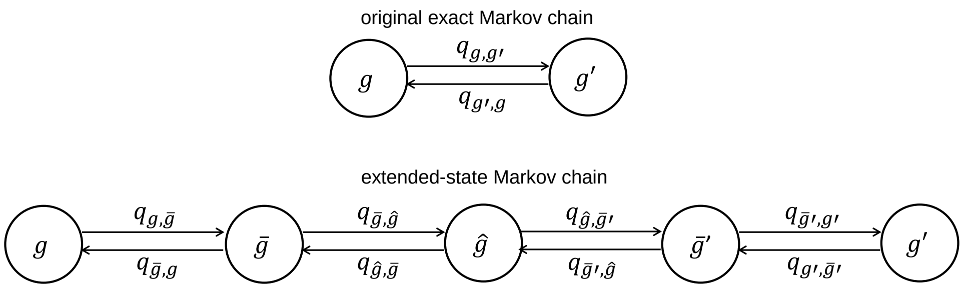


Figure 4.4: An example of the original two-state exact Markov chain and the extended state Markov chain. In the original exact Markov chain, the transition rates are assigned according to (4.6). Extended state Markov chain is obtained by adding one intermediate state $\tilde{g} = g \cup g'$, two extended states $\bar{g}, \bar{g'}$ and four bidirectional links $g \rightleftharpoons \bar{g}, \bar{g} \rightleftharpoons \tilde{g}, \tilde{g} \rightleftharpoons \bar{g'}, \bar{g'} \rightleftharpoons g'$ among them. The transition rates are assigned according to (4.14)-(4.21).

where $\bar{\alpha}, \tilde{\alpha}$ are positive constants.

Denote this extend-state Markov chain by M^e and its state space by $\mathcal{C}$. We have the following result:

Proposition 4. *The extend-state Markov chain M^e is time-reversible with a unique stationary distribution $\{p_g^e, g \in \mathcal{C}\}$. When both $\bar{\alpha}$ and $\tilde{\alpha}$ approach infinity, the state space $\mathcal{C}$ degenerates into G and extend-state Markov chain M^e degenerates into the original exact Markov chain with stationary distribution* (4.6).

Proof. First, given extended-state Markov chain M^e, let the stationary distribution of state j be denoted by π_j. For any two states j, j' satisfying detailed balance equation, we have $\pi_j q_{j,j'} = \pi_{j'} q_{j',j}$. Let $\rho_{j,j'} = q_{j,j'}/q_{j',j}$ for any $q_{j',j} \neq 0$, and then $\pi_{j'} = \pi_j \rho_{j,j'}$.

Second, for the state-transition diagram of extended state Markov chain M^e, we map it to an undirected graph G^m=(V^m, E^m), where the node set $V^m = \mathcal{C}$ is the set of states and any edge $e(i,j) \in E^m, i,j \in V^m$ represents the state-pair (i,j) with $q_{i,j} \neq 0$.

It can be shown that G^m is a connected graph. Therefore, we can always find a spanning tree to connect all nodes in G^m and there exists only one path between any pair of nodes. Suppose we have constructed a spanning tree on G^m, denoted by T^m. Denote the root state by state 0, and the nodes in V^m by states $1, 2, \ldots, |V^m| - 1$, according to the breadth-first search order on the spanning tree. Let path(0, i) be the path between state *0* and state *i* ($1 \leq i \leq |V^m| - 1$), passing $m_i + 1$ number of states (including states *0* and *i*). We order the nodes on path(0,i) according to their distances from state 0, and denoted them by $v_{0,i}^j (0 \leq j \leq m_i)$.

According to detailed balanced equations along the path(0, i), we have the following:

$$\pi_i = \pi_0 \cdot \prod_{j=0}^{m_i-1} \rho_{v_{0,i}^j, v_{0,i}^{j+1}}, \quad 1 \le i \le |V^m| - 1 \tag{4.22}$$

$$\pi_0 + \sum_{i=1}^{|V^m|-1} \pi_i = 1 \tag{4.23}$$

Then we get the distribution

$$\pi_0 = \frac{1}{1 + \sum\limits_{k=1}^{|V|-1} \prod\limits_{j=0}^{m_k-1} \rho_{v_{0,k}^j, v_{0,k}^{j+1}}} \tag{4.24}$$

$$\pi_i = \frac{\prod_{j=0}^{m_i-1} \rho_{v_{0,i}^j, v_{0,i}^{j+1}}}{1 + \sum\limits_{k=1}^{|V|-1} \prod\limits_{j=0}^{m_k-1} \rho_{v_{0,k}^j, v_{0,k}^{j+1}}}, \quad 1 \le i \le |V| - 1 \tag{4.25}$$

We now verify the distribution computed based on the spanning tree, i.e., (4.24)-(4.25), is the correct stationary distribution, by testing the detailed balance equations between any two states $j, j' \in V$.

1. If $q_{j,j'} = 0$, then the detailed balance equation trivially holds.

2. If $q_{j,j'} \neq 0$ and the edge $e(j, j')$ belongs to the spanning tree, then by (4.25), we know that $\pi_{j'} = \pi_j \rho_{j,j'}$, i.e., the detailed balance equation holds.

3. If $q_{j,j'} \neq 0$ and the edge $e(j, j')$ does not belong to the spanning tree, then we focus on the cycle consisting of path(j',j) and $e(j, j')$. Starting from node $j' = v_{j',j}^0$, we can visit nodes $v_{j',j}^k, 1 \le k \le m_{j',j} - 1$ and node $j = v_{j',j}^{m_{j',j}}$ in sequence along the path(j',j). Thus we have

$$\rho_{j',j} = \prod_{k=0}^{m_{j',j}-1} \rho_{v_{j',j}^k, v_{j',j}^{k+1}} \tag{4.26}$$

By (4.25) and (4.26) we have

$$\frac{\pi_j}{\pi_{j'}} = \prod_{k=0}^{m_{j',j}-1} \rho_{v_{j',j}^k, v_{j',j}^{k+1}} = \rho_{j',j} \tag{4.27}$$

Therefore, the detailed balance equation between *j* and *j'* holds.

Combining the above scenarios, we know that the detailed balance equations between any two states $j, j' \in V^m$ hold. By [31] we know that the distribution shown in (4.24)-(4.25) is indeed the stationary distribution and extended state Markov chain M^e is time-reversible.

When both $\bar{\alpha}$ and $\tilde{\alpha}$ approach infinity, by (4.14)-(4.21), we know that the rates leaving intermediate states and extended states are also approaching infinity, making these states transient. In this way, the state space $\mathcal{C}$ degenerates into G and extend-state Markov chain M^e degenerates into the original exact Markov chain with stationary distribution in (4.5). $\square$

4.3.4 Impacts of Perturbation Errors

Now we proceed to study the stationary distribution given the perturbation errors $\omega_{g,g'}$ by utilizing the extended-state MC. For perturbation of Markov chain M^e, given any state $g \in \mathcal{C}$, there is only one dimension perturbation error ω_g depending on state g only. Note that two-dimension perturbation errors $\omega_{g,g'}$ in original exact Markov chain is mapped to one-dimension perturbation error $\omega_{\bar{g}}$ in Markov chain M^e. Therefore, we can apply quantization error model proposed in [2] to time-reversible Markov chain M^e.

Given the utility function $V(z_u) = \min(\gamma_m, \sum x_r), u \in U_m$, we know that the perturbation error satisfies $-\frac{1}{2}x_{\tilde{g}\backslash g'} \leq \omega_{g,g'} \leq \frac{1}{2}x_{\tilde{g}\backslash g'}$. However we know that $-\bar{x}_{\tilde{g}\backslash g'} \leq c_{\max}$ for any g, g', where c_{max} is the maximum link rate over all physical links. There fore, we can assume $\omega_g, \forall g \in \mathcal{C}$ is bounded in the region $[-\frac{c_{max}}{2}, \frac{c_{max}}{2}]$. We quantize such error ω_g into $2n_g + 1$ values $[-\frac{c_{max}}{2}, \dots, 0, \frac{c_{max}}{2n_g}, \frac{c_{max}}{n_g}, \dots, \frac{c_{max}}{2}]$, and assume that $\omega_g = \frac{kc_{max}}{2n_g}$ with probability $\xi_{f,k}, k = -n_g, \dots, n_g$ and $\sum_{k=-n_g}^{n_g} \xi_{f,k} = 1$. Then we have the following lemma:

Lemma 5. *(a) The stationary distribution of perturbed extend-state Markov chain is given by:*

$$\bar{p}^e_g = \frac{\sigma_g \exp(\mu\Phi_g)}{\sum_{g' \in \mathcal{C}} \sigma_{g'} \exp(\mu\Phi_{g'})} \tag{4.28}$$

where $\sigma_g = \sum_{k=-n_g}^{n_g} \xi_{g,k} \exp\left(\mu \frac{kc_{max}}{2n_g}\right), \forall g \in \mathcal{C}$.
(b) When both $\bar{\alpha}$ and $\tilde{\alpha}$ approach infinity, the state space $\mathcal{C}$ degenerates into G and perturbed extend-state Markov chain degenerates into perturbed Markov chain. The stationary distribution of perturbed Markov chain is given by

$$\bar{p}_g = \frac{\sigma_g \exp(\mu\Phi_g)}{\sum_{g' \in G} \sigma_{g'} \exp(\mu\Phi_{g'})}, \forall g \in G \tag{4.29}$$

where

$$\sigma_g = \sum_{k=-n_g}^{n_g} \xi_{g,k} \exp\left(\mu \frac{kc_{\max}}{2n_g}\right), \forall g \in G. \tag{4.30}$$

Proof. Consider a modified MC as follows: expand each state g of the extended MC M^e to $2n_g + 1$ states $g_k, k = -n_g, \dots, 0, 1, \dots, n_g$ with the following transition rates:

$$\bar{q}_{g_k, g'_{k'}} = \frac{\xi_{g'_{k'}}}{\exp\left(\kappa\left(\Phi_g - \Phi_{\bar{g}} + \frac{kc_{\max}}{2n_g}\right)\right)}, \tag{4.31}$$

where $\xi_{g'_{k'}}, k = -n_{g'}, \dots, 0, 1, \dots, n_{g'}$, is the probability measure on expanded states and $\sum_{k'=-n_{g'}}^{n_{g'}} \xi_{g'_{k'}} = 1$. Note that g_0 refers to state g with zero error. Using equation (4.31) and detailed balance equations $p_{g_k}\bar{q}_{g_k,g'_{k'}} = p_{g'_{k'}}\bar{q}_{g'_{k'},g_k}$, we have $\forall g_0, g'_{k'}$:

$$\frac{p_{g_0}}{\xi_{g_0} \exp(\kappa\Phi_g)} = \frac{p_{g'_{k'}}}{\xi_{g'_{k'}} \exp\left(\kappa\left(\Phi_{g'} + \frac{k'c_{\max}}{n_{g'}}\right)\right)} = const. \tag{4.32}$$

Using $\sum_{g\in G}\sum_{k=-n_g}^{n_g} p_{g_k}=1$, we obtain:

$$p_{g_k}=\frac{\xi_{g_k}\exp\left(\kappa\left(\Phi_g+\frac{kc_{\max}}{n_g}\right)\right)}{\sum_{g'\in G}\sum_{k'=-n_{g'}}^{n_{g'}}\xi_{g'_{k'}}\exp\left(\kappa\left(\Phi_{g'}+\frac{k'c_{\max}}{n_{g'}}\right)\right)}. \tag{4.33}$$

Denote by $\sigma_g=\sum_{k=-n_g}^{n_g}\xi_{g_k}\exp\left(\kappa\frac{kc_{\max}}{n_g}\right)$ and we have:

$$\bar{p}_g^e=\sum_{k=-n_g}^{n_g} p_{g_k}=\frac{\sigma_g\exp\left(\kappa\Phi_g\right)}{\sum_{g'\in G}\sigma_{g'}\exp\left(\kappa\Phi_{g'}\right)}. \tag{4.34}$$

Part (b) follow the same proof concepts used in Proposition 4, which we omit here. □

4.3.5 Soft Worst Neighbor Choking

Now that we have fully understood the perturbed MC, the soft-worst-neighbor-choking topology graph selection algorithm is straightforward. Let g denote the current topology graph. Each cache node $v\in H$ (and similarly for user node $u\in U$) implements the algorithm as follows.

- **Initialization:** It randomly selects and builds connections with B_v (which is its maximum degree) number of neighbors from its neighbor list Q_v. Denote by N_v^g the connected neighbors.

- **Step 1:** It counts down to zero from an exponential timer with mean $T=\tau/(|Q_v|-|N_v^g|)$, where $\tau>0$ is a constant[4].

- **Step 2:** When the count-down expires, it randomly chooses a new inactive neighbor w from $Q_v\setminus N_v^g$, and requests to connect to it. If the node degree bound of neither node w nor v is violated, the connection is established; otherwise, no new connection is made. The system transits to a temporary topology graph $\tilde{g}$.

