NETWORKING AT INFORMATION CENTRIC EDGE

A Dissertation

by

TAO ZHAO

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	I-Hong Hou
Committee Members,	P. R. Kumar
	Le Xie
	Jyhwen Wang
Head of Department,	Miroslav Begovic

August 2019

Major Subject: Computer Engineering

ABSTRACT

More and more heterogeneous devices are getting connected to the Internet, leading us to the era of Internet of Things. However, this trend is hardly sustainable under today's Internet, because of the inefficiency of IP networking for user information and the potential long delay of cloud services. To address these issues, information centric networking and edge computing have been proposed in the literature. These two concepts naturally fit into each other, resulting in Information Centric Edge, an emerging network architecture where heterogeneous devices are connected mostly through wireless to nearby edge servers that have computation resources provisioned, and their inter-networking is centered on user contents. With information centric edge, users can get what they want faster. However, such network architecture also face challenges including limited edge resources, strategic users, implementation and so on.

In this dissertation, we address these challenges of realizing Information Centric Edge by designing and implementing systems and algorithms therein. To cope with the limited capacity of edge servers, we study an dynamic service caching problem and propose an online algorithm with provable worst-case performance guarantee. We also investigate a content caching problem that calls for joint optimization of content placement, cache selection, and video version selection, and present practical distributed algorithms that achieve close to optimal performance in implementation. To handle strategic users, we design a non-monetary mechanism via efficient cost allocation which makes distributed individual optimization converge to the global optimum. The same method is generalized to distribute revenue of a cyber-enabled manufacturing system to its geo-distributed suppliers. Besides, we are building an Information Centric Edge testbed with a programmable network stack for intelligent caching, routing, and scheduling algorithms to support emerging virtual reality applications in the era of Internet of Things.

DEDICATION

To my mother, my father, and my wife.

ACKNOWLEDGMENTS

I would like to thank my advisor and committee chair, Prof. Hou, and my committee members, Prof. Kumar, Prof. Xie, and Prof. Wang, as well as Prof. Shakkottai for their guidance and support throughout the course of this research.

Thanks also to my friends and colleagues (especially Han Deng, Ping-Chun Hsieh, Xi Liu, and Daojing Guo) and the department faculty and staff (especially Vickie Winston) for making my time at Texas A&M University a great experience. I also want to extend my gratitude to the funding agencies for their support of my research.

Besides, I would like to thank the Texas A&M University Office of Graduate and Professional Studies for providing the LaTeX thesis template. Special thanks to the reviewers for carefully reviewing this material.

Gig 'em Aggies!

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Professor I-Hong Hou [advisor], Professor P. R. Kumar, and Professor Le Xie of the Department of Eletrical & Computer Engineering and Professor Jyhwen Wang of the Department of Engineering Technology and Industrial Distribution.

All work for the dissertation was completed by the student, under the advisement of Professor I-Hong Hou. The work for Chapter 2 was also under the advisement of Dr. Shiqiang Wang and Dr. Kevin Chan. The work for Chapter 3 was carried out in collaboration with Ms. Archana Sasikumar, and also advised by Professor Srinivas Shakkottai. Chapter 4 and Chapter 5 were completed under the advisement of Professor Korok Ray. Besides, Professors V. Jorge Leon and Jyhwen Wang advised the work for Chapter 5. In addition, the testbed work for Chapter 6 was conducted in collaboration with my colleagues including Daojing Guo, Sibendu Paul (Purdue University), Mohd Faisal Khan, Narges Zarnaghi-Naghsh, and Santosh Ganji under the guidance of Professors I-Hong Hou, Srinivas Shakkottai, Charlie Hu (Purdue University), and P. R. Kumar.

Funding Sources

Graduate study was supported by a research grant from the Mays Innovation Research Center, Texas A&M University. This work was made possible in part by NSF under contract/Grant Number 1547867, CNS-1719384, and CNS-1149458, NSF-Intel CNS-1719384, AST 1443891, the U.S. Army Research Laboratory and the U.S. Army Research Office under contract/Grant Number W911NF-15-1-0279 and W911NF-18-1-0331, the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001, Office of Naval Research under Contract N00014-18-1-2048, and NPRP Grant 8-1531-2-651 of Qatar National Research Fund (a member of Qatar Foundation).

The views and conclusions contained in this document are those of the authors and should not

be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government.

NOMENCLATURE

AP	Access Point
API	Application Programming Interface
CNC	Computer Numerical Control
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
LP	Linear Program
LTE	Long Term Evolution
MAC	Medium Access Control
MIP	Mixed Integer Program
NDN	Named Data Networking
PC	Personal Computer
REST	REpresentational State Transfer
SDN	Software Defined Networking
VR	Virtual Reality

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

1.1 Background

Internet of Things (IoT) is turning from a vision to a reality. More and more heterogeneous devices are getting connected to the Internet every day. Smart home, the concept where home appliances such as refrigerators, thermometers, light bulbs become smart by connecting to the Internet, has reached hundreds of millions of households [1]. As for public infrastructure, smart grid renovates the electricity distribution network by integrating communication networks into the utility grid. Turning our focus to manufacturing industry, cyber-enabled manufacturing has been proposed to connect distributed manufacturing devices to a cloud platform to increase the efficiency of manufacturing process. One key question remains as we enter the era of IoT: is today's Internet good enough to meet the needs of IoT users?

We think the answer to the above question is no based on the following two observations. First, users care about the information they want instead of the IP address, although today's Internet is almost exclusively based on IP. Specifically, the dominant usage of Internet is world wide web, or simply the Web. Information on the Web is identified by its universal resource locator (URL), which includes a domain name part. To acquire certain information on the Web, we need domain name service (DNS) to translate the domain name to an IP address of the server hosting the information. This indirection increases the complexity and decreases the efficiency of the network. Second, today's Internet heavily rely on cloud services, which can incur large delay since cloud data centers are typically far away from the end users. Therefore, it is rather challenging to support delay sensitive IoT applications in today's Internet.

Fortunately, there are proposals addressing each of the two aforementioned issues. For the first issue, information centric networking (ICN) has been proposed to build the network centered on user demand. Unlike the current Internet architecture where IP is the thin waist, ICN argues that the thin waist should be content chunks, i.e. the network should focus on delivering content chunks

users need directly. In this way, user demand can be satisfied in a more direct and efficient fashion. For the second issue, the concept of edge computing, also known as fog networking, cloudlets, etc., has been proposed to push cloud services to the edge of the network, which is closer to mobile users and thus leads to less delay as well as less backhaul bandwidth usage. Therefore, better quality of service (QoS) and quality of experience (QoE) can be brought to end users.

It is natural to see that these two concepts fit into each other, resulting in Information Centric Edge, an emerging network architecture where heterogeneous devices are connected mostly through wireless to nearby edge servers that have computation resources provisioned, and their inter-networking is centered on user contents. With information centric edge, users can get what they want faster. In this dissertation, we would like to study the way of realizing Information Centric Edge by designing and implementing systems and algorithms therein.

1.2 Challenges and Research Problems

There are quite a few challenges ahead in realizing Information Centric Edge. One of the fundamental limitation of edge servers is that their capacity is rather limited compared to remote data centers, since they have to be geographically distributed. For example, it is unreasonable to assume an edge server would be able to host all cloud services or all contents users want. On the other hand, user demand is ever changing and not always predictable. Therefore, edge servers need to dynamically allocate their resources, or "cache" a subset of cloud services or user contents.