- **Step 3**: Node v measures the overlay link rate $\bar{x}_{(v,u)}$ from every neighbor $u\in N_v^{\tilde{g}}$ (including the newly added neighbor w if there is any), and then chokes an in-use neighbor u with probability

$$\frac{\exp(-\frac{1}{2}\mu\bar{x}_{\tilde{g}\backslash g'})}{1+\sum\limits_{g''\in\mathcal{A}_{v,\tilde{g}}}\exp(-\frac{1}{2}\mu\bar{x}_{\tilde{g}\backslash g''})},\quad g'=\tilde{g}\backslash\{(v,u)\}\,. \tag{4.35}$$

  Afterwards, node v repeats **Step 1**.

The above algorithm is fully distributed – each node uses the overlay link rates from his neighbors as the only metric to perform the topology selection. Recall that in the worst-neighbor-choking algorithm in the toy example in Figure 4.1, the worst link is always choked. The only difference here is that the worst link is dropped with high probability, hence the term "soft-worst" choking.

[4]It is a tuning parameter which affects the count-down time and the algorithm's mixing time. Details will be given in the subsequent section.

Proposition 6. *The soft-worst-neighbor-choking algorithm induces the following transition rates for the perturbed Markov chain: for any two topology graphs $g, g' \in G$ satisfying direct-transition condition:*

$$q_{g,g'} = \tau^{-1} \cdot \frac{\exp(-\frac{1}{2}\mu \bar{x}_{\tilde{g}\backslash g'})}{1 + \sum\limits_{g'' \in \mathcal{A}_{v(g,g'),\tilde{g}}} \exp(-\frac{1}{2}\mu \bar{x}_{\tilde{g}\backslash g''})}, \tag{4.36}$$

where $\bar{x}_{\tilde{g}\backslash g'}$ is the rate of the only overlay link in $\tilde{g}\backslash g'$ under topology graph $\tilde{g}$; and $q_{g,g'} = 0$ otherwise.

The proof will be similar to that for the solution seed, which we will omit in this case. Comparing (4.36) with (4.9), we can see that the global quantity $\Phi_{g'} - \Phi_{\tilde{g}}$, which is the overall utility difference between after and before the node drops the overlay link $\tilde{g}\backslash g'$, is replaced by $-\frac{1}{2}\bar{x}_{\tilde{g}\backslash g'}$ which is a locally measurable quantity, thanks to our intuition from the worst-neighbor-choking algorithm derived from the toy example. Fortunately, we can show that under some reasonable assumptions, the perturbed MC can still achieve close-to-optimal system performance, as we show in Chapter 5.

4.4 Constant Countdown Time

In the previous section, we have a Markov chain based topology graph selection algorithm in which each node needs to count down following an exponential distribution. We observe in this section an insensitivity property of the algorithm: the stationary distribution of each topology graph depends on node's count-down time distribution only through its mean, i.e., as long as the mean values of the countdown time in the neighbor choking algorithm are kept, the stationary distribution of the MC holds regardless of the underlying distribution of the countdown time. As a special case, the countdown time can be a constant which is very easy to implement. In fact, BitTorrent's tit-for-tat algorithm uses exactly a constant countdown time in its tit-for-tat neighbor selection algorithm.

In the following, we present the insensitivity result of the countdown time distribution.

Proposition 7. *In implementations of exact Markov chain based topology selection algorithm, for each node v with current topology graph g, if we change the distribution of its count-down time from exponential distribution to general distribution and keep the same mean of countdown time $\tau/(|Q_v| - |N_v^g|)$, then the stationary distribution of any topology $g \in G$ is still p_g^* in* (4.6).

Proof. Suppose that within any topology $g \in G$, the count-down time for each node $v \in U \cup H$ are independent with probability density function $l_{v,g}$ and mean $\tau/(|Q_v| - |N_v^g|)$. Previously we model the node neighboring state as node topology $g \in G$. This model is complete under the exponential count-down time assumption memoryless property of the exponential distribution. In general, for each topology $g \in G$, we define an extended state extended state $Y_g = (g, \{R_v(g), v \in U \cup H\})$, where $R_v(g) \in [0, +\infty), \forall v \in U \cup H$ is the residual count-down time. $R_v(g)$ decreases continuously and its rate of decrease is $\frac{dR_v(g)}{dt} = -1$. Since there are infinite possible values for state space is infinite. Therefore $Y = \{Y_g, g \in G\}$ is a continuous-state Markov process and we denote $y = (g, \{r_v(g), v \in U \cup H\})$ as one realization.

Let $p_Y(t, y)$ be the state probability density at time t. Its derivative with respect to t is

$$\frac{dp_Y(t, y)}{dt} = \lim_{\triangle t \to 0} \frac{p_Y(t + \triangle t, y) - p_Y(t, y)}{\triangle t} \tag{4.37}$$

At the time interval from t to $t + \triangle t$, the state changes as a result of node finishing counter-down and node continuing counter-down. There is only one type of jump events that cause a discontinuity in the evolution of y: a node finishing count-down. For small values of $\triangle t$, multiple jump events occur with probability in order $o(\triangle t)$ and can be disregarded. Between the jump events, nodes are continue counting down, in which case y changes continuously without g being changed. For a particular realization y, we have

$$p_Y(t + \triangle t, y) = A + B + o(\triangle t), \tag{4.38}$$

where A is the contribution due to count-down-to-zero jump events, and B is the contribution due to ordinary counting down without any jump events, and $\lim_{\triangle t \to 0} \frac{o(\triangle t)}{\triangle t} = 0$.

(a) Let $y = (g, \{r_v(g)\}_{v \in U \cup H})$ be the state at $t + \triangle t$. Then we have

$$A = \sum_{v \in U \cup H} p_Y(t, RC_{v0^+}(y)) l_{v,g}(r_v(g)) \triangle t \tag{4.39}$$

where $RC_{v0^+}(y)$ is the operation that sets $r_v(g)$ in g to be 0^+ (i.e, just before the counter-down of node v completes), and $l_{v,g}(r_v(g))$ is the probability density of a newly generated count-down time for node v.

(b) Let $y = (g, \{r_v(g)\}_{v \in U \cup H})$ be the state at $t + \triangle t$ and suppose that no count-down-to-zero events occur during the interval from t to $t + \triangle t$. Then the state at time t must have been $y = (g, \{r_v(g) + \triangle t\}_{v \in U \cup H}$, for the $r_v(g)$ decrease at rate -1. Therefore,the contribution is $B = p_Y(t, (g, \{r_v(g) + \triangle t\}_{v \in U \cup H}$. By expanding in a Taylor series about each $r_v(g)$, we have

$$B = p_Y(t, (g, \{r_v(g) + \triangle t\}_{v \in U \cup H} \tag{4.40}$$

$$= p_Y(t, y) + \sum_{v \in U \cup H} \frac{\partial p_Y(t, y)}{\partial r_v(g)} \triangle t + o(\triangle t). \tag{4.41}$$

Putting (4.39) and (4.41) into (4.38), and applying the definition of derivative in (4.37), we have

$$\frac{dp_Y(t, y)}{dt} = \sum_{v \in U \cup H} \left[p_Y(t, RC_{v0^+}(y)) l_{v,g}(r_v(g)) \; + \frac{\partial p_Y(t, y)}{\partial r_v(g)} \right]. \tag{4.42}$$

In stationarity, the derivative with respect to time t cancel, so that $\frac{dp_Y(t,y)}{dt} = 0$ and we have the following balance equation

$$\sum_{v \in U \cup H} \left[p_Y(RC_{v0^+}(y)) l_{v,g}(r_v(g)) \; + \frac{\partial p_Y(y)}{\partial r_v(g)} \right] = 0 \tag{4.43}$$

Next, we will show that the stationary probability density of Y is:

$$p_Y(y) = p_g^* \prod_{v \in U \cup H} \frac{(1 - \int_0^{r_v(g)} l_{v,g}(t) dt)}{\frac{\tau}{(|Q_v| - |N_v^g|)}} \tag{4.44}$$

where p_g^* is given by (4.6).

In other words, we will show that the stationary probability density in (4.44) satisfies the balance equation (4.43). In fact, we will show that

$$\left[p_Y(RC_{v0^+}(y))l_{v,g}(r_v(g)) + \frac{\partial p_Y(y)}{\partial r_v(g)}\right] = 0, \forall v \in U \cup H. \tag{4.45}$$

Under (4.44), $\forall v \in U \cup H$,

$$\begin{aligned} & -\frac{\partial p_Y(y)}{\partial r_v(g)} \\ & = -\frac{d\frac{\left(1-\int_0^{r_v(g)} l_{v,g}(t)dt\right)}{\frac{\tau}{(|Q_v|-|N_v^g|)}}}{dr_v(g)} \cdot p_g^* \cdot \prod_{v' \in U \cup H - \{v\}} \frac{(1-\int_0^{r_{v'}(g)} l_{v',g}(t)dt)}{\frac{\tau}{(|Q_v|-|N_v^g|)}} \quad (4.46) \\ & = \frac{l_{v,g}(r_v(g))}{\frac{\tau}{(|Q_v|-|N_v^g|)}} \cdot p_g^* \cdot \prod_{v' \in U \cup H - \{v\}} \frac{(1-\int_0^{r_{v'}(g)} l_{v',g}(t)dt)}{\frac{\tau}{(|Q_v|-|N_v^g|)}}. \quad (4.47) \end{aligned}$$

On the other hand, in state y, $\forall v \in U \cup H$, it is not hard to see the probability density function of residual count-down time $r_v(g)$ is $\frac{(1-\int_0^{r_v(g)} l_{v,g}(t)dt)}{\frac{1}{\tau(|Q_v|-|N_v^g|)}}$ [39, 53]. Then we have

$$\begin{aligned} & p_Y(RC_{v0^+}(y))l_{v,g}(r_v(g)) \\ & = l_{v,g}(r_v(g)) \cdot \frac{(1-\int_0^{0^+} l_{v,g}(t)dt)}{\frac{1}{\tau(|Q_v|-|N_v^g|)}} \cdot p_g^* \\ & \cdot \prod_{v' \in U \cup H - \{v\}} \frac{(1-\int_0^{r_{v'}(g)} l_{v',g}(t)dt)}{\frac{1}{\tau(|Q_v|-|N_v^g|)}} \quad (4.48) \\ & = \frac{l_{v,g}(r_v(g))}{\frac{1}{\tau(|Q_v|-|N_v^g|)}} \cdot p_g^* \cdot \prod_{v' \in U \cup H - \{v\}} \frac{(1-\int_0^{r_{v'}(g)} l_{v',g}(t)dt)}{\frac{1}{\tau(|Q_v|-|N_v^g|)}}. \quad (4.49) \end{aligned}$$

Thus by comparing (4.47) and (4.49), we know that

$$-\frac{\partial p_Y(y)}{\partial r_v(g)} = p_Y(RC_{v0^+}(y))l_{v,g}(r_v(g)), \forall v \in U \cup H. \tag{4.50}$$

Therefore, the stationary probability density in (4.44) satisfies the balance equation (4.43).

By integrating $p_Y(y)$ in (4.44) overall all possible values of $r_v(g), \forall v \in U \cup H$, we see that the stationary distribution for any topology graph $g \in G$ is p_g^* in (4.5). This means the stationary distribution of topology graph g is insensitive to the distribution of count-down times. This concludes the proof. In the same way, similar insensitivity results can be obtained for implementations of perturbed Markov chain based topology graph hopping algorithm. □

Since the countdown time is insensitive to its distribution as long as its mean is preserved, it offers great simplicity when implemented in practice. As a special case, each node can use a constant countdown time. In fact, in BitTorrent's tit-for-tat neighbor selection algorithm, a constant thirty-second countdown time is used. The connections between our neighbor selection algorithm and that of BitTorrent's can be found in [54].

In summary, the soft-worst-neighbor-choking algorithm is very easy to implement in practice despite the heavy discussions, and it is analogous to the heuristic worst-neighbor-choking algorithm proposed at the beginning of this Chapter which has intuitive explanations. In the next Chapter, we show that the soft-worst-neighbor-choking algorithm achieves within a small gap of the performance achieved by the algorithm that induces the exact Markov chain, and that gap goes to zero when the number of users $|U|$ becomes large.

Chapter 5

Performance Analysis

In earlier chapters we have described the content placement, link rate allocation and topology graph selection algorithms. Despite their simpleness, they have provably performance guarantee. This chapter proves that the proposed algorithms achieve close-to-optimal solutions, which we verify via simulation results. We also briefly discuss the convergence time by providing theoretical bounds on the mixing time of the topology graph selection algorithm, and corroborate the algorithm's practicality via simulation results that show quick convergence.

5.1 Performance Guarantee

We have the following theorem of the performance guarantee of our algorithms.