We identify two distinct types of caching in Information Centric Edge: service caching and content caching. Service caching, also known as service migration, or moving functions, refers to the scenario that cloud services, including their code, database, and state information, are downloaded and replicated at the edge server. On the other hand, content caching corresponds to dissemination of user contents e.g. video clips. We note that both are different from traditional data caching (also known as page replacement) problem extensively studied in computer architecture literature. Traditional data caching typically assumes all requested data are hot and thus cached to minimize cache misses. However, for service caching, downloading a service usually costs much more than simply forwarding a request. Hence, when to cache a service needs to be care-

fully decided. It is of great interest to study optimal algorithms for online service caching. As for content caching, whether to cache a particular content or not in a particular edge server, which is the content placement problem, needs investigation, since empirically a lot contents will only be requested for few times. Content placement has been extensively studied in the literature, while often assuming the user demand is given. However, we note that in Information Centric Edge, the decisions of content placement will affect and be affected by user decisions including which cache to use (cache selection) and which content to request (content selection). These subproblems are fundamentally intertwined and hence call for a joint optimization approach.

Another big challenge in Information Centric Edge is how to handle strategic users. Wireless clients typically aim to maximize their own individual utilities, and they are not willing to share private utility valuation truthfully. On the other hand, system designers would like to maximize the total utility of the whole system. This conflict of interests raises an interesting question: Can we make the Nash Equilibrium, where no individual user will deviate unilaterally, also optimal for the whole system? Existing studies suggest the answer is yes with the help of certain pricing mechanisms, such as auction or direct payment. However, we believe the monetary approach, although reasonable in economics, can be difficult to realize in Information Centric Edge, due to lack of monetary exchange infrastructure. A non-monetary approach would better fit Information Centric Edge. To realize that, we borrow the idea of efficient cost allocation from managing multiple departments in a large corporation. In the edge network, we can potentially allocate the total cost of the system, such as the total delay of all clients, in an efficient way so as to induce the individual optimization to converge to the global optimum. Similarly, it could be possible to distribute revenue to multiple individual entities in an efficient way to achieve global optimum through distributed optimization. How to design and enforce such efficient cost allocation or revenue distribution rules is an interesting research problem.

Besides, there is the challenge of implementation of Information Centric Edge, due to the well known gap between simulations and real world systems. Existing testbeds often focuses on only one or two layers of the whole network stack. In contrast, we would like to build an Information

Centric Edge testbed with a programmable network stack so that we can experiment with intelligent caching, routing, scheduling algorithms throughput the layers. One key research issue is: Can we support emerging virtual reality (VR) applications that require high throughput and low latency while providing high programmability?

Our dissertation attempts to shed lights on the answers to the proposed research problems to address these big challenges. The rest of this dissertation is organized as follows. Chapter 2 and Chapter 3 study the service caching problem and the content caching problem respectively. The efficient cost allocation problem and the efficient revenue distribution problem are investigated in Chapter 4 and Chapter 5 respectively. We present our Information Centric Edge testbed in Chapter 6, and conclude the dissertation in Chapter 7.

2. SERVICE CACHING AT THE EDGE*

Edge servers, which are small servers located close to mobile users, have the potential to greatly reduce delay and backhaul traffic of mobile Internet applications by moving cloud services to the edge of the network. Due to limited capacity of edge servers and dynamic request arrival, proper service caching at the edge is essential to guarantee good performance. This chapter proposes a tractable online algorithm called *retrospective download with least-requested deletion* (RED/LED) that caches services dynamically without any assumptions on the arrival patterns of mobile applications. We evaluate the competitive ratio of our policy, which quantifies the worst-case performance in comparison to an optimal offline policy. We prove that the competitive ratio of our policy is linear with the capacity of the edge server. We also show that no deterministic online policy can achieve a competitive ratio that is asymptotically better than ours. Moreover, we prove that our policy is scalable, in the sense that it only needs doubled capacity to achieve a constant competitive ratio. The utility of our online policy is further evaluated on real-world traces. These trace-based simulations demonstrate that our policy has better, or similar, performance compared to many intelligent offline policies.

2.1 Introduction

Many emerging mobile applications rely on cloud computing technology to greatly expand the capability of resource-constrained mobile devices. In a typical scenario, a mobile device sends a request, such as a picture containing text in a foreign language, to a remote cloud, which is often hosted by a remote data center. The remote cloud then generates a response, such as translations of the text, using its massive computational power and storage. However, the long distance between mobile devices and remote clouds can result in significant delay and severe burden on the backhaul connection, which can limit the development of real-time and data-intensive applications. The

*Reprinted with permission from "RED/LED: An asymptotically optimal and scalable online algorithm for service caching at the edge" by T. Zhao, I.-H. Hou, S. Wang, and K. S. Chan, *IEEE J. Sel. Areas Commun.*, Copyright 2018 IEEE

concept of edge computing, also known as cloudlets, fog computing, etc., has been proposed to address this issue [2, 3, 4]. In edge computing, small edge servers are deployed close to mobile users, such as at the locations of cellular base stations or Wi-Fi access points. These edge servers can host a small number of popular services, and provide timely response to local user requests directly without communicating with remote clouds. Edge servers are not necessarily real "servers". They can be, for example, home Wi-Fi routers in smart home environment, as suggested in [5].

Edge servers have limited computational power and storage compared to remote clouds. Thus, they can only "cache" a small number of services, out of all the available services hosted by the remote cloud [6, 7]. In this chapter, we say that a service is *cached* at the edge server when the entire set of code and data required for service execution has been downloaded from the remote cloud to the edge server. The edge server can fully execute a cached service on its own, without interacting with the remote cloud. Since the arrivals of service requests from mobile devices can be highly dynamic, proper service caching at the edge is essential to guarantee good performance, and it is challenging to find the optimal caching policy that determines which services to cache at the edge server.

2.1.1 Motivation

While there have been studies on optimal service caching at the edge, most of them assume either that the edge servers have reasonably good predictions about future requests [8, 9], or that the arrivals of requests follow a pre-specified stochastic process [10, 11, 12, 13, 14]. These studies then use predictions or stochastic models to determine the services that the edge servers should cache to achieve the optimal average performance. However, newly emerging services can be challenging to predict or model in advance. Besides, as suggested by a real-world trace of requests at the cloud [15], the request arrival patterns can change frequently over time, which are difficult to timely predict or model as a known stochastic process. In addition, predictions are generally imperfect. Algorithms that rely on predictions or stochastic models can result in poor performance for these systems. Further, many mission-critical systems require "worst-case" performance guarantees, instead of "average" performance guarantees.

In this chapter, we study online algorithms that determine which services to cache at the edge dynamically without making any assumptions on the arrival patterns of requests. We focus on deterministic policies since they have predictable system behavior and are generally more robust in practice. We consider an edge server that can cache only a limited number of services. When a request arrives but its service is not cached at the edge server, the edge server has two options: It can either forward the request to the remote cloud for processing, or it can download and cache the service so that it can directly serve the requests for this service in the future. Both options have a cost, which can reflect the delay and the network bandwidth usage. We assume the cost of downloading a service is larger than the cost of forwarding a request. This is motivated by the fact that forwarding a request to a remote cloud for processing has a response time that is usually within a second, as shown by the experimental results presented in [16]. Downloading a service, which entails both downloading necessary files and setting up a virtual machine or container, takes at least several seconds according to recent experimental studies [17, 18, 19].

We aim to design online algorithms that result in a small total cost, including the cost of forwarding requests and downloading services, for any sequence of request arrivals. As online algorithms have no knowledge about future arrivals, we evaluate the performance of an online policy by its *competitive ratio*, defined as the largest possible ratio between the cost of the online policy and the minimum (optimal) offline cost, under any sequence of arrivals. While this problem may seem to bear some similarities with the classic data caching problem, we note that the option of forwarding a request, and the significant difference between the costs of forwarding a request and downloading a service, make the problem of service caching fundamentally different from the problem of data caching. In fact, as we will demonstrate in Section 2.9, the Belady's algorithm, a well-known optimal offline policy for data caching, performs poorly for service caching.