Theorem 8. *Denote by $\Phi^O = \max_{g\in G}\Phi_g$ the optimal system utility, $\Phi^E = \sum_{g\in G} p_g^* \cdot \Phi_g$ the expected system utility of the exact Markov chain, and $\Phi^P = \sum_{g\in G} p_g^{'} \cdot \Phi_g$ the expected system utility of perturbed Markov chain, where p_g^* and $p_g^{'}$ are the stationary distributions of the exact and perturbed MC respectively. Let $\bar{\Phi}^O = \frac{1}{|U|}\Phi^O$, $\bar{\Phi}^E = \frac{1}{|U|}\Phi^E$ and $\bar{\Phi}^P = \frac{1}{|U|}\Phi^P$ be the corresponding system utilities averaged among the users. When the users' utility function $V(z_u) = \min(\gamma_m, \sum x_r), u \in U_m$, the optimality gaps with and without perturbation errors are shown as follows:*

$$0 \le \bar{\Phi}^O - \bar{\Phi}^E \le \frac{1}{\mu} B_{\max} \log N_{\max} \tag{5.1}$$

$$0 \le \bar{\Phi}^O - \bar{\Phi}^P \le \frac{1}{\mu} B_{\max} \log N_{\max} + \frac{1}{2|U|} c_{\max} \tag{5.2}$$

where $B_{\max}$ is the maximum degree bound over all users, $N_{\max}$ is the max neighbor size over all users, $c_{\max}$ is the maximum underlay link capacity, and $|U|$ is the total number of users.

Proof. Let $g_{\max} \in \arg\max_{g\in G}\Phi_g$ and consider the dirac distribution:

$$\hat{p}_g = \begin{cases} 1 & \text{if } g = g_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

We know that p_g^* in (4.6) is the optimal distribution for entropy-approximated problem (4.4). Therefore,

$$\sum_{g\in G} p_g^*\Phi_g - \frac{1}{\mu}\sum_{g\in G} p_g^*\log p_g^* \geq \sum_{g\in G}\hat{p}_g\Phi_g - \frac{1}{\mu}\sum_{g\in G}\hat{p}_g\log\hat{p}_g \tag{5.4}$$

$$= \Phi^O. \tag{5.5}$$

With Jensen's inequality [13] we know that

$$-\sum_{g\in G} p_g^*\log p_g^* = \sum_{g\in G} p_g^*\log\frac{1}{p_g^*} \tag{5.6}$$

$$\leq \log(\sum_{g\in G} p_g^*\cdot\frac{1}{p_g^*}) = \log|G|. \tag{5.7}$$

Combining (5.5) and (5.7), we have

$$\Phi^E = \sum_{g\in G} p_g^*\cdot\Phi_g \leq \sum_{g\in G} p_g^*\cdot\Phi^O = \Phi^O \tag{5.8}$$

$$\leq \Phi^E - \frac{1}{\mu}\sum_{g\in G} p_g^*\log p_g^* \leq \Phi^E + \frac{1}{\mu}\log|G|. \tag{5.9}$$

Therefore,

$$0 \leq \Phi^O - \Phi^E \leq \frac{\log|G|}{\mu}. \tag{5.10}$$

Next we show the bounds on optimality gap for perturbed Markov chain. By (4.29), probability distribution $\bar{p}$ can be regarded as the optimal solution to entropy-approximated problem (4.4) by changing the utility function Φ_g in (4.4) to $\Phi_g' = \Phi_g + \frac{\log\sigma_g}{\mu}$. Following the inequality in (5.1), we have

$$\max_{g'\in G}\Phi_{g'} - \sum_{g'\in G}\bar{p}_{g'}\Phi_{g'} \leq \frac{\log|G|}{\mu}. \tag{5.11}$$

Putting in the values of $\Phi_{g'}$, we have

$$\max_{g\in G}[\Phi_g + \frac{\log\sigma_g}{\mu}] - \sum_{g\in G}\bar{p}_g[\Phi_g + \frac{\log\sigma_g}{\mu}] \leq \frac{\log|G|}{\mu}. \tag{5.12}$$

By (4.30) it is not hard to see that:

$$\exp(-\mu\frac{c_{\max}}{2}) \leq \sigma_g \leq \exp(\mu\frac{c_{\max}}{2}), \forall g\in G. \tag{5.13}$$

Thus

$$-\frac{c_{\max}}{2} \leq \frac{\log\sigma_g}{\mu} \leq \frac{c_{\max}}{2}, \forall g\in G. \tag{5.14}$$

Combining (5.12) and (5.14), we have

$$\Phi^O = \max_{g \in G} \Phi_g \leq \max_{g \in G} [\Phi_g + \frac{\log \sigma_g}{\mu}] \tag{5.15}$$

$$\leq \sum_{g \in G} \bar{p}_g [\Phi_g + \frac{\log \sigma_g}{\mu}] + \frac{\log |G|}{\mu} \tag{5.16}$$

$$\leq \sum_{g \in G} \bar{p}_g \Phi_g + \frac{c_{\max}}{2} + \frac{\log |G|}{\mu}. \tag{5.17}$$

It follows that

$$\Phi^P = \sum_{g \in G} \bar{p}_g \cdot \Phi_g \leq \sum_{g \in G} \bar{p}_g \cdot \Phi^O = \Phi^O \tag{5.18}$$

$$\leq \Phi^P + \frac{\log |G|}{\mu} + \frac{c_{\max}}{2}. \tag{5.19}$$

Therefore

$$0 \leq \Phi^O - \Phi^P \leq \frac{\log |G|}{\mu} + \frac{c_{\max}}{2} \tag{5.20}$$

Since each user has at most $\binom{N_{\max}}{B_{\max}}$ number of neighboring topology graphs, where $N_{\max}$ is the maximum number of neighbors the user can see, and $B_{\max}$ is the maximum degree bound over all users, the number of topology graphs can be bounded above by $|G| \leq \binom{N_{\max}}{B_{\max}}^{|U|} \leq N_{\max}^{B_{\max}|U|}$.

Plugging in this inequality and dividing both sides of (5.10) and (5.20) by $|U|$ we have:

$$0 \leq \bar{\Phi}^O - \bar{\Phi}^E \leq \tfrac{1}{\mu} B_{\max} \log N_{\max} \tag{5.21}$$

$$0 \leq \bar{\Phi}^O - \bar{\Phi}^P \leq \tfrac{1}{\mu} B_{\max} \log N_{\max} + \tfrac{1}{2|U|} c_{\max} \tag{5.22}$$

□

We make the following observations:

- The upper bound on optimality gap of the perturbed Markov chain is $\frac{c_{\max}}{2|U|}$ away from that of the exact Markov chain, which we call "the price of local perturbation". However, this error is very small compared to the average utility of users.
- When both the number of users $|U|$ and μ approach infinity, the average system performance in both cases approaches to the optimal value of Φ_O.
- When we formulated the topology selection algorithm, we made the assumption that the underlying content placement and link rate allocation algorithms have fully converged. When this is not the case however, we obtain inaccurate values of the link rates x_{uv} and therefore $\Phi_g, g \in G$. We can treat this inaccuracy as a one-dimension perturbation error to exact system utilities $\Phi_g, g \in G$. Following the same method to the proof of Theorem 8, we can still obtain bounds on utility gap similar to those in (5.1)-(5.2).
- While larger μ reduces the optimality gap, it may also increase the mixing time of the Markov chain. We will present bounds on the mixing time in the next subsection. In practice, however, as we show in the experimental results, the algorithm is able to achieve very close to the lower bound in an extensive emulation of cases in reasonably short time.

5.2 Convergence Time

In general, as we will also show in the simulations, the convergence time is the combined content placement, link rate allocation and topology graph selection algorithms will be bottlenecked by the mixing time of the topology graph selection algorithm, which we focus in this section. We derive our analysis on the mixing time of the *exact* Markov chain, while noting that the analysis to perturbed Markov chain is a straightforward extension. First, we define mixing time of the exact Markov chain as follows:

$$t_{mix}(\epsilon) \triangleq \inf\left\{t \geq 0 : \max_{g\in G} d_{TV}(\boldsymbol{H}_t(g), \boldsymbol{p}^*) \leq \epsilon\right\} \tag{5.23}$$

where $\boldsymbol{p}^*$ in (4.6) is the stationary distribution of the exact Markov chain, $\boldsymbol{H}_t(g)$ is the probability distribution of the MC states at time t given that the initial state is g, and d_{TV} stands for the total variational distance [17] between two probability distributions:

$$d_{TV}(\boldsymbol{p}, \boldsymbol{p}') \triangleq \frac{1}{2}\sum_{g\in G}|p_g - p'_g| \tag{5.24}$$

Without loss of generality, we assume that the nodes $v \in U \cup H$ have a uniform neighbor set size $|N_v| = N$ and node degree bound $B_v = B$. We have the following results.

Theorem 9. *The mixing time $t_{mix}(\epsilon)$ of the exact Markov chain has the following bounds:*
(a) for general $\mu \in (0, \infty)$, we have

$$t_{mix}(\epsilon) \geq \frac{\exp\left(-\mu\left(\Phi^O - \Phi^{\min}\right)\right)}{2n\tau^{-1}\cdot(N-B+1)}\ln\frac{1}{2\epsilon} \tag{5.25}$$

$$t_{mix}(\epsilon) \leq \frac{2n\cdot(B+1)^2(N-B+1)}{\tau^{-1}}\binom{N}{B}^{2n}\exp(5\mu(\Phi^O - \Phi^{\min}))$$
$$\cdot\,[\ln\frac{1}{2\epsilon} + \frac{n}{2}\ln\binom{N}{B} + \frac{1}{2}\mu(\Phi^O - \Phi^{\min})] \tag{5.26}$$

where $\Phi^O = \max_{g\in G}\Phi_g$, $\Phi^{\min} = \min_{g\in G}\Phi_g$ and $n = |U \cup H|$ denotes the total number of user and cache nodes in system.
(b) When

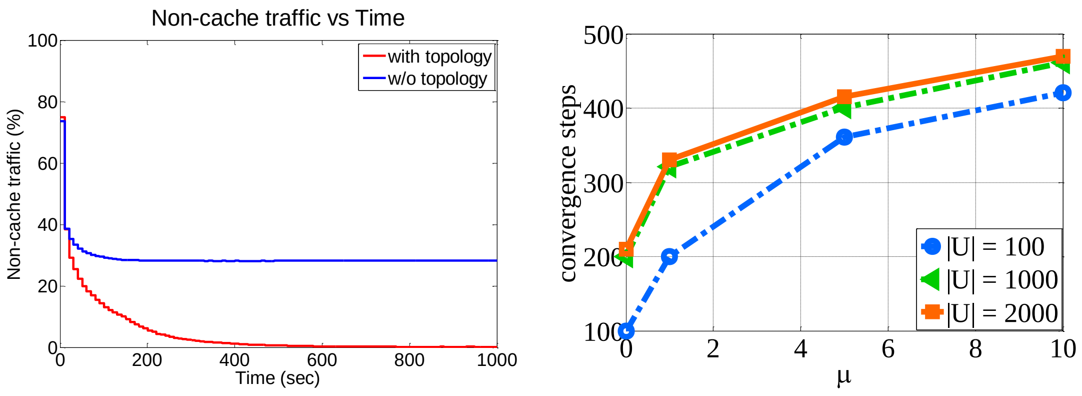
$$0 < \mu < \frac{1}{\Phi^O - \Phi^{\min}}\ln[(1+\frac{1}{B})(1+\frac{1}{N-B-1})], \tag{5.27}$$

we have a tighter upper bound:

$$t_{mix}(\epsilon) \leq \frac{\frac{1}{\tau^{-1}(N-B)}\cdot\ln\frac{nB}{\epsilon}}{1-(1-\frac{1}{N-B})\cdot(\frac{B}{B+1}\exp(\mu(\Phi^O - \Phi^{\min})))} \tag{5.28}$$

We will omit the heavy proof of this theory for conciseness, and will instead use simulation results to verify the fast convergence of the proposed algorithm. Interested readers can refer to /citeshao11tech for details.

We can see following trade-off between the optimality gap (Theorem 8) and mixing time (Theorem 9).



(a) performance (b) convergence time

Figure 5.1: (a) compares no choking and soft-worst-neighbor choking; (b) shows convergence time for various μ's and number of nodes in the system.

- As $\mu \to \infty$, the optimality gap approaches to zero while the upper bound of mixing time scales with $\exp(\Omega(n))$ (slow mixing).
- As $\mu \to 0$, the optimality gap approaches infinity while the upper bound of mixing time scales with $O(\log(n))$ (fast-mixing).
- The mixing time undergoes a transition around the threshold $\mu_{th} = \frac{1}{\Phi^O - \Phi^{\min}} \ln[(1+\frac{1}{B})(1+\frac{1}{N-B-1})]$. When $\mu \leq \mu_{th}$, the system is fast mixing; otherwise it is slow mixing.

5.3 Experimental Results

We use some simple experiments to show the convergence behavior of the combined algorithms, and show that while the analysis on the mixing time presents certain bounds on the performance, in practice the algorithm converges fast in most cases. The same parameters as those in Figure 3.3 are used for comparison, and the only addition is the topology selection part. In Figure 5.1 (a), we compare our results of soft-worst-neighbor-choking and no choking. The proposed soft-worst-neighbor-choking outperforms other methods and is able to achieve the theoretical lower bound of server load. In Figure 5.1(b), we compare the convergence time at different μ and $|U|$ (total number of users in the system). We can also see that in practice, the convergence time increases almost linearly with μ, and sublinear in the network size $|U|$. From Figure 5.1(a), we see that the algorithm approaches to very close to optimal value when $\mu = 10$, and even in this case, the convergence time is about a few hundred steps.

Chapter 6

System Design

In this chapter, we discuss the practical aspects of the system design. While earlier chapters build the theoretical foundations of the system, this chapter focuses on the actual system design and implementation. We first revisit our fractional storage concept and propose how to achieve the fluid limit in practice. Specifically, we propose the storage architecture and discuss in detail how to design network codes to achieve good performance. We also list other practical considerations including content placement update intervals, node neighbor set selection, and time-varying demand. While the theoretical framework cannot capture every detail, we give constructive solutions to deal with these considerations in practice. Finally, we summarize the system prototyping implementation, and present the algorithms of each system module. Based on the theoretical framework and practical guidelines, we have built a system prototype at Berkeley, which we implemented using Python and C++. We will present the simulation and experimental results in the next chapter.

6.1 Video Storage Design

In previous chapters, we assume that the constraint (3.7) $x_{r:=(h,u)} \leq W_{hm}\gamma_m$ still applies in the case of fractional storage, but how to achieve it in practice? We design the following video storage architecture and choose appropriate network codes to answer this question.

6.1.1 Uniform Storage

In VoD services, a video can be requested by any user watching at any arbitrary time points of that video. Therefore, we first break each video into non-overlapping chunks of k packets of equal size w. To enable fractional storage while making constraint (3.7) valid, we make the following design choice: a cache node h storing a fraction W_{hm} of video m store $W_{hm}k$ packets of each chunk of that video. An example is illustrated in Figure 6.1, where a video with size $\beta = 2$GB and rate $\gamma = 512$kbps is chopped into 4096 chunks, each having $k = 64$ packets of size 8KB each. A chunk contains 8 seconds of video stream on average. When a cache node h stores $W = 0.25$ of the video, it stores $Wk = 16$ packets of each chunk. Using this structure, the cache node is able to provide any user streaming any chunk of the video at a maximum rate of 128KB per 8 seconds, i.e, 128kbps, which is 25% of the streaming rate. Similarly, a cache node h can delivery up to a rate of $W_{hm}\gamma_m$ to a user watching video m if it stores W_{hm} fraction of video m.

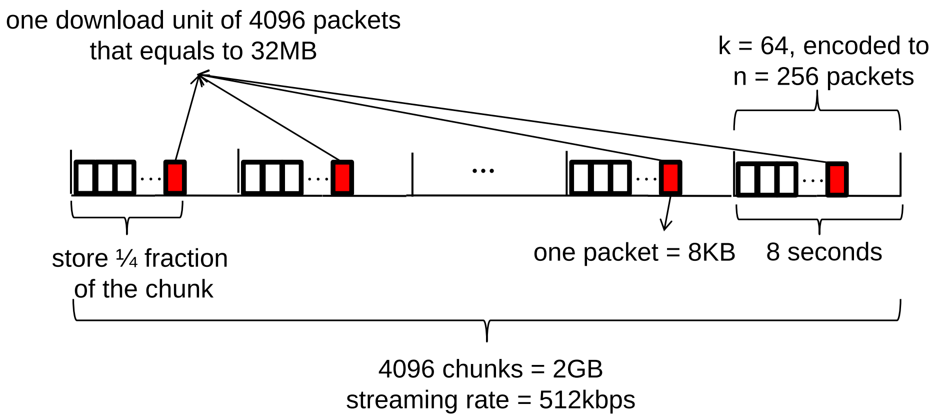

one download unit of 4096 packets that equals to 32MB
k = 64, encoded to n = 256 packets
...
store ¼ fraction of the chunk
one packet = 8KB
8 seconds
4096 chunks = 2GB
streaming rate = 512kbps

Figure 6.1: A cache node's storage of a video. The video has a size of 2GB and a rate of 512kbps. It is chopped into 4096 chunks, each having $k = 64$ packets of size $w = 8$KB. When the stored fraction is $W = 0.25$, the cache node is able to serve at a rate of $512 \times 0.25 = 128$ kbps to any user watching this video.

However, if the original packets are sprinkled among caches as is, the equality in constraint (3.7) may still be far from achievable. This is because there exists many duplicated packets in the network. A user who accesses k packets from a randomly chosen set of caches may not be able to obtain k unique packets. To ensure the equality in constraint (3.7) is achievable, the stored packets should ideally allow a user to recover a video chunk formed of k packets by downloading *any* k packets from caches.

In the next sections, we will elaborate on the choice of the practical codes for our VoD system. As we mentioned earlier, the codes play a central role in our problem formulation, and helps convert a combinatorially difficult problem to a convex one with a fluid model. We start by describing the general properties that such codes should satisfy and then propose a solution based on a family of codes called DRESS (Distributed Replication-based Simple Storage) codes [47, 19] that offer desirable tradeoffs among these properties.

6.1.2 Desirable Code Properties

We identify the following properties that codes should satisfy when used in our VoD systems:

1) MDS or Quasi-MDS property to approximate the fluid model. As we mentioned earlier, the code should ideally allow a user to recover a video chunk formed of k packets by downloading *any* k packets from caches[1]. This property is needed to make the fluid assumption valid. This corresponds to encoding the chunk using a Maximum Distance Separable (MDS) code, such as a Reed-Solomon (RS) code [52] or Quasi-MDS codes like Fountain codes [14].

2) Decentralized growth property. Our problem formulation in (2.1) precluded cache-to-cache

[1]The users may have to pay a little overhead, i.e., it may need to download a little more then k packets depending on the design parameters of the code.

communication. Since the formulation assumes a static scenario where users' demands do not change over time and cache nodes never leave the system, the cost of populating content to the caches from the backup server can be neglected. However, in the following scenarios, it may be more beneficial for the system that a cache downloads its content from other caches instead of downloading from the server:

1. Cache nodes have free resources to share with neighboring cache nodes. In this case, downloading from cache nodes is more cost-effective from downloading from the backup server.

2. The network conditions favor cache-to-cache download instead of the backup server pushing to the cache cloud. In this case, downloading from neighboring cache nodes is more efficient.

3. The upload bandwidth of the backup server is limited, and the number of its allowed connections is restrictive. In this case, the caches have no options but to download from other cache nodes.

We call such cache-to-cache communication "decentralized growth", i.e., the cache cloud needs to grow to a larger size in a decentralized manner. We would like to design codes such that decentralized growth is also efficient.
3) Security property to protect the system from malicious nodes. When a cache node or a user downloads a packet from the cache network, it needs to verify its integrity and detect any malicious nodes in order to prevent it from polluting the entire network. If new coded packets are generated at the caches, verifying their integrity pose severe security challenges to the system. Therefore, we require that all packets be created at centrally, e.g., at the backup server.
4) Small block length property to ensure a small start-up delay. The streaming nature of the VoD problem imposes stringent restrictions on the number of packets k that can be coded together. With a fixed packet size, the video player will not be able to decode and play the video after all k packets are downloaded. For example, for a video of rate 1Mbps with packet size equal to 10KB and 10 seconds start-up delay, the choice of k is equal to 128 packets.

6.1.3 DRESS Code Design

Different codes in the literature, such as random linear network codes [24], Fountain codes [14] or Regenerating codes [18, 51] provide different tradeoffs of the code properties discussed in this subsection. A class of distributed replication based simple storage (DRESS) codes were introduced in [47, 19] in the context of a distributed storage application, which we choose to use in our VoD system. For each chunk, DRESS code encodes the k data packets into n packets using an outer (n, k) (n to be chosen later) Reed-Solomon code at the backup server. We now address the question of how to choose the design parameters n, k and w with regard to the properties discussed above.

MDS versus Decentralized Growth

For the DRESS codes to satisfy the MDS or Quasi-MDS property, we need $n \to \infty$. When n approaches infinity, every packet of each video chunk is unique in the cache network, and it is enough for any user to download any k packets from the cache nodes. When n is finite however, a user may have to contact more nodes to retrieve all the unique packets due to packet duplicates.

Table 6.1 shows the average number of caches a user needs to contact to retrieve k unique packets under certain parameters. In this example, we assume each cache node stores exactly α packets of a chunk.

user	No. of storage units accessed	4	5	6	7	8	9	10
	average % of missing packets	20.5	7.2	1.2	0.1	0.007	0.0003	0
cache	No. of storage units accessed	10	15	20	25	30	35	40
	average % of missing packets	28.2	15.0	8.0	4.2	2.2	1.2	0.6

Table 6.1: The table shows the average performance for both user requests and decentralized growth, given a DRESS code that uses Reed-Solomon code as an outer code with parameters $k = 20, n = 40, \alpha = 5$.

As we can see, when $n = 40$ and every cache node stores $\alpha = 5$ packets, almost all the users can retrieve $k = 20$ unique packets (thus are able to decode the file) by accessing 6 to 7 caches, instead of 4 as implied by the fluid model. In this case, there is performance loss in the storage efficiency, i.e., the cache node might be able to deliver only much less than $W_{hm}\gamma_m$ even if he stores W_{hm}. As n goes to infinity, the number of caches a user needs to contact should approach to 4, and the fluid limit (and therefore constraint (3.7)) will be fully satisfied.

On the other hand, a larger n will hurt the performance of decentralized growth. Consider the following cache-to-cache communication strategy. When a cache node needs to download α packets from a chunk, it randomly chooses (without replacement) these α packets out of n packets. It then randomly contacts a number of other cache nodes, and requests the corresponding packets from them. This procedure amounts to the backup server providing the coded packets in the form of sets of uniformly random α packets, thus resulting in a seamless decentralized growth. Table 6.1 shows the average number of cache nodes it downloads from versus the percentage of packets missing. We can see that even with n equal to 40, a cache node already needs to contact 35 nodes to ensure more than 98% of the desired packets can be retrieved. This will become even worse when α becomes larger.

Using the coupon collection argument, one can compute the expected number of cache nodes X that a user has to contact to get k distinct coded packets:

$$E(X) \le \frac{k}{\alpha r}\log\frac{1}{1-r} \tag{6.1}$$

where $r = k/n$ is the rate of the outer MDS code.

Similarly, a cache node that attempts to download α new unique packets using the decentralized growth mechanism has to contact a random number Y of other caches, and the expected value of Y can be computed as:

$$\begin{aligned} E(Y) &= \frac{1}{\alpha}\left[\frac{n}{\alpha} + \frac{n}{\alpha-1} + \cdots + \frac{n}{1}\right] \\ &< \frac{k}{r\alpha}\left[\log\alpha + \frac{1}{2\alpha} + 0.58\right] \end{aligned} \tag{6.2}$$

where we have used $\sum_{n=1}^{m} 1/n < \log m + 0.58 + 1/2m$ from Harmonic series approximation.

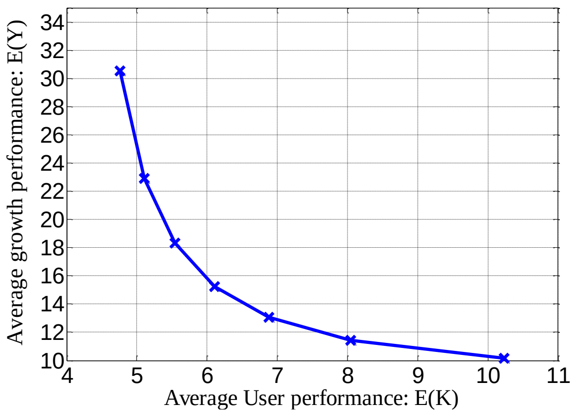


Figure 6.2: The tradeoff between the expected number $E(X)$ of cache nodes accessed by a user to recover the file and the expected number $E(Y)$ of cache nodes accessed by a cache node for the decentralized growth, parameterized by the rate of the code $r = k/n$. The DRESS code with $k = 20$ and $\alpha = 5$ is used in this example.

The details of the characterization can be found in [47]. From equations (6.1) and (6.2), we see that for any fixed value of k and α, there is a tradeoff, parameterized by $r = k/n$, between the users performance and the decentralized growth performance of a DRESS code. This is illustrated in Figure 6.2.

It is worth noting that by using DRESS code, all of the coded packets are generated at the oracle server and can be easily authenticated using cryptographic hashes. Therefore the security property can be easily satisfied.