2.1.2 Main Contribution

We first focus on a homogeneous system where all services require the same amount of computational power and storage, and they have the same forward cost as well as the same download

cost.[1] Using an observation of the optimal offline policy, we propose an online policy, *retrospective download with least-requested deletion* (RED/LED), for service caching at the edge, which is easy to implement in practice. We prove that the competitive ratio of our RED/LED policy only grows linearly with the capacity of the edge server. We further prove that no deterministic online policy can achieve a competitive ratio that is sublinear with the capacity of the edge server. Therefore, our RED/LED policy indeed achieves the optimal asymptotic performance. Moreover, we prove that our policy is scalable, in the sense that if the capacity of the edge server is doubled for RED/LED, it can achieve a constant competitive ratio with regard to the optimal offline batch-download policy.

We then address several practical issues of RED/LED. We demonstrate that it can be implemented as an algorithm with linear time complexity. We also propose an extension of RED/LED for heterogeneous systems where different services may be associated with different costs and require different amounts of edge-server resources.

We evaluate the performance of our RED/LED policy using real-world traces of service request arrivals. We compare our policy against a randomized online policy, the optimal offline batch-download policy, and three other offline policies that correspond to solutions based on dynamic programming, stochastic optimization, and data caching respectively. We note that the time complexity of the optimal offline algorithm is very high, and all the offline policies have complete knowledge of all future arrivals. Each of the offline policies achieves the optimal performance under some specific scenarios. Simulation results show that our policy achieves better, or at least similar, performance compared to all these policies in both homogeneous and heterogeneous systems.

The rest of the chapter is organized as follows. Section 2.2 reviews related studies. Section 2.3 formally describes the service caching problem. Section 2.4 introduces our online policy based on a property of the optimal offline policy. Section 2.5 derives the competitive ratio of our policy. Section 2.6 proves that no deterministic online policy can be asymptotically better than ours. Section 2.7 shows the scalability of our policy. Section 2.8 addresses practical issues including het-

[1]The download cost is different from the forward cost.

erogeneous systems and low complexity implementation. Section 2.9 compares our policy against several others through trace-based simulations. Finally, Section 2.10 concludes the chapter.

2.2 Related Work

The dramatic increase in network traffic, particularly due to the proliferation of mobile devices and Internet of Things (IoT), has made it critical to move some computing and storage jobs from remote data centers to the edge of the network. There have been a number of architecture proposals of edge computing in the literature [2, 3, 20, 21, 4, 22, 7]. The utility of edge computing has been demonstrated by various prototypes [23, 24, 5]. Comprehensive surveys on this new paradigm have been published recently [25, 26].

To address the challenge of managing limited resources of edge servers, some studies rely on an accurate prediction of future requests. Tadrous et al. [8] have considered systems where one can predict the popularity of services, so that edge servers can proactively download and replicate popular services during off-peak hours. Llorca et al. [9] have studied the content placement problem and proposed an optimal offline policy. Pasteris et al. [27] have proposed an approximation algorithm for multiple edge servers. Wang et al. [21] have proposed an online algorithm with polynomial time complexity to minimize the long-term average cost. Yang et al. [28] have studied the joint optimization of service placement and load dispatching. Unlike these existing studies, in this chapter we investigate online policies which assume no knowledge about future requests.

Besides, there are many studies that employ stochastic optimization for service caching at the edge. Amble et al. [10] have considered a system where request arrivals follow an independent and identically distributed (i.i.d.) stochastic process with unknown distribution, and proposed a stochastic control policy that maximizes the capacity region of the system. Borst et al. [12] have proposed an algorithm for a static system where the popularity of services does not change over time. On the other hand, Wang et al. [13] and Urgaonkar et al. [14] have considered dynamic systems where request arrivals are modeled as a Markov process, and proposed solutions that achieve the optimal long-term average performance. Qiu et al. [11] have employed Lyapunov optimization to maximize the performance of edge servers. However, these solutions based on

stochastic optimization cannot provide "worst-case" performance guarantees with regard to an optimal offline policy. Our work, on the other hand, addresses such issues.

The problem of service caching at the edge bears some similarities with the classic data caching problem in computer architecture. In the data caching problem, requests for data arrive sequentially, and a "miss" occurs when the data is not stored in the cache. The requested data is then downloaded, and a policy needs to decide which data to delete to minimize the total number of misses. When one has complete knowledge of all future requests, the Belady's algorithm [29] achieves the optimal performance. As for online policies without any knowledge of future requests, Sleator and Tarjan [30] have established a least recently used (LRU) policy that is optimal among all deterministic policies. They have also showed that LRU has a constant competitive ratio if its cache capacity is a constant factor of that of the optimal offline policy. Achlioptas et al. [31] have studied the competitive ratios of randomized policies. Bansal et al. have investigated the weighted case where the download costs vary for different data [32]. Despite the similarities, there are notable differences between the data caching problem and the service caching problem. In the data caching problem, one cache miss implies one download with some cost, and forwarding is disallowed. However, in the service caching problem, forwarding is permitted and incurs a smaller cost than downloading, and a policy needs to decide whether to download a service in addition to which service to delete.

2.3 System Model

We consider an edge-cloud system as illustrated in Figure 2.1. An edge server and a back-end cloud are connected through a backhaul connection. The edge server and the back-end cloud jointly host a set $\mathbb{S}$ of services, numbered as $S_1, S_2, \ldots$. The services in the system include face detection, video streaming, translation, smart home services and so on [17, 5]. The back-end cloud has massive capacity, and can host all services. On the other hand, the edge server has limited capacity and can only cache K services. Without loss of generality, we assume that, when the system starts, the edge server caches services $S_1, S_2, \ldots, S_K$.

Requests for services arrive at the edge server sequentially. Requests from different mobile

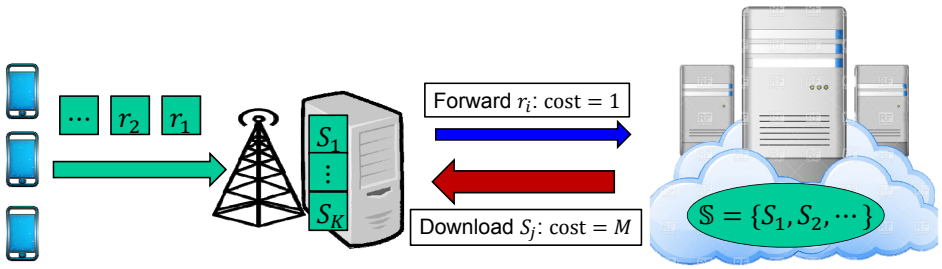


Figure 2.1: An illustration of the edge-cloud system. The back-end cloud can host all services $\mathbb{S}$, while the edge server can only cache a subset of size K. Requests can incur different costs depending on which services are cached at the edge. Reprinted with permission from [33].

devices can arrive at and be processed by the edge server either on a first-come-first-served basis, or based on some other scheduling mechanism using queues/buffers. We use $r_n \in \mathbb{S}$ to denote the service requested by the n-th request. If r_n is cached by the edge server when the request arrives, then the edge server can serve the request immediately without causing any significant cost. On the other hand, if r_n is not cached by the edge server, then the edge server has two choices: First, it can forward this request to the back-end cloud for processing. This will cause some delay as well as some traffic on the backhaul connection. We say that forwarding a request to the back-end cloud incurs a cost of one unit. Second, the edge server can instead download and replicate the whole service r_n and serve the request. Downloading a service can cause much higher delay and more traffic. Therefore, we say that each service download incurs a cost of M units, with $M \geq 1$. In practice, we can choose M as the ratio of the average delay of downloading to the average delay of forwarding. On the other hand, since the edge server caches the service r_n after the download, it can then serve subsequent requests for r_n without incurring any costs. The edge server can only cache K services. Therefore, when it downloads a service, it also needs to delete a service from its storage.

We aim to minimize the total cost of the system by intelligently reconfiguring the set of services cached by the edge server. Intuitively, if we know that a service will have a lot of requests in the near future, we should download this service so that all these requests only incur M units of cost. Otherwise, we should simply forward all these requests to the back-end cloud without

downloading the service, and pay one unit of cost for each request. In practice, however, we may not have accurate prediction for future requests. In such cases, we need to rely on online policies that assume no information about future requests.