Small Block Length

Since the packets of a chunk are coded, the users will have to wait for all of the packets to be downloaded in order to play the corresponding stream. Therefore, the start-up delay increases as the chunk size increases. We choose kw/γ_m to be less than 10 seconds, which is a reasonable start-up delay. Table 6.2 shows a qualitative summary of how each parameter of n, k and w can affect the design properties.

parameter	decrease	increase
n	worse fluid-limit approx.	less efficient growth; coding complexity increases; more server storage
k	worse fluid-limit approx.	coding complexity increases; start-up delay increases
w	overhead increases	start-up delay increases

Table 6.2: A qualitative summary of how each coding parameter will affect the design properties.

6.1.4 Experimental Results

We now conduct some experiments to (1) demonstrate that the proposed coding scheme outperforms traditional non-coding schemes (which are often implemented in the literature and practical systems) and converges closely to the solution of the fluid-level optimization problem (2.1); and (2) guide the choice of a set of coding parameters (n, k, w). The same simulation setting in Chapter 5 is used. However, we replace the fluid drops with actual packets and require that each user receives enough unique packets instead of just enough drops to play the video.

Table 6.3 shows the percentage of non-cache traffic using different coding schemes. We can see that the traditional non-coding scheme (or replication scheme) is far from optimal. When $k = 20$ as we increase n from 40 to 80, the system performance improves and approaches to the fluid limit. Referring to Figure 6.2 which shows the cache-to-cache communication, it proves reasonable to choose n between 40 and 80 and $k = 20$ in our system design.

	Fluid	$(n, k) = (80, 20)$	$(n, k) = (40, 20)$	$(n, k) = (20, 20)$
non-cache traffic $\%$	7.15%	8.17%	9.89%	17.83%

Table 6.3: Percentage of non-cache traffic of different coding schemes compared to the optimal fluid limit.

6.2 Practical Considerations

6.2.1 Time Varying Demand

We observe that the aggregate video demand and its distribution can be time-varying due to many reasons. First, the total number of users in the system may be quite different during different times of a day and during different days of a week. In many VoD systems, the number of users peak during weekends and in the evenings. Second, users' statistical demand distribution can also change over time. For example, the popularity of a number of videos can take a peak for a number of days during its debut and gradually declines as time goes by. While the theory for these situations can be quite challenging, we evaluate the performance of our proposed algorithms in Chapter 7.

6.2.2 Placement Update Frequency

Another consideration is the frequency of performing the content placement algorithm. While updating content placement more frequently allows the system to adapt to varying demands more gracefully, each update incurs both computational and content migrating overhead. We let the storage update only every D seconds, and evaluate the trade-off between the choice of D and system performance in Chapter 7.

6.2.3 Neighbor Set Selection

We have made the assumption that a cache or user node v is limited to connecting a fixed neighbor set Q_v. The choice of such a neighbor set often depends on many system considerations. One way is to base on geographical proximity, which is used in many CDN scenarios. Another example is to choose a neighbor set whose video supply matches the demand the most[2]. Other choices can depend on load balancing or ISP-friendliness considerations. In addition, the list can be periodically updated as more information is obtained. The analysis of how to choose such a neighbor set is beyond the scope of our discussions. In this paper, we simply assume that a certain strategy is used to obtain Q_v. However, any more sophisticated mechanisms would only boost our system performance.

6.2.4 Link Price Update

The parameters x_r, λ_r, W_{hm} and ω_h in solution (3.11)-(3.14) can be maintained and updated locally at each cache. However, the variables c_l (link capacity) and link congestion price θ_l may not be readily available at the cache node level. We make an important observation that the update equation $\dot{\theta}_l = [\eta_l(\sum_{r:l\in r} x_r - c_l)]^+_{\theta_l}$ has a physical interpretation. Integrating both sides of the equation over time, we have:

$$\theta_l(t) = \int [\eta_l(\sum_{r:l\in r} x_r - c_l)]^+_{\theta_l} dt$$

The right hand side of the equation is the aggregate difference between demand and supply, which is proportional to the link queue length. In practice, the queue length is approximately proportional to the transmission delay. Therefore $q_r(t) = \sum_{l:l\in r} \theta_l(t)$ can be treated as the aggregate route transmission delay at time t, which can be measured by the round trip time from the cache node to the user node.

6.3 System Architecture

Based on the above findings, we have architected and built a prototype VoD system at Berkeley and CUHK. The DRESS code part is written in C++ and the rest of the system is written in Python. The system consists of a backup server, a scheduler, a set of distributed caches and users interconnected via a LAN network. The system is illustrated in Figure 6.3. The users perform packets decoding and video playing in realtime. All the packets are transmitted through the network using TCP. Each user has a GUI to select videos to watch. A monitoring system is implemented to demonstrate the system performance in real time.

Based on our theoretical design and analysis, we summarize the packet-level algorithms of each unit as follows.

6.3.1 Central Server

A central server contains all the video content, and is connected to every cache node and user node. The server is responsible for the following tasks:

- It stores all the available videos using the DRESS code design: it breaks a video into a number of chunks each having k packets that are encoded into n packets. These n packets will become the only packets of that chunk which are circulated in the network. No new packets are

[2]For example, the system can keep track of which caches store which videos to make such decisions

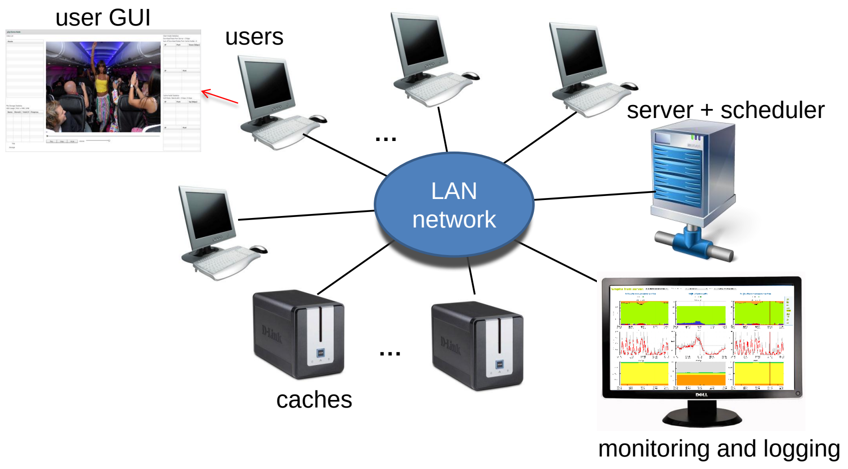


Figure 6.3: Architecture of the VoD system that we have built at Berkeley.

generated from the caches or users. We choose $n = 80$ and $k = 20$. The size w of each packet is chosen such that the length of an encoded chunk is roughly $kw/\gamma = 10$ seconds.

- The server operates in a "pull mode", i.e., it uploads packets to a user or a cache node whenever it is requested to do so, acting as the "life-line" to supplement any streaming deficit between the cache clouds and the users.
- It uploads videos to the cache nodes when they cannot get them from each other.

6.3.2 Scheduler

A scheduler (tracker) is implemented that keeps track of all the participating nodes including the backup server, the users and the caches. When a new node joins the system, it first communicates with the scheduler to register its identity, e.g., IP address, and obtains a list of neighbors.

6.3.3 Cache Protocol

The cache nodes store the videos in a similar way as the backup server, except that they only store up to k packets out of a total of n packets. The cache nodes update their storage and bandwidth allocation according to the previously proposed algorithm. The detailed protocol of cache h is given in Algorithm 1.

6.3.4 User Protocol

Once a user finds his neighborhood from the tracker, it initiates connections to the maximum allowed number of cache neighbors randomly chosen from the neighbor list, and starts to

Algorithm 1 Cache Protocol

```
1: Set $t_1 = 30$ and $t_2 = D$ iterate:
2: Retrieve from each user node $u$ their marginal utility $V_{x_r}(z_u)$ and measure the round-trip delay $q_r$ for each route $r$.
3: Update the rate $x_r$ for each route using $\Delta x_r = [\delta_r(V_{x_r}(z_u) - \lambda_r - q_r)]^+_{x_r}$.
4: Update the availability price for each route $r$ by $\Delta\lambda_r = [\kappa_r(x_r - W_{hm}\gamma_m)]^+_{\lambda_r}$.
5: if $t_2 = 0$ then
6:     Change the storage fraction $W_{hm}$ for each video by $\Delta W_{hm} = [\iota_{hm}(\Lambda_{hm} - \beta_m\omega_h)]^{[0,1]}_{W_{hm}}$. If $\Delta W_{hm}$ is negative, it removes video $m$ by $W_{hm}$ amount; otherwise, it chooses randomly $\lfloor nW_{hm} \rfloor$ number of packets exclusive of what it already has of every chunk of video $m$, and requests to download from neighboring caches (or from the backup server if not successful). Finally, update the storage price by $\Delta\omega_h = [\nu_h(\sum_{m \in M} W_{hm}\beta_m - s_h)]^+_{\omega_h}$. Reset $t_2 = D$.
7: end if
8: For each neighbor user $u$, allocate number of packets proportional to $x_{hu}$.
9: if $t_1 = 0$ then
10:     Randomly choose and connect to a new neighbor from the neighbor list. Then drop a connected user $u$ with probability $\frac{\exp(-\frac{1}{2}\mu\bar{x}_{\bar{g}\backslash g'})}{1+\sum_{g'' \in \mathcal{A}_{h,\bar{g}}} \exp(-\frac{1}{2}\mu\bar{x}_{\bar{g}\backslash g''})}$. Reset $t = 30$.
11: end if
12: $t_1 \leftarrow t_1 - 1, t_2 \leftarrow t_2 - 1$.
```

Algorithm 2 User Protocol

```
1: Set $t = 30$ and iterate:
2: Compute the average received rate from each connected cache node since the last update. Derive its marginal utility value and send it to each connected cache node.
3: if buffer length is less than an emergency threshold then
4:     Download from the server all the missing packets to fill up the buffer.
5: end if
6: if has received $k$ packets of the next chunk then
7:     Decode the chunk and flush the data to the media player.
8: end if
9: if $t = 0$ then
10:     Randomly choose and connect to a new neighbor from the neighbor list. Then drop a connected cache node $h$ with probability $\frac{\exp(-\frac{1}{2}\mu\bar{x}_{\bar{g}\backslash g'})}{1+\sum_{g'' \in \mathcal{A}_{h,\bar{g}}} \exp(-\frac{1}{2}\mu\bar{x}_{\bar{g}\backslash g''})}$. Reset $t = 30$.
11: end if
12: $t \leftarrow t - 1$.
```

request video packets of some video at his choice. To maintain smooth playback, it maintains a buffer of, e.g., 30 seconds.

A user requests packets from his connected cache neighbors by sending to each of them his marginal utility value, and then waits for the packets to arrive. The number of packets he receives during a certain time window is determined by the rate allocation algorithm which the cache nodes run. The user decodes a chunk whenever he receives any k unique packets out of n coded packets of that chunk. When a user's playback buffer is running out before he can obtain all the packets needed to decode, it requests to download from the backup server to fill up the buffer pipe immediately. The users also implement a "soft-worst-neighbor-choking" algorithm to perform topology graph updates. The detailed user protocol is given in Algorithm 2.

The deployment of such a practical system is quite simple. The users and caches are "plug-and-play", i.e., they run distributed algorithms that only require local statistics.

Chapter 7

Experimentation

This chapter explores extensive simulations to verify the feasibility and efficiency of our system design. In early chapters, we have seen the convergence result of the content placement and link rate allocation scheme, and that of the topology selection algorithm. We have also shown that with appropriate choice of codes, the system result approaches that of the fluid model.

In this chapter, we we use realworld traces and show that the the system works in real-world dynamic cases. We demonstrate that the algorithms implicitly learn the video demand, and yield high bandwidth utilization. We show extensive results to test the algorithm's scalability and robustness to changes in user dynamics and demand patterns. We show that our solution high utilization of cache nodes storage and bandwidth resources, and automatically learns and caches the video according to the demand patterns. We observe that there exists a complex interplay between disk space, network bandwidth and node degree bound. We also present guidelines to important practical design choices including caching update intervals, demand prediction and provisioning. We compare the simulation results with those from the actual prototype system we built at Berkeley to verify the practicality of our simulation results.

7.1 Simulation Results

In this section, we evaluate the overall system performance using realworld traces crawled from YouTube [15]. We compare the performance of our scheme with least recently used (LRU) and least frequently used (LFU) alternatives which are commonly used in practice. We also compare with the MIP approach introduced in [9] because their problem definition and setup is the closest to that of ours.

7.1.1 Setup

We crawled 2000 videos from YouTube with 3 days of traffic. The lengths of the videos vary from 5 mins to 20 mins. For simplicity, we map these videos to have 10 mins each with a streaming rate of 2Mbps. In the simulations, 50 cache nodes are scattered randomly in all locations of the network. We focus on the scenario where the cache nodes have equal storage capacities and node degree bound, but also present results where resources are heterogeneous. We model the

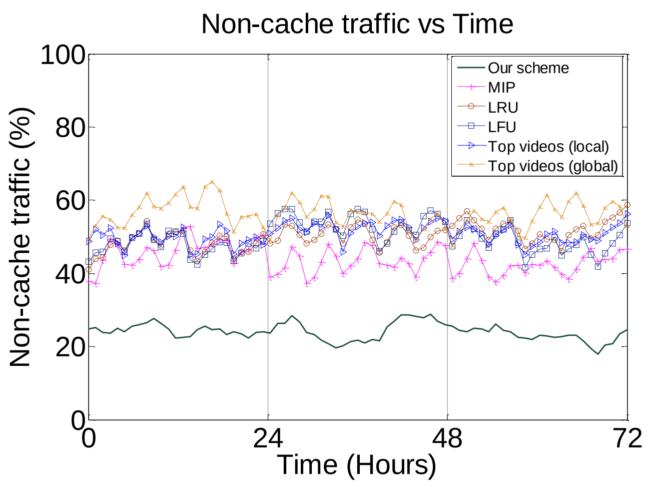


Figure 7.1: Performance comparison of our scheme with LRU, LFU and MIP. The figure shows the non-cache traffic of different schemes and the total demand for a period of 3 days.

network link capacities to follow a bi-module distribution. The actual values of all the parameters are varied to understand the trade-off between storage capacity, link bandwidth and node degree.

In most of our experiments, we perform caching updates in (3.13) and (3.14) everyday for only two hours during peak time between 6pm to 8pm unless stated otherwise. To make comparisons fair, the same update intervals are also used for the comparing schemes. We compare the fraction of the total non-cache traffic for the different schemes, assuming the traffic requests that cannot be satisfied from the cache nodes are delivered by an Oracle database that stores all videos and has infinite degree bound and bandwidth.

7.1.2 Performance

In this experiment, we compare our results with LRU, LFU, top videos (local), top videos (global) and MIP using the trace we crawled. In the peak hours starting from 6pm everyday, the maximum number users is 40000. The following parameters are chosen: the total storage for caches is 25% of the entire catalog size; the total bandwidth are randomly distributed among the caches and the total bandwidth can just cover the total demand of streaming rates. The node degree bound on each cache is 12000. The methods and parameters used in the comparing schemes are:

- MIP: our algorithm is first run, and then on each cache the entire video with the largest converged fraction of storage is stored, followed by the video with the second largest converged value, and so on and so forth, until the storage capacity is reached.

- LRU: at the beginning of each day, every cache collects information from user demands for a short period of one hour of the most three popular videos. The cache then replaces their locally least recently used videos by the most three popular videos.

- LFU: at the beginning of each day, every cache collects information from user demands for a short period of one hour of the most three popular videos. The cache then replaces their locally least frequently used videos by the most three popular videos.

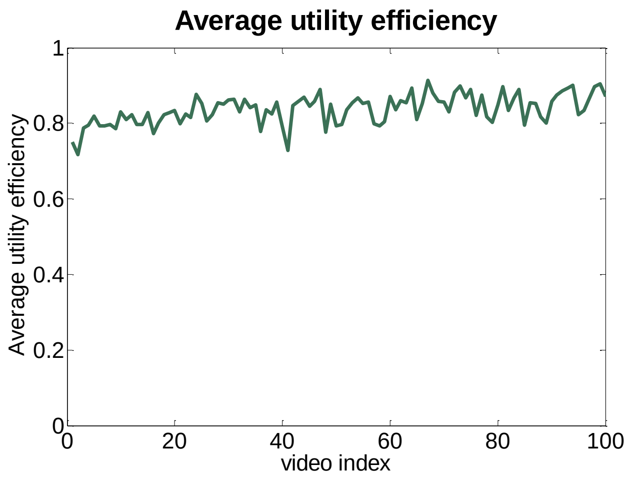


Figure 7.2: (Utility efficiency for different videos. The utility efficiency for video m on each cache node h is defined as $\frac{\sum_{u \in U_{m,h}} x_{hm}}{W_{hm}\gamma_m |U_{m,h}|}$, which represents how useful any stored fraction of a video is averaged among the cache nodes.

- Top videos (local): at the beginning of each day, every cache stores the most globally popular 5 videos of that day.
- Top videos (global): at the beginning of each day, every cache stores the most locally popular 5 videos of that day as measured by its connected user neighbors.

Figure 7.1 shows the non-cache traffic for each method and the total demand versus time. We observe that for the same amount of system resources, our scheme allows the maximum support of the caches, thanks to our theoretical guarantee of optimal performance. Specifically during peak hours at 7pm, the corresponding non-cache traffic for our scheme was 24.1% while it is 44.0%, 50.2%, 50.0%, 50.9% and 56.4% for MIP, LRU, LFU, top videos (local) and top videos (global) . Storing the globally popular videos, as is done in most practical cases, performs the worst. This is because video demand distribution is so heavily tailed that providing caching service of the vast number of unpopular videos becomes unavoidable. MIP performs the most closely to our scheme but is still far from optimal. This corroborates our previous observations in the toy examples that fractional storage provides much flexibility to the caching service. The simulations show strong evidence that fractional storage performs much better in practice.

To understand better the performance of our algorithm, we show in Figure 7.2 the average resource utility efficiency for each video. The utility efficiency for video m on each cache node h is defined as

$$\frac{\sum_{u \in U_{m,h}} x_{hm}}{W_{hm}\gamma_m |U_{m,h}|}$$

which is the ratio between the total upload rate to users watching video m and the total available rate given the storage of video m. It shows how much useful any stored fraction of a video is on each cache. The efficiency factor for each video is averaged across all the caches which store that video.

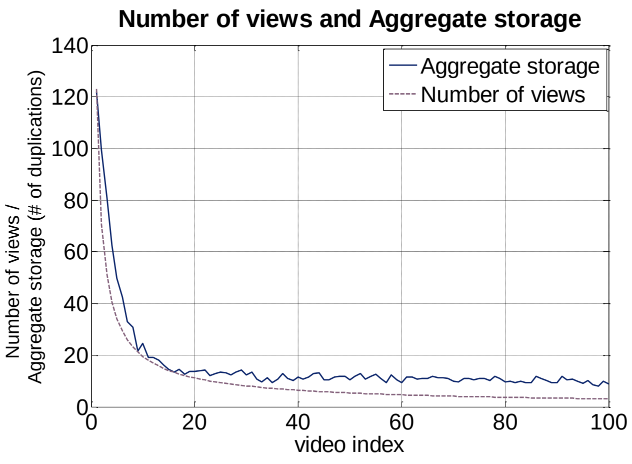


Figure 7.3: Total storage of videos in the cache nodes and the actual demand distributions shown in log scale.

Figure 7.2 shows that most of the efficiency factors are greater than 80%, and the efficiency factors are quite uniform across the videos. This says that most video storage allocations are efficient.

In Figure 7.3, we break up the total storage by the videos and compare it with the demand distribution. We can see that although running distributively, the algorithm is able to implicitly learn the demand distribution and store the amount of videos in a distribution similar to the demand patterns. This is a useful property of the algorithm, because separately estimating the demand in practice can induce large overhead. Our scheme is demand-agnostic, i.e., it does not require any prior knowledge of the demand to perform optimally by automatically learning the demand distribution automatically in a distributed way.

We also check how the load is balanced across the caches. Because of sufficient resources, caches do not have to fully utilize the resources but have to just satisfy the demand. In this case, how the load is distributed? Are only few powerful caches chosen or all caches are equally used? The bandwidth utilization (sum of upload rates divided by upload capacity) is measured for each cache and empirical cumulative distribution function is plotted with the legend 'all caches' in Figure 7.4. The CDF shows that about 70% of caches are utilizing more than 90% of upload bandwidth and half of them are utilizing bandwidth almost fully, which indicates the algorithm evenly distributes the supply to users. In addition to the CDF across all caches, we check two CDFs which are separately attained from 'high bandwidth cache group' and 'low bandwidth cache group'. Interestingly, caches with lower bandwidth are more utilized than higher bandwidth which is counter-intuitive at first glance. However, it can be interpreted as following: under same storage, caches with lower upload bandwidth can more easily find the optimal resource allocation.

7.1.3 Interplay between Storage, Bandwidth and Node Degree

To understand further the tradeoff between various resources including storage, bandwidth and degree bound, we compare the percentage of average non-cache traffic at peak hours from 6pm to 8pm by fixing one dimension of the resources and vary the other two dimensions. In Figure 7.5(a),

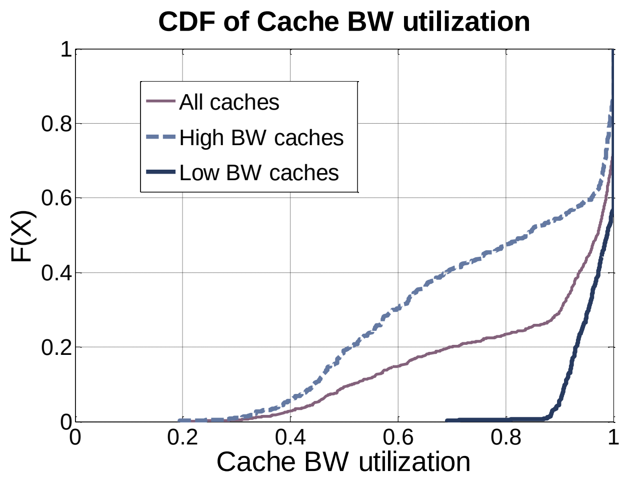
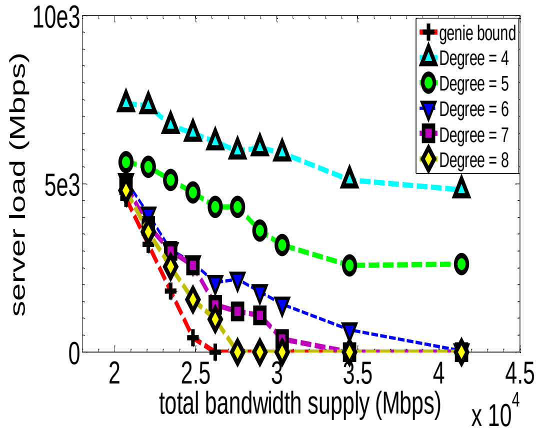


Figure 7.4: CDF of cache bandwidth utilization.

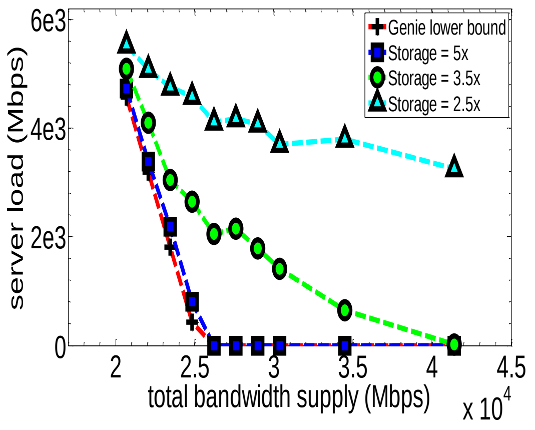
we fix the total storage of all the cache nodes to be 3.5 times of the size of the entire video catalog, and vary the average node degree bound. The figure shows the performance versus total average bandwidth for each degree bound. We can see that when the degree bound resource is too scarce, e.g., less than or equal to 5, the non-cache traffic is non-negligible regardless of the bandwidth sources given. When degree resource is greater than 5 however, two difference cases occur: (1) when the system is in either extreme bandwidth deficit or extreme bandwidth surplus mode, the non-cache traffic easily achieve the genie lower bound regardless of the storage capacities; and (2) when the bandwidth resource is in between the two extremes, the non-cache traffic consistently drops with the increase of the degree bound. Although we do not have any theoretical proofs, we believe this should suggest that there exists some an intrinsic minimal degree bound for the system to perform optimally.

In Figure 7.5(b), similar results are shown when the degree is fixed to be 6, and the non-cache traffic versus bandwidth curves are shown for different the storage capacities. When the total storage of all the cache nodes is 5 times of the size of the catalog, the non-cache traffic easily achieves the genie lower bound but if the total storage is 2.5 times of the size of the catalog, the non-cache traffic is high with sufficient bandwidth resources.

To see the interplay among all three resources, we show a contour plot in Figure 7.6. The x-axis represents the degree bound and the y-axis presents the total storage capacities as a fraction of the catalog size. For each contour, we set the total bandwidth supply equal to the total required streaming rates. On the contours shows the non-cache traffic in percentage, e.g, 10%, 20% and 40% etc. One can see that there appears to be a feasibility area near the diagonal line. Outside that area, one needs exponentially increasing resources of one kind to compensate a small decrease of that of another to achieve the same performance. This helps understand the fundamental minimal resources needed to deploy satisfy users' need when designing a VoD system.



(a)



(b)

Figure 7.5: Interplay between storage, bandwidth and node degree bound. (a) Total storage is fixed at 3.5 video's catalog. The figure shows the average server load versus total bandwidth supply for different degree bound. (b) Degree bound is fixed at 6. Server load versus total bandwidth supply is shown for different average storage capacity.