Let OPT be the optimal offline policy that minimizes the total cost of the system, which has full information about all future request arrivals. Let η be an online policy that makes its decision solely based on past events. For a given sequence of request arrivals, $r_1, r_2, \ldots$, let C_{OPT} be the total cost of OPT, and C_η be the total cost of η. Note that the total costs, C_{OPT} and C_η, are functions of the sequence $r_1, r_2, \ldots$, but we omit the sequence to simplify the notation. There may be multiple offline policies that achieve the minimum cost. In this case, we let OPT be one that makes the most downloads among them. We evaluate the performance of an online policy η by its *competitive ratio*, which is the largest possible value of C_η / C_{OPT}, over all possible sequences of request arrivals.

Definition 1 (Competitive Ratio)**.** *An online policy η is said to be β-competitive, or have a competitive ratio of β, if $C_\eta \leq \beta C_{OPT}$, for every possible sequence of request arrivals.*

An online policy with a low competitive ratio has similar performance with the optimal offline policy. Therefore, we aim to develop an online policy with a low competitive ratio, as well as a lower bound of competitive ratios for all online policies.

For brevity, we say that a policy caches a service if, under the said policy, the edge server caches the service. To facilitate the analysis, let $\zeta(n) \subset \mathbb{S}$ denote the subset of services cached by a policy ζ, online or offline, after the n-th arrival, for a given sequence of request arrivals, $r_1, r_2, \ldots$. Again, note that $\zeta(n)$ is a function of the sequence of arrivals. For a service S_i, we use $x_i(n) := \mathbb{1}\{r_n = S_i\}$ to indicate whether or not the n-th request is for this service. For brevity, let $[n, m]$ denote the inclusive time interval between the n-th and the m-th arrival.[2] Therefore, $\sum_{l=n}^{m} x_i(l)$ is the number of requests for S_i during $[n, m]$.

Before proceeding to the next section, we note that the system model in this section is a ho-

[2]Throughout this chapter, $[n, m] = \{n, n+1, \ldots, m\}$. Time refers to indices of request arrivals, which are not necessarily physical time.

mogeneous one: All services have the same cost of forwarding requests, and the same cost of downloading. Moreover, all services require the same amount of edge-server resources. While our analytical results mainly focus on the homogeneous system, we will show that our policy can be easily extended to heterogeneous systems in Section 2.8, and it works very well in evaluation in Section 2.9.

2.4 The RED/LED Online Policy

In this section, we first establish a basic property of OPT, and then use it to develop our online policy RED/LED.

2.4.1 A Basic Property of OPT

Theorem 1. *Suppose we are given a sequence $r_1, r_2, \ldots$, and OPT(n), which is the subset of services cached by OPT after the n-th arrival. If there exists an integer $m > n$, a service $S_i \notin$ OPT(n), and another service $S_j \in$ OPT(n) such that:*

$$\sum_{l=n+1}^{m} x_i(l) \geq \sum_{l=n+1}^{m} x_j(l) + 2M, \tag{2.1}$$

then OPT downloads at least one service during $[n+1, m]$.

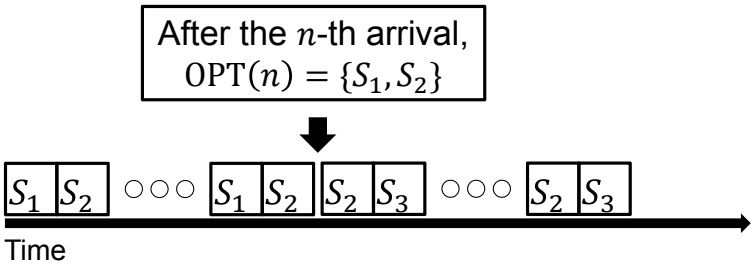


Figure 2.2: An example illustrating operations of OPT and RED/LED. The first n arrivals are $S_1, S_2, \ldots, S_1, S_2$, and the next $4M$ arrivals are $S_2, S_3, \ldots, S_2, S_3$. Reprinted with permission from [33].

Before proving Theorem 1, we first use Fig. 2.2 for illustration. Suppose the first n arrivals are

$S_1, S_2, \ldots, S_1, S_2$, and the edge server can cache two services. After the n-th arrival, OPT caches S_1 and S_2. Between the $(n+1)$-th arrival and the $(n+4M)$-th arrivals, we have $2M$ requests for S_3 and no requests for S_1, that is, $\sum_{l=n+1}^{n+4M} x_1(l) = 0$, and $\sum_{l=n+1}^{n+4M} x_3(l) = 2M$. Theorem 1 then states that OPT downloads at least one service during $[n+1, n+4M]$. Note that Theorem 1 does not specify which service to download, and which service to delete from the edge server.

Proof of Theorem 1. We now prove Theorem 1 by contradiction. Suppose OPT does not download any service, and therefore does not delete any service, during $[n+1, m]$. That is, $\text{OPT}(l) = \text{OPT}(n)$, for all $n+1 \leq l \leq m$.

We construct a different policy ξ as follows: ξ caches the same subset of services as OPT before the n-th arrival and after the $(m+1)$-th arrival, that is, $\xi(l) = \text{OPT}(l)$, for all $l \leq n$, and all $l \geq m+1$. After the n-th arrival, ξ downloads S_i and deletes S_j so that $\xi(n+1) = \text{OPT}(n) \setminus \{S_j\} \cup \{S_i\}$. After the m-th arrival, ξ downloads S_j and deletes S_i, and then follows the same decisions that OPT makes so that $\xi(m+1) = \text{OPT}(m+1)$.

We now compare the costs of ξ and OPT. Since ξ and OPT are the same before the n-th arrival and after the m-th arrival, they incur the same amount of costs in these two durations. Between the n-th arrival and the m-th arrival, ξ downloads two services and OPT downloads none. Therefore, ξ needs to pay $2M$ units of cost. Meanwhile, ξ needs to pay one unit cost for each request for S_j, and there are $\sum_{l=n+1}^{m} x_j(l)$ of them, while OPT needs to pay one unit cost for each of the $\sum_{l=n+1}^{m} x_i(l)$ requests for S_i. The two policies incur the same amount of costs for all other requests. In summary, we have:

$$C_\xi = C_{\text{OPT}} + 2M + \sum_{l=n+1}^{m} x_j(l) - \sum_{l=n+1}^{m} x_i(l) \leq C_{\text{OPT}},$$

where the last inequality follows by (2.1). Therefore, ξ also minimizes the total cost. Moreover, ξ makes two more downloads than OPT. Recall that OPT is defined as the policy that makes the most downloads among all policies with minimum cost. The existence of ξ therefore contradicts with assumptions of OPT. □

2.4.2 The RED/LED Online Policy

We now introduce our online policy. The policy needs to consist of two parts: deciding whether to download a service, and, when a download occurs, deciding which service to delete from the edge server. We propose the *retrospective download with least-requested deletion* (RED/LED) policy that uses a *retrospective download* (RED) policy for the first part, and a *least-requested deletion* (LED) policy for the second part. We use RL(n) to denote the subset of services cached by RED/LED after the n-th request arrival.

RED is used to determine whether to download a service, and is formally defined as follows:

Definition 2 (RED)**.** *When a request $r_n = S_i \notin$ RL$(n-1)$ arrives,* RED *downloads and replicates S_i at the edge server if there exists an integer τ and a service S_j such that for all $n-\tau \leq l \leq n-1$, $S_j \in$ RL(l) and $S_i \notin$ RL(l), and*

$$\sum_{l=n-\tau}^{n} x_i(l) \geq \sum_{l=n-\tau}^{n} x_j(l) + 2M. \tag{2.2}$$

The intuition of RED comes from Theorem 1. Suppose $S_j \in$ OPT$(n-\tau)$, $S_i \notin$ OPT$(n-\tau)$, and (2.2) holds. Then Theorem 1 states that OPT downloads at least one service between the $(n-\tau)$-th arrival and the n-th arrival. RED then downloads S_i when, in retrospect, it finds OPT would have already downloaded a service.

When RED decides to download a service, we need to choose a service to delete from the edge server. We propose a *least-requested deletion* (LED) policy as follows:

Definition 3 (LED)**.** *Suppose the edge server decides to download a service at the n-th arrival. For each service S_i currently cached by the edge server, let τ_i be the smallest integer such that there are at least $2M$ requests for S_i in the past τ_i arrivals, that is, $\sum_{l=n-\tau_i}^{n} x_i(l) \geq 2M$.* LED *deletes the service with the largest τ_i.*

A service with larger τ_i needs to go further back in time to find at least $2M$ requests. LED therefore deletes the service that is least requested starting from $n-\tau_i$.

RED/LED uses RED to decide whether to download a service, and LED to decide which service to delete. We use Fig. 2.2 to illustrate the operation of RED/LED. Suppose RL$(n) = \{S_1, S_2\}$, that is, RED/LED caches S_1 and S_2 after the n-th arrival. At the $(n + 4M)$-th arrival, we find that $\sum_{l=n+1}^{n+4M} x_3(l) = 2M \geq \sum_{l=n+1}^{n+4M} x_1(l) + 2M$, and RED decides to download S_3 at the $(n + 4M)$-th arrival. Meanwhile, during $[n + 1, n + 4M]$, there are $2M$ requests for S_2 and none for S_1. We then have $\tau_1 > \tau_2$, and LED decides to delete S_1 to accommodate S_3.

2.5 The Competitive Ratio of RED/LED

This section establishes the competitive ratio of RED/LED by proving the following theorem:

Theorem 2. RED/LED *is* $10K$*-competitive, where* K *is the number of services that can be cached by the edge server.*

2.5.1 Overview of the Proof

We will compare the performance of RED/LED and OPT. Given OPT and the arrival sequence, we can divide the arrival sequence into frames, $[t_1 + 1, t_2], [t_2 + 1, t_3], \ldots$, so that OPT downloads services only at the beginning of frames, i.e., after the t_1-th, t_2-th, $\ldots$, arrivals. Before the first download under OPT, there must be no download under RED/LED due to its retrospective nature. Therefore, RED/LED and OPT behave the same during $[1, t_1]$, and below we will focus on the performance in frames $[t_1 + 1, t_2], [t_2 + 1, t_3], \ldots$.

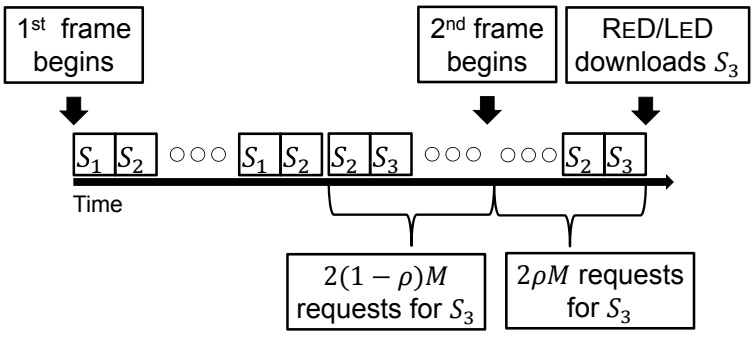


Figure 2.3: An example for dividing the costs into frames. Reprinted with permission from [33].

We will define the costs of OPT and RED/LED in each frame. We define the cost of OPT in a frame as the sum of the download cost at the beginning of the frame and all the costs of forwarding requests to the back-end cloud during the frame. On the other hand, it can be challenging to define the cost of RED/LED in each frame properly. Consider the example in Fig. 2.3. RED/LED downloads S_3, and incurs M units of download cost, shortly after the second frame begins. The decision to download S_3 actually involves many requests in the first frame. It is then unreasonable to count all the M units of download cost against the second frame. Instead, we separate the M units of download cost as follows: Suppose there are only $2\rho M$, where $\rho < 1$, requests for S_3 in the second frame when RED/LED downloads it, we say that the download is a *partial download* of fraction ρ, and only incurs ρM units of cost. At the same time, another partial download of fraction $(1-\rho)$ occurs at the end of the first frame, and incurs $(1-\rho)M$ units of cost. This separation does not change the total cost of RED/LED. To further simplify the notation in the next subsection, we say that, at the end of a frame, all services not cached by RED/LED have a partial download, possibly with fraction 0. The cost of RED/LED in a frame will then consist of all the download costs, including those from partial downloads, and all the costs of forwarding requests to the back-end cloud, in this frame.

Below, we will calculate the costs of OPT and RED/LED in each frame, and prove Theorem 2 by showing that RED/LED incurs at most $10K$ times as much cost as OPT in each frame.

2.5.2 Costs in a Frame

Without loss of generality, we calculate the costs in a frame $[t_g + 1, t_{g+1}]$. We use $C_{\text{OPT}}(g)$ and $C_{\text{RL}}(g)$ to denote the costs of OPT and RED/LED in this frame, respectively.[3]

We first consider a service $S_i \in \text{OPT}(t_g + 1)$. In the frame, let $E_i \in \mathbb{Z}$ be the number of times RED/LED downloads S_i, and $D_i \in \mathbb{Z}$ be the number of times RED/LED deletes S_i.[4] Note we have $D_i \geq E_i - 1$, as a service needs to be deleted first in order to be downloaded again. Let $f_{i,z}$ be the number of requests for S_i that RED/LED forwards to the back-end cloud within the z-th

[3]There can be partial downloads of services under RED/LED during $[1, t_1]$. In this case, $C_{\text{RL}} \leq 1.5C_{\text{OPT}}$ during $[1, t_1]$ and the proof is a special case of the latter proof.

[4]E_i includes partial downloads and each partial download counts as one.

interval of S_i not being cached at the edge server, i.e., after the previous deletion and before the next download of S_i, in this frame. Fig. 2.4 illustrates an example of the downloads and deletions of S_i in a frame, as well as the definition of $f_{i,z}$. S_i then incurs at most

$$E_i M + \sum_z f_{i,z} \tag{2.3}$$

units of cost under RED/LED.

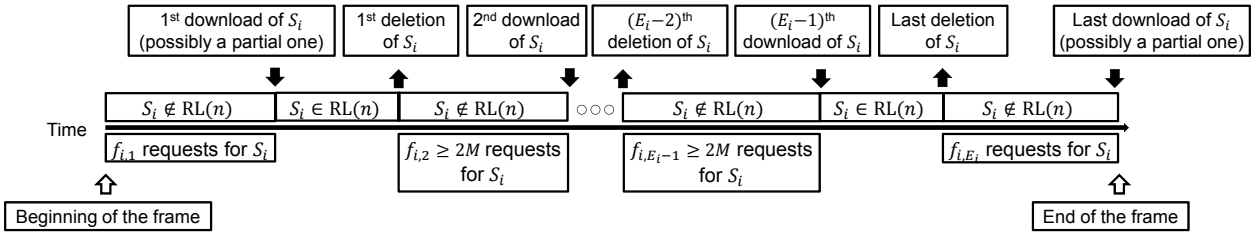


Figure 2.4: An example of the downloads and deletions of S_i in a frame. Reprinted with permission from [33].