7.1.4 Scalability and Robustness

In this experiment, we keep the user arrival patterns but vary the video catalog size and investigate how the system responds. We maintain the popularity pattern of the trace, which exhibits a Zipf's distribution, but scale the distribution such that the total popularity probability is one. In Figure 7.7(a), we plot the average non-cache traffic during peak hours from 6pm to 8pm for different catalog sizes. It can be seen that although the catalog sizes vary from 2000 to 10000, the non-cache traffic only increases slightly. The resource allocation algorithm automatically adjusts the caching strategies to maximize the contribution of the caches. The robustness of the performance to the changes in the catalog size is especially useful when a sudden flash crowd of new videos appear and the system does not have ample time to call for more resources. Over time when the catalog size keeps increasing, we project that one needs to increase the total storage on the cache only slightly in order to keep the same amount of non-cache traffic.

To study how the system handles different user dynamics, we vary the user arrival rates and observe how the system behaves. To keep the average number of requests roughly the same, we apply Little's law by tweaking the video length and the arrival rate such that their product is kept constant. Storage of each cache is also adjusted to keep the system's storage constant with the consideration of the change of movie size. Figure 7.7(b) shows the system handles different user dynamics very well. The average non-cache traffic during peak hours only increased slightly when the user arrival rate increases.

We also change the video demand patterns and investigate how the system adapts. In Figure 7.8(a) and (b), we change the demand distribution in the middle of the peak hours and study how the system adapts to it. In Figure 7.8(a), the demand is changed to a uniform distribution at the 1000th step. In Figure 7.8(b), the change is more dramatic, i.e., the entire distribution is flipped where the most popular video becomes the least popular video and vice versa. In Figure 7.8(c), a

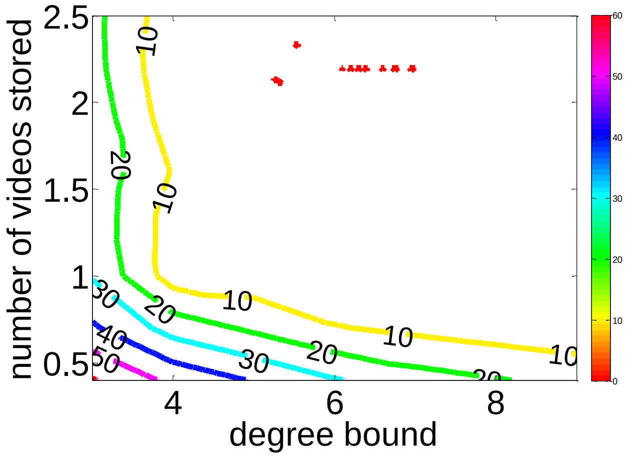


Figure 7.6: Contour plots of optimality gaps with different degree bound and average storage capacity. Total bandwidth supply is equal to the total demand in each point in the curves.

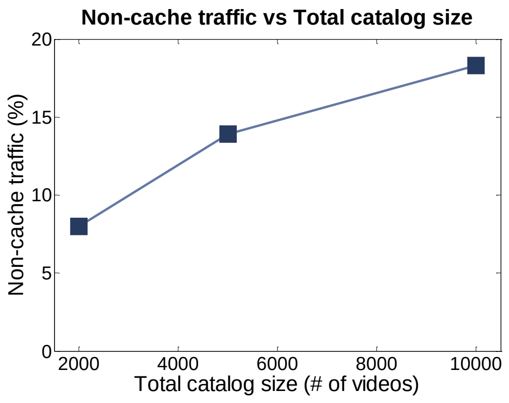
more realistic demand change is simulated, i.e., new popular movies are constantly added and unpopular movies are fading slowly. In order to simulate this demand change pattern, we inject 10% of new most popular movies and remove 10% of least popular movies. Other 90% old movies become slightly less popular than before. In all of three cases, the server load experiences a negligible jump at the time of the change, but quickly goes back to normal. In practice, we expect that the changes in demand to be much slower and that the system can easily adapt to such changes.

7.1.5 Content Placement Frequency

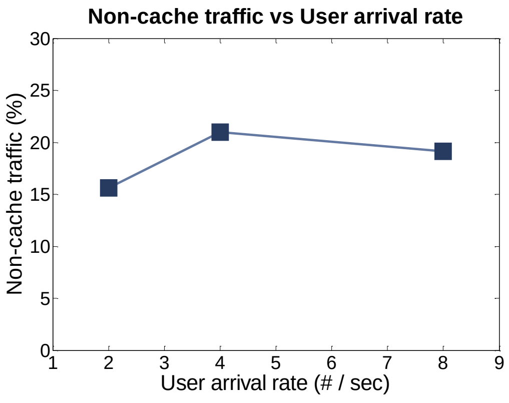
Running the content placement algorithm often allows us to handle demand changes gracefully. However, each iteration incurs migration overheads. To update the content on the cache network, the content needs to be downloaded to the cache from either neighboring cache nodes or an Oracle database. We experimented with different content placement frequencies to see how they affect performance. Specifically, we make the running interval to be 1 day, 2 days, 3 days and 6 days. Table 7.1 shows both the server load and the migration overheads averaged over a week. It can be seen that running the content placement algorithm too often can also hurt the total system performance. Note that updating every 2 days does not increase non-cache traffic but slightly reduces migration traffic. However, finding the optimal update period with different settings can be easily found by similar argument with different parameters. We are developing a theory which try to capture the effect of update period.

Table 7.1: Update period and total traffic

Update period (days)	1	2	3	6
Non-cache traffic (%)	14.9	14.9	34.6	50.2
Migration traffic (%)	5.7	5.3	2.0	0.9
Overall traffic (%)	20.6	20.2	36.6	51.1

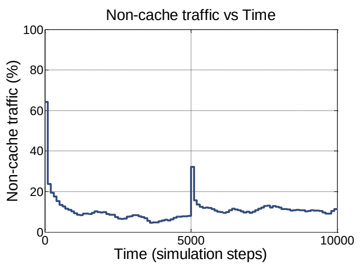


(a)

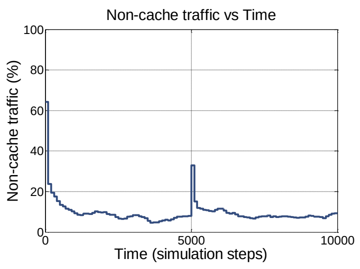


(b)

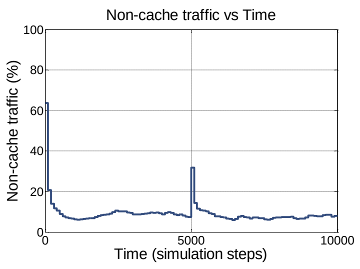
Figure 7.7: Average non-cache traffic during peak hours between 6pm and 8pm versus (a) varying catalog size; and (b) varying user dynamics.



(a)



(b)



(c)

Figure 7.8: System's responsiveness to change of demand patterns. (a) Demand is changed abruptly to a uniform distribution in the middle of the simulation. (b) Demand is flipped in the middle of the simulation. (c) New popular videos are added unpopular movies are removed everyday.

7.1.6 Demand Prediction

We investigate what the system does if it knows the demand distribution *a priori*. Specifically, we run a pre-allocation step, by scattering virtual users into the system that follow the video demand distribution, run our algorithms of content placement, link rate allocation and topology selection until they converge. During the pre-allocation period, all users are fixed, i.e., they do not leave the system during this period. We then fix the converged video storage at the cache nodes, feed the system with the actual users from realworld traces and start the simulation. Only the link rate allocation and topology selection algorithms are running since then.

Figure 7.9 shows how system performs versus simulation time. It is seen that the server load (or non-cache traffic) remains low at all times. This indicates that the pre-allocation of content is very efficient and that the link rate allocation and topology selection algorithms converge fast in this case. This provides a useful practice when building realworld systems. In this case, if the VoD service has a certain level of prediction of the demand, it can deploy a pre-allocation scheme by running a shadow version of the algorithm to determine the storage amount of the caches, and

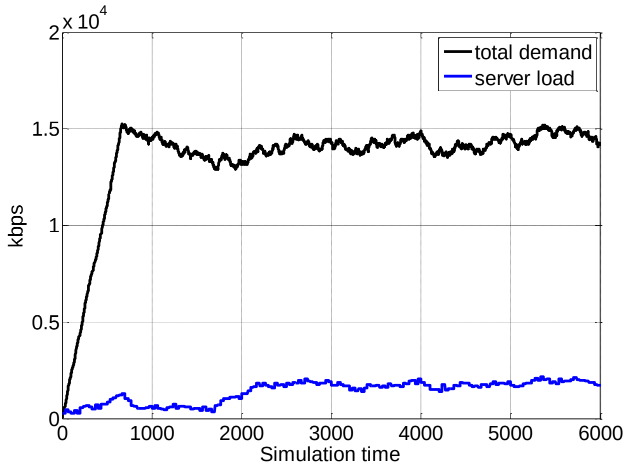


Figure 7.9: System performance with pre-allocation of content placement according a knowing demand *a priori*.

then fix the storage allocation. This will help avoid the necessity of running the content placement algorithm while the demand is happening to reduce the *hit* paid by the convergence period.

7.2 Experimental Results of Prototype System

To corroborate the design choice and the simulation results, we have built a system prototype at Berkeley, described in the last section of Chapter 6. The system consists of 200 users and 100 caches (mimic of a P2P scenario). The video catalog consists of 20 videos with a flat streaming rate of 256kbps and a flat duration of 40 minutes. The video demand is Zipf's distributed, i.e, they choose video m with probability proportionally to $1/m^s$ with $s = 0.8$. Each cache can store a single video, resulting 5 times of video catalog's worth in total. The bandwidth capacity follows a uniform distribution of 256kbps, 512kbps and 768kbps. Although it is a relatively small-scale system, it implements the full cycle of our algorithms plus video coding and video playing. A user GUI is implemented to monitor the performance of each user and cache node, tracking their bandwidth utilization, connected neighbor set and storage allocations. We collected realtime trace results from testing the system, and compared them with the simulation results under the same setting. Figure 7.10 shows the server load of the simulator versus time. It converges quickly to a low value, and shows comparable performance with the simulations.

During the entire experiments, we also closely monitor the video streaming experience of all 200 users. In Figure 7.11(a), we show the distribution of start-up delays of the users. All the users have a start-up delay of less than 6 seconds, and most of them are less than 3 seconds. This shows the practicality of the network codes and the content delivery algorithm. Although each PC is running 20 instances of users, the network codes decoding and video decoding did not take a hit on the performance. In fact, most of the decoding was done in less than a seconds as we observe empirically. From Figure 7.11(b), we see that all the users have constant available buffer content of more than 10 seconds, which means 0% of users experienced any jittering and the video playback is

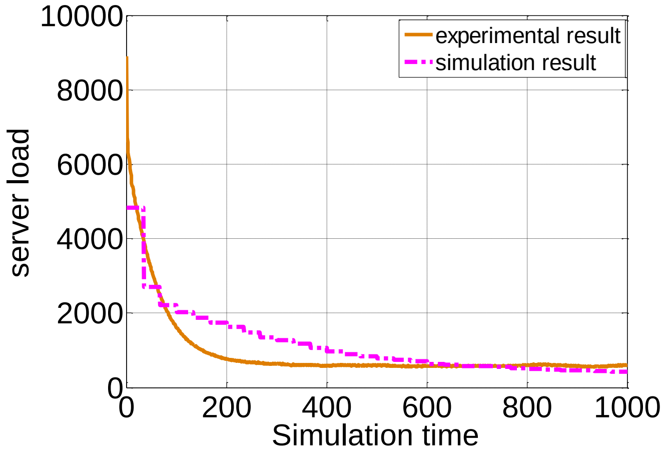
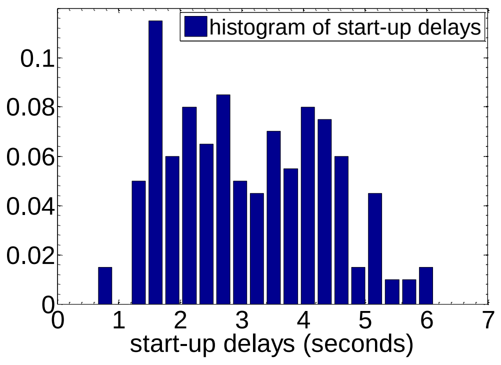
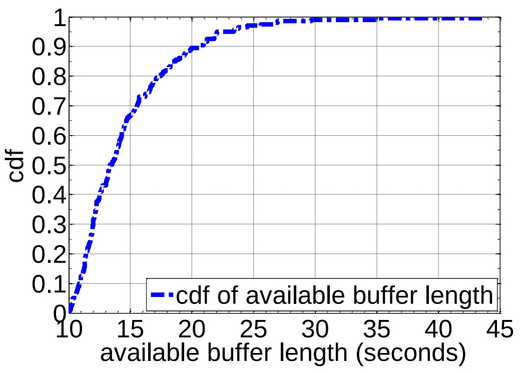


Figure 7.10: Comparing results of prototype system and those from simulations.

smooth during the entire video session. The results demonstrate the practical feasibility of the joint design of network codes, fractional storage and algorithm design.