On the other hand, we have the following lemma for OPT:

Lemma 1. *Suppose there exist integers $n < m$ and a service S_i such that for all $n \leq l \leq m$, OPT(l) = OPT(n), $S_i \in$ OPT(l), and $S_i \notin$ RL(l), then the number of requests that OPT forwards to the back-end cloud during $[n, m]$ is at least $\sum_{l=n}^{m} x_i(l) - 4M$.*

Proof. See Appendix A.1. □

By Lemma 1, if $f_{i,z} > 4M$ for some i, z, then during the time of these $f_{i,z}$ requests for S_i, OPT forwards at least $f_{i,z} - 4M$ requests to the back-end cloud, and incurs at least $f_{i,z} - 4M$ units of cost. Apply this argument for all z, and we have $C_{\text{OPT}}(g) \geq \sum_z (f_{i,z} - 4M)^+ + M$, where $x^+ := \max\{0, x\}$. The last term M is the download cost at the beginning of the frame. Let $\mathcal{S}_{\text{OPT}} := \{i \in \mathbb{Z} \mid S_i \in \text{OPT}(t_g + 1)\}$ denote the set of indices of services that are cached by OPT

in the g-th frame. We now have the first bound on $C_{\text{OPT}}(g)$:

$$\begin{aligned} C_{\text{OPT}}(g) &\geq \max_{i \in \mathcal{S}_{\text{OPT}}} \sum_{z=1}^{E_i} (f_{i,z} - 4M)^+ + M \\ &\geq \max_{i \in \mathcal{S}_{\text{OPT}}} \left(\sum_{z=1}^{E_i} f_{i,z} - 4ME_i \right) + M =: B_1. \end{aligned} \tag{2.4}$$

Next, we consider the deletions of S_i. We have the following lemma for OPT.

Lemma 2. *Suppose* RED/LED *deletes a service S_i at the n-th arrival and at the m-th arrival, $n < m$, and for all $n \leq l \leq m$, OPT(l) = OPT(m), then OPT forwards at least $2M$ requests to the back-end cloud during $[n, m]$.*

Proof. See Appendix A.2. □

By Lemma 2, for every $z > 1$, OPT forwards at least $2M$ requests between the $(z-1)$-th deletion and the z-th deletion of S_i. This gives us the second bound on $C_{\text{OPT}}(g)$:

$$\begin{aligned} C_{\text{OPT}}(g) &\geq \max_{i \in \mathcal{S}_{\text{OPT}}} 2M(D_i - 1) + M \\ &\geq \max_{i \in \mathcal{S}_{\text{OPT}}} 2ME_i - 3M =: B_2. \end{aligned} \tag{2.5}$$

Finally, we consider a service $S_j \notin \text{OPT}(t_g + 1)$. Let $\mathcal{S}_{\text{OPT}}^{\complement} := \{j \in \mathbb{Z} \mid S_j \in \mathbb{S} \setminus \text{OPT}(t_g + 1)\}$ denote the set of indices of services that are *not* cached by OPT in the g-th frame. Suppose there are A_j requests for S_j in this frame. OPT needs to forward all these requests to the back-end cloud, for all $S_j \notin \text{OPT}(t_g + 1)$, which gives us the third bound on $C_{\text{OPT}}(g)$:

$$C_{\text{OPT}}(g) \geq \sum_{j \in \mathcal{S}_{\text{OPT}}^{\complement}} A_j + M =: B_3. \tag{2.6}$$

On the other hand, RED/LED might either forward or download for each request for S_j. With the concept of partial download, each request for S_j incurs a download cost of at most $\frac{M}{2M} = 0.5$ in proportion, since at least $2M$ requests for S_j in retrospect are needed for RED to make one

download of S_j. Besides, one request incurs at most one unit of forward cost. Hence, A_j requests for S_j incur at most

$$(1 + 0.5)A_j = 1.5A_j \tag{2.7}$$

units of cost.

Combining (2.3) and (2.7) gives us a bound on $C_{\text{RL}}(g)$:

$$C_{\text{RL}}(g) \leq \sum_{i \in \mathcal{S}_{\text{OPT}}} \left(E_i M + \sum_z f_{i,z} \right) + \sum_{j \in \mathcal{S}_{\text{OPT}}^{\complement}} 1.5A_j \tag{2.8}$$

We are now ready to prove Theorem 2.

Proof of Theorem 2.

$$\begin{aligned}
C_{\text{RL}}(g) &\leq \sum_{i \in \mathcal{S}_{\text{OPT}}} \left(E_i M + \sum_z f_{i,z} \right) + \sum_{j \in \mathcal{S}_{\text{OPT}}^{\complement}} 1.5A_j \\
&\leq K \left[\max_{i \in \mathcal{S}_{\text{OPT}}} \left(E_i M + \sum_z f_{i,z} \right) + \sum_{j \in \mathcal{S}_{\text{OPT}}^{\complement}} 1.5A_j \right] \\
&\leq K \left[\max_{i \in \mathcal{S}_{\text{OPT}}} \left(E_i M + \sum_z f_{i,z} \right) + \sum_{j \in \mathcal{S}_{\text{OPT}}^{\complement}} 6.5A_j \right] \\
&= K(B_1 + 2.5B_2 + 6.5B_3) \leq 10KC_{\text{OPT}}(g),
\end{aligned}$$

for every frame. □

2.6 Lower Bound of the Competitive Ratio

In this section, we prove that the competitive ratio of any deterministic online policy is at least K. Since the competitive ratio of RED/LED is $10K = \Theta(K)$, this implies that RED/LED is asymptotically optimal among all deterministic online policies with respect to K, i.e. RED/LED performs at most a constant factor worse than the best deterministic online policy for large K.

Theorem 3. *The competitive ratio of any deterministic online policy is at least K.*

Proof. Given a deterministic online policy η, we construct a sequence of arrivals as follows: When the system starts, the first N_1 arrivals are all requests for a service $Z_1 \notin \{S_1, S_2, \ldots, S_K\}$. Recall that the edge server caches services $\{S_1, S_2, \ldots, S_K\}$ when the system starts. Therefore, the service Z_1 is not initially cached by the edge server. If η never downloads Z_1, then we choose N_1 to be arbitrarily large, and the system ends after N_1 arrivals of Z_1. In this case, OPT can download Z_1 at the first arrival, and only incurs a cost of M, while η incurs a cost of N_1. The competitive ratio of η is then $\frac{N_1}{M}$, which can be made arbitrarily large. From now on, we can assume that η downloads Z_1 after a finite number of requests for Z_1, and choose the value of N_1 so that η downloads Z_1, and deletes a service, after N_1 arrivals of Z_1. Let $Z_2 \in \{S_1, S_2, \ldots, S_K\}$ be the service that η deletes.

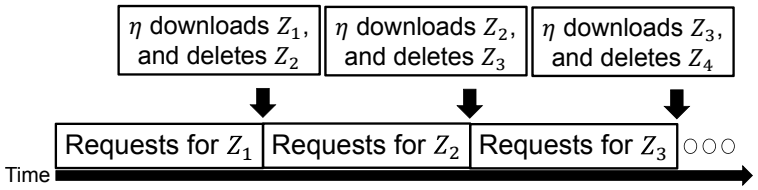


Figure 2.5: An example for the proof of Theorem 3. Reprinted with permission from [33].

We construct the remaining of the sequence of arrivals iteratively: For all $k = 2, 3, \ldots, K$, there are N_k requests for service Z_k after the first $\sum_{l=1}^{k-1} N_l$ arrivals. If η never downloads Z_k, we can make the competitive ratio arbitrarily large by choosing N_k to be arbitrarily large. Therefore, we can assume that η downloads Z_k after a finite number of requests for Z_k, and choose N_k to be that number. Let $Z_{k+1} \in \{S_1, S_2, \ldots, S_K, Z_1\}$ be the service that η deletes when it downloads Z_k. The system ends after the last N_K requests for Z_K. Fig. 2.5 illustrates such a sequence.

By the construction of our sequence, η makes K downloads and therefore incurs a cost of at least KM. On the other hand, there are at most K different services among $\{S_1, S_2, \ldots, S_K, Z_1\}$ that have any requests in this sequence. Therefore, at least one service in $\{S_1, S_2, \ldots, S_K\}$ does not have any requests. Let Z^* be that service. An offline policy can then download Z_1 and delete Z^* when the system starts. This only incurs a cost of M, as all subsequent requests are directly

served by the edge server. Therefore, the competitive ratio is at least $\frac{KM}{M} = K$. □

2.7 Scalability of RED/LED

Theorem 3 describes a rather pessimistic result: the competitive ratio of any deterministic online policy at best increases with the capacity of the edge server. In this section, we consider improving the scalability of online policies by provisioning additional capacity to the edge server. We show that, by doubling the capacity of the edge server, our RED/LED policy achieves a constant relative cost in comparison with the optimal offline batch-download policy, which will be defined below.

2.7.1 Definitions

Definition 4 (Batch-download policy)**.** *A policy is said to be a* batch-download *policy if, whenever it makes a download, it downloads K services to the edge server, and incurs a download cost of KM.*

Definition 5 (OPTb)**.** *A policy is an* optimal offline batch-download *policy and denoted by OPTb, if it minimizes the total cost of the system under any sequence of request arrivals, with full knowledge of the request sequence, among all batch-download policies.*

Obviously, OPTb cannot be better than OPT. However, it is indeed OPT in the special case where $K = 1$. Moreover, as we will show in Section 2.7.3, there exists a polynomial time algorithm for finding OPTb, and our trace-based simulations in Section 2.9 suggest that OPTb has similar performance as OPT in realistic settings.

We now introduce another definition of competitive ratio that takes into account the possibility of increasing the capacity of the edge server for online policies. For fair comparison under different capacities, we assume when the system starts, both edge servers cache "pseudo" services that have no request arrivals.

Definition 6. *An online policy η is said to be (α, β)-OPTb-competitive, if $C_\eta \leq \beta C_{OPTb}$ for every possible sequence of request arrivals, where C_η and C_{OPTb} is the total cost of η and OPTb respectively, and the capacity of the edge server under η is α times of that under OPTb.*

2.7.2 Competitive Ratio of RED/LED With Additional Capacity

Theorem 4. RED/LED *is* $(2, 10)$*-OPTb-competitive.*

Throughout this section below, we assume RED/LED operates on an edge server with capacity $2K$, while OPTb operates on an edge server with capacity K.

Similar to Section 2.5, we shall divide the arrival sequence into frames so that OPTb downloads services only at the beginning of frames. We can assume the first frame always starts before any request arrival, since otherwise it does not increase the cost of OPTb by moving its first download to the very beginning.[5] Then we compare the costs of RED/LED and OPTb in each frame. In the g-th frame, first consider a service S_i that is cached by OPTb. Recall that $\sum_{l=n}^{m} x_i(l)$ is the number of requests for S_i during $[n, m]$. We have the following lemma for OPTb.

Lemma 3. *Suppose there exist integers* $n < m$ *such that for all* $n \leq l \leq m$, *OPTb*$(l) =$ *OPTb*(n), *and* $S_i \in$ *OPTb*(l), *but* $S_i \notin$ *RL*(l), *then the number of requests that OPTb forwards to the back-end cloud during* $[n, m]$ *is at least*

$$K\left(\sum_{l=n}^{m} x_i(l) - 4M\right). \tag{2.9}$$

Proof. See Appendix A.3. □

Recall that E_i (resp. D_i) is the number of times RED/LED downloads (resp. deletes) S_i in the frame, and $f_{i,z}$ is the number of requests for S_i that RED/LED forwards to the back-end cloud within the z-th interval of S_i not being cached at the edge server in this frame. By Lemma 3, OPTb forwards at least $K(f_{i,z} - 4M)$ requests to the back-end cloud. Summing up for all z, we have the first bound on the cost of OPTb in this frame, $C_{\text{OPTb}}(g)$, as follows:

[5] If there is no download under OPTb, then $C_{\text{RL}} \leq 1.5C_{\text{OPTb}}$, and the proof is a special case of the later proof.

$$C_{\text{OPTb}}(g) \geq \max_{i \in \mathcal{S}_{\text{OPTb}}} \sum_{z=1}^{E_i} [K(f_{i,z} - 4M)]^+ + KM$$
$$\geq \max_{i \in \mathcal{S}_{\text{OPTb}}} \left(\sum_{z=1}^{E_i} K f_{i,z} - 4KME_i \right) + KM =: B_1', \tag{2.10}$$

where $\mathcal{S}_{\text{OPTb}} := \{i \in \mathbb{Z} \mid S_i \in \text{OPTb}(t_g + 1)\}$ is the set of indices of services that are cached by OPTb in the g-th frame. Note that B_1' has a factor of K compared with B_1 in (2.4).

Next, consider the deletions of S_i under RED/LED. We have the following lemma:

Lemma 4. *Suppose* RED/LED *deletes a service S_i at the n-th arrival and at the m-th arrival, $n < m$, and for all $n \leq l \leq m$, OPTb(l) = OPTb(m), then OPTb forwards at least $2KM$ requests to the back-end cloud during $[n, m]$.*

Proof. See Appendix A.4. □

By Lemma 4, between each two consecutive deletions of S_i, OPTb needs to forward at least $2KM$ requests. Therefore, we obtain the second bound on $C_{\text{OPTb}}(g)$ as follows:

$$C_{\text{OPTb}}(g) \geq \max_{i \in \mathcal{S}_{\text{OPTb}}} 2KM(D_i - 1) + KM$$
$$\geq \max_{i \in \mathcal{S}_{\text{OPTb}}} 2KME_i - 3KM =: B_2'. \tag{2.11}$$

Again, note the factor of K in B_2' due to the fact that RED/LED with double capacity can cache K more services than OPTb.

Finally, we consider a service $S_j \notin \text{OPTb}(t_g+1)$. Let $\mathcal{S}^{\complement}_{\text{OPTb}} := \{j \in \mathbb{Z} | S_j \in \mathbb{S} \backslash \text{OPTb}(t_g+1)\}$ denote the set of indices of services that are *not* cached by OPTb in the g-th frame. Recall A_j is the number of requests for S_j in this frame. OPTb needs to forward all these requests to the back-end cloud, for all $S_j \notin$ OPTb, which gives us the third bound on $C_{\text{OPTb}}(g)$:

$$C_{\text{OPTb}}(g) \geq \sum_{j \in \mathcal{S}^{\complement}_{\text{OPTb}}} A_j + KM =: B_3'.$$

On the other hand, RED/LED downloads S_j at most $\frac{A_j}{2M}$ times. S_j then at most incurs $1.5A_j$ units of cost. Adding the costs of downloading and forwarding for all S_i, we have a bound on the cost of RED/LED in the frame, $C_{\text{RL}}(g)$:

$$C_{\text{RL}}(g) \leq \sum_{i \in \mathcal{S}_{\text{OPTb}}} \left(E_i M + \sum_z f_{i,z} \right) + \sum_{j \in \mathcal{S}^{\complement}_{\text{OPTb}}} 1.5A_j \tag{2.12}$$

We are now ready to prove Theorem 4.

Proof of Theorem 4.

$$\begin{aligned} C_{\text{RL}}(g) &\leq \sum_{i \in \mathcal{S}_{\text{OPTb}}} \left(E_i M + \sum_z f_{i,z} \right) + \sum_{j \in \mathcal{S}^{\complement}_{\text{OPTb}}} 1.5A_j \\ &\leq K \left[\max_{i \in \mathcal{S}_{\text{OPTb}}} \left(E_i M + \sum_z f_{i,z} \right) \right] + \sum_{j \in \mathcal{S}^{\complement}_{\text{OPTb}}} 1.5A_j \\ &\leq K \left[\max_{i \in \mathcal{S}_{\text{OPTb}}} \left(E_i M + \sum_z f_{i,z} \right) \right] + \sum_{j \in \mathcal{S}^{\complement}_{\text{OPTb}}} 6.5A_j \\ &= B'_1 + 2.5B'_2 + 6.5B'_3 \leq 10C_{\text{OPTb}}(g), \end{aligned}$$

for every frame. □

Theorem 4 demonstrates that RED/LED is indeed scalable, in the sense that it only needs doubled capacity to achieve a constant competitive ratio.