(a)



(b)

Figure 7.11: QoS of the VoD system. (a) Distribution of the start-up delays. (b) Cumulative distribution of the available video content in the buffer in seconds.

Chapter 8

Conclusions

8.1 Summary of Results

In this dissertation, we propose the design and architecture of an optimized video-on-demand system. We formulate the design problem by jointly optimizing over practical constraints of storage capacity, network link capacity and node degree bound. While the overall optimization problem is NP-hard, we break it into two parts, i.e., a content placement and link rate allocation problem and a topology graph selection problem.

To tackle the difficulty in content placement problem, we design a fractional storage scheme and use appropriate network codes to transform the problem into a tractable and convex one. We observe the interesting fact that the fractional storage scheme outperforms the holistic storage scheme, and it yields a fully distributed algorithm which is easy to implement in practice. The proposed algorithm implicitly learns the video demand distribution, and yields high utilization efficiency of network resources. To tackle the difficulty of topology selection problem, we use an entropy approximation scheme, which transforms the problem into a Markov chain design problem. We develop an easy distributed algorithm that we show achieve close to optimal solution.

To corroborate our theoretical results, we present detailed discussions on practical system design, which takes into consideration of video storage, algorithm implementation and other practical issues including storage update intervals, user dynamics and etc. We also build a prototype system at Berkeley to demonstrate the feasibility and efficiency of our design choice. We presented detailed simulation results that cover a wide range of situations to test the system's responsiveness to user dynamics, demand changes and video catalog size. We show that our proposed scheme maximizes the video traffic supported by cache nodes, which efficiently utilize their system resources to support large-scale demand. We observe the the complex interplay between disk space, network bandwidth and node degree bound, whose further theoretical study is of profound interest. We also present guidelines to critical practical design choices including caching update intervals, demand prediction and provisioning.

8.2 Future Work

The work in this dissertation suggests several avenues of further study. We itemize some of the directions as follows.

- One possible future direction is to alter the proposed framework to study the problem of resource provisioning of video-on-demand services. For example, one question is how much storage and bandwidth resources is needed to deliver a certain video catalog?
- Another dimension is to extend the theoretical framework into other video applications including realtime streaming, video conferencing, video gaming and etc. Many of the proposed ideas may be extendible such as the link rate allocation algorithm, the topology selection algorithm and the network coding schemes.
- Another possible future work is to study the benefits of demand predictions. The theoretical framework proposed in this paper is prediction-agnostic. However, it is unclear how demand predictions, which are often in practice accessible to some certain extent, will help system performance including convergence time, better load balancing and etc. The prediction may also help the system handle better of flash crowd, which comes and goes quickly and is difficult to be captured by a static model.
- On the system side, it is worth pursuing system development at a larger scale. For example, Amazon S3 servers can be used to deploy the video-on-demand service. The benefit of experimenting large-scale tests of the prototype system is that it can capture many practical aspects not included in the prototype system, including real network effects such as bandwidth fluctuation and transmission delay, real cache server performance characteristics such as disk read/write speed and system failure, and other issues including control plane overhead, error handling and etc. On the other hand, different scenarios can also be tested including wireless video streaming, peer-to-peer video streaming and etc. A full scale test of the system helps understand both the potential and limitations of the theoretical model and design schemes.

Bibliography

[1] http://www.pptv.com/.

[2] Anonymize to align with the double-blind policy.

[3] Cisco Visual Networking Index: Forecast and Methodology, 2010-2015. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf.

[4] V. Aggarwal, O. Akonjang, and A. Feldmann. Improving user and isp experience through isp-aided p2p locality. In *proc.of IEEE INFOCOM*, 2008.

[5] R. Ahlswede, N. Cai, S. Li, and R. Yeung. Network information flow. *Information Theory, IEEE Transactions on*, 46(4):1204–1216, 2000.

[6] J. Almeida, D. Eager, M. Vernon, and S. Wright. Minimizing delivery cost in scalable streaming content distribution systems. *IEEE Transactions on Multimedia*, 6(2):356–365, 2004.

[7] N. Andrade, M. Mowbray, A. Lima, G. Wagner, and M. Ripeanu. Influences on cooperation in bittorrent communities. In *Proc. of ACM SIGCOMM workshop on Economics of peer-to-peer systems*, 2005.

[8] S. Annapureddy, C. Gkantsidis, P. Rodriguez, and L. Massoulie. Providing video-on-demand using peer-to-peer networks. In *Proc. of IPTV workshop*, 2006.

[9] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. Ramakrishnan. Optimal content placement for a large-scale vod system. In *Proceedings of the 6th International COnference*, page 4. ACM, 2010.

[10] A. Bellissimo, B. Levine, and P. Shenoy. Exploring the use of bittorrent as the basis for a large trace repository. *University of Massachusetts Technical Report*, pages 04–41, 2004.

[11] S. Borst, V. Gupta, and A. Walid. Distributed caching algorithms for content distribution networks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[12] Y. Boufkhad, F. Mathieu, F. de Montgolfier, D. Perino, and L. Viennot. Achievable catalog size in peer-to-peer video-on-demand systems. In *Proc. of IPTPS*, 2008.

[13] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[14] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *ACM SIGCOMM Computer Communication Review*, volume 28, pages 56–67. ACM, 1998.

[15] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 1–14. ACM, 2007.

[16] M. Chen, S. Liew, Z. Shao, and C. Kai. Markov approximation for combinatorial network optimization. In *Proc. of IEEE INFOCOM*, 2010.

[17] P. Diaconis and D. Stroock. Geometric bounds for eigenvalues of Markov chains. *The Annals of Applied Probability*, 1(1):36–61, 1991.

[18] A. Dimakis, P. Godfrey, Y. Wu, M. Wainright, and K. Ramchandran. Network coding for distributed storage systems. *IEEE Trans. Inform. Theory*, 56(9):4539–4551, Sep. 2010.

[19] S. El Rouayheb and K. Ramchandran. Fractional repetition codes for repair in distributed storage systems. In *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pages 1510–1517. IEEE, 2010.

[20] B. Fan, D. Chiu, and J. Lui. The delicate tradeoffs in bittorrent-like file sharing protocol design. In *Proc. of IEEE ICNP*, 2006.

[21] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust incentive techniques for peer-to-peer networks. In *Proc. of ACM EC*, 2004.

[22] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross. A measurement study of a large-scale p2p iptv system. *IEEE Transactions on Multimedia*, 9(8):1672–1687, 2007.

[23] K. Ho, W. Poon, and K. Lo. Video-on-demand systems with cooperative clients in multicast environment. *Circuits and Systems for Video Technology, IEEE Transactions on*, 19(3):361–373, 2009.

[24] T. Ho, R. Koetter, M. Médard, M. Effros, J. Shi, and D. Karger. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, 52(10):4413–4430, October 2006.

[25] T. Ho, M. Médard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong. A random linear network coding approach to multicast. *Information Theory, IEEE Transactions on*, 52(10):4413–4430, 2006.

[26] C. Huang, J. Li, and K. Ross. Can Internet Video-on-Demand be Profitable? In *Proc. of ACM SIGCOMM*, 2007.

[27] Y. Huang, T. Fu, D. Chiu, J. Lui, and C. Huang. Challenges, design and analysis of a large-scale p2p-vod system. In *Proc. of ACM SIGCOMM*, 2008.

[28] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. Al Hamra, and L. Garces-Erice. Dissecting bittorrent: Five months in a torrent's lifetime. *Passive and Active Network Measurement*, pages 1–11, 2004.

[29] S. Jaggi, P. Sanders, P. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen. Polynomial time algorithms for multicast network code construction. *Information Theory, IEEE Transactions on*, 51(6):1973–1982, 2005.

[30] S. Jun and M. Ahamad. Incentives in bittorrent induce free riding. In *Proc. of ACM SIGCOMM workshop on Economics of peer-to-peer systems*, 2005.

[31] F. Kelly. *Reversibility and Stochastic Networks*. Wiley,Chichester, 1979.

[32] S. Kulkarni. Bandwidth efficient video-on-demand algorithm (beva). In *Telecommunications, 2003. ICT 2003. 10th International Conference on*, volume 2, pages 1335–1342. IEEE, 2003.

[33] N. Laoutaris, D. Carra, and P. Michiardi. Uplink allocation beyond choke/unchoke. In *Proc. of ACM CoNEXT*, 2008.

[34] N. Laoutaris, V. Zissimopoulos, and I. Stavrakakis. On the optimization of storage capacity allocation for content distribution. *Computer Networks*, 47(3):409–428, 2005.

[35] J. LaSalle. Some extensions of Liapunov's second method. *IRE Transactions on Circuit Theory*, 7(4), 1960.

[36] A. Legout, N. Liogkas, E. Kohler, and L. Zhang. Clustering and sharing incentives in bittorrent systems. In *Proc. of ACM SIGMETRICS*, 2007.

[37] A. Legout, G. Urvoy-Keller, and P. Michiardi. Rarest first and choke algorithms are enough. In *Proc. of ACM IMC*, 2006.

[38] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee. Bittorrent is an auction: analyzing and improving bittorrent's incentives. In *Proc. of ACM SIGCOMM*, 2008.

[39] S. Liew, C. Kai, H. Leung, and P. Wong. Back-of-the-envelope computation of throughput distributions in CSMA wireless networks. *IEEE Transactions on Mobile Computing*, 9:1319–1331, 2010.

[40] M. Lin, B. Fan, J. Lui, and D. Chiu. Stochastic analysis of file-swarming systems. *Performance Evaluation*, 64(9-12):856–875, 2007.

[41] X. Lin and N. Shroff. Utility maximization for communication networks with multipath routing. *IEEE Transactions on Automatic Control*, 51(5):766–781, 2006.

[42] Y. Liu, X. Liu, L. Xiao, L. Ni, and X. Zhang. Location-aware topology matching in p2p systems. In *proc. of IEEE INFOCOM*, 2004.

[43] Z. Liu, C. Wu, B. Li, and S. Zhao. UUSee: Large-Scale Operational On-Demand Streaming with Random Network Coding.

[44] L. Massoulie and M. Vojnovic. Coupon replication systems. *IEEE/ACM Transactions on Networking*, 16(3):603–616, 2008.

[45] P. Michiardi, K. Ramachandran, and B. Sikdar. Modeling seed scheduling strategies in bittorrent. *NETWORKING 2007. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*, pages 606–616, 2007.

[46] G. Neglia, G. Presti, H. Zhang, and D. Towsley. A network formation game approach to study bittorrent tit-for-tat. *Network Control and Optimization*, pages 13–22, 2007.

[47] S. Pawar, N. Noorshams, S. El Rouayheb, and K. Ramchandran. Dress codes for the storage cloud: Simple randomized constructions. In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pages 2338–2342. IEEE, 2011.

[48] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in bittorrent. In *Proc. of NSDI*, 2007.

[49] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *Proc. of IPTPS*, 2005.

[50] D. Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *Proc. of ACM SIGCOMM*, 2004.

[51] K. Rashmi, N. Shah, and P. Kumar. Enabling node repair in any erasure code for distributed storage. In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pages 1235–1239. IEEE, 2011.

[52] I. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[53] B. Sevastyanov. An ergodic theorem for Markov processes and its application to telephone systems with refusals. *Theory of probability and its applications*, 2:104, 1957.

[54] Z. Shao, H. Zhang, M. Chen, and K. Ramchandran. Reverse-Engineering BitTorrent. *Technical Report*, 2011. Available at http://personal.ie.cuhk.edu.hk/∼zyshao/bt.pdf.

[55] R. Srikant. *The mathematics of Internet congestion control*. Birkhauser, 2004.

[56] B. Tan and L. Massoulié. Brief announcement: adaptive content placement for peer-to-peer video-on-demand systems. In *Proc. of ACM PODC*, 2010.

[57] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez. Greening the internet with nano data centers. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 37–48. ACM, 2009.

[58] J. Wang, C. Huang, and J. Li. On isp-friendly rate allocation for peer-assisted vod. In *Proc. of ACM Multimedia*, 2008.

[59] J. Wu and B. Li. Keep cache replacement simple in peer-assisted vod systems. In *Proc. of IEEE INFOCOM*, 2009.

[60] S. Zhang, Z. Shao, and M. Chen. Optimal distributed p2p streaming under node degree bounds. In *Network Protocols (ICNP), 2010 18th IEEE International Conference on*, pages 253–262. IEEE, 2010.

[61] X. Zhou and C. Xu. Optimal video replication and placement on a cluster of video-on-demand servers. In *Parallel Processing, 2002. Proceedings. International Conference on*, pages 547–555. IEEE, 2002.

[62] X. Zhou and C. Xu. Efficient algorithms of video replication and placement on a cluster of streaming servers. *Journal of Network and Computer Applications*, 30(2):515–540, 2007.