2.7.3 Implementation of OPTb

OPTb allows polynomial time implementation by dynamic programming. Note that once we know when OPTb makes the downloads, each download cost is a constant KM, and OPTb will download the top K popular services between two consecutive downloads to minimize the forward cost, which is then equal to the number of requests from those non-top-K services. Specifically, let $C(m)$ be the minimum total cost with batch download for a sequence of m requests. We set $C(0) = 0$. Suppose the last download occurs after the n-th arrival, then the total cost $C(m)$ is

$C(m) = C(n) + KM + f(n, m)$, where $f(n, m)$ is the number of requests from the non-top-K services during $[n+1, m]$. If there is no download at all, then $C(m) = \sum_{l=1}^{m} \mathbb{1}\{r_l \notin \{S_1, \ldots, S_K\}\}$ is the number of requests whose services are not cached in the beginning. Therefore, the dynamic programming recursion is:

$$C(m) = \min \begin{cases} \min_{0 \le n < m} C(n) + KM + f(n, m), \\ \sum_{l=1}^{m} \mathbb{1}\{r_l \notin \{S_1, \ldots, S_K\}\}. \end{cases}$$

For a sequence of N requests, it is easy to check the time complexity is at most $O(N^2(N + |\mathbb{S}| \log |\mathbb{S}|))$, where $|\mathbb{S}|$ is the total number of services in the system.

2.8 Practical Issues

2.8.1 Extensions to Heterogeneous Systems

Our analysis so far has assumed that all services are homogeneous. In practice, however, some services are very sensitive to delays, while others are not. Different services also require different amounts of edge-server resources and incur different download costs. We now discuss how to address these heterogeneous features.

We model the heterogeneous features as follows: Forwarding a request for S_i to the back-end cloud incurs a cost of F_i, and downloading the service S_i to the edge server incurs a cost of M_i, with $M_i \ge F_i > 0$. Each service S_i requires $W_i > 0$ units of edge-server resources, and the edge server only has K units of resources. Therefore, if the edge server caches a subset $\mathbb{T}$ of services, we then have $\sum_{i:S_i \in \mathbb{T}} W_i \le K$. Our previous analysis corresponds to the special case where $F_i \equiv 1$, $M_i \equiv M$, and $W_i \equiv 1$.

We have the following theorem for OPT in heterogeneous systems:

Theorem 5. *Suppose we are given a sequence $r_1, r_2, \ldots$, and OPT(n), which is the subset of services cached by OPT after the n-th arrival. If there exists an integer $m > n$, a service $S_i \notin$*

OPT(n)*, and another service* $S_j \in$ *OPT*(n) *with* $W_j \geq W_i$ *such that:*

$$\sum_{l=n+1}^{m} F_i x_i(l) \geq \sum_{l=n+1}^{m} F_j x_j(l) + M_i + M_j, \tag{2.13}$$

then OPT downloads at least one service during $[n+1, m]$.

Proof. The proof is virtually the same as that of Theorem 1. □

With the intuition provided by Theorem 5, we can modify RED and LED as follows:

Definition 7. *When a request* $r_n = S_i \notin$ *RL*$(n-1)$ *arrives,* RED *downloads and replicates* S_i *at the edge server if there exists an integer* τ *and a service* S_j *such that for all* $n - \tau \leq l \leq n-1$, $S_j \in$ *RL*(l) *and* $S_i \notin$ *RL*(l)*, and*

$$\sum_{l=n-\tau}^{n} F_i x_i(l) \geq \sum_{l=n-\tau}^{n} F_j x_j(l) + M_i + M_j. \tag{2.14}$$

Definition 8. *Suppose the edge server decides to download a service at the* n*-th arrival. For each service* S_i *currently cached by the edge server, let* τ_i *be the smallest integer such that* $\sum_{l=n-\tau_i}^{n} F_i x_i(l) \geq 2M_i$. LED *sorts all cached services in descending order of* τ_i*, and deletes services in this order until there are enough resources to accommodate the new service.*

2.8.2 Implementation and Complexity

This subsection discusses the implementation and complexity of RED/LED. We focus on the homogeneous system to simplify the notation. However, it is straightforward to extend the implementation for heterogeneous systems.

We first discuss the implementation of the retrospective download (RED) policy. For all $S_i \in$ RL$(n-1)$ and $S_j \notin$ RL$(n-1)$, let $\mathcal{S}_\tau := \{\tau \in \mathbb{Z}^+ \mid S_i \in \text{RL}(l), S_j \notin \text{RL}(l), \forall l \in [n-\tau, n-1]\}$, and define

$$b_{ij}(n) := \left[\max_{\tau \in \mathcal{S}_\tau} \sum_{l=n-\tau}^{n} x_j(l) - \sum_{l=n-\tau}^{n} x_i(l) \right]^+.$$

By the definition of RED, a service S_j will be downloaded at the n-th arrival if $b_{ij}(n) \geq 2M$, for some $S_i \in \text{RL}(n-1)$. Finding the value of $b_{ij}(n)$ can be transformed into the well-known maximum subarray problem as follows: Construct a sequence of integers $\{a_n\}$ such that $a_n = 1$ if $x_j(n) = 1$, $a_n = -1$ if $x_i(n) = 1$, and $a_n = 0$, otherwise. We then have

$$b_{ij}(n) = \left[\max_{\tau \in \mathcal{S}_\tau} \sum_{l=n-\tau}^{n} a_l \right]^+,$$

which can be computed easily as follows: If service S_i is downloaded at the n-th arrival, set $b_{ij}(n) = 0$, for all j. Otherwise, $b_{ij}(n) = [b_{ij}(n-1) + x_j(n) - x_i(n)]^+$.

Next, we discuss the implementation of LED. When RED/LED decides to download a service, LED needs to compute τ_i for all services S_i cached by RED/LED, where τ_i is chosen so that there are $2M$ requests for S_i in the last τ_i requests. In order to obtain τ_i, each service can maintain the arrival times of its last $2M$ requests, which can be easily done by using a queue of size $2M$ to store the arrival times of past requests.

It is straightforward to extend the above discussions for heterogeneous systems. The complete pseudocode of RED/LED for heterogeneous systems is shown in Algorithm 1. It is easy to check that the time complexity of RED/LED is $O(|\mathbb{S}|)$ per request for homogeneous systems and $O(K|\mathbb{S}|)$ per request for heterogeneous systems, where $|\mathbb{S}|$ is the total number of services in the system. The space complexity is $O(|\mathbb{S}|^2)$. Even when the number of unique services is as large as 10^4, the memory footprint of RED/LED is only about $400\,\text{MB}$, which is easily manageable for edge servers.

2.9 Trace-Based Simulations

In this section, we compare the performance of RED/LED against OPTb, three other offline policies, and one online policy, all using real-world data traces.

Algorithm 1 RED/LED for Heterogeneous Systems

$b_{ij} \leftarrow 0, \forall i, j$
$n \leftarrow 0$
Initialize a queue, q_i, with $\lceil \frac{2M_i}{F_i} \rceil$ elements of 0, $\forall i$
while a new request arrives **do**
    Suppose the request is for service i^*
    $n \leftarrow n + 1$
    Push n into q_{i^*}, and pop out an element
    if $i^* \in \text{RL}(n-1)$ **then**
        Serve this request at the edge server
        for $j \notin \text{RL}(n-1)$ **do**
            $b_{i^*j} \leftarrow (b_{i^*j} - F_{i^*})^+$
    else
        $c \leftarrow$ False // whether to cache i^*
        for $j \in \text{RL}(n-1)$ **do**
            $b_{ji^*} \leftarrow b_{ji^*} + F_{i^*}$
            if $b_{ji^*} \geq M_j + M_{i^*}$ and $W_{i^*} \leq K$ **then**
                $c \leftarrow$ True
        if c **then**
            $\tau_j \leftarrow n - (\text{head of } q_j), \forall j \in \text{RL}(n-1)$
            repeat
                Delete j in descending order of τ_j
            until there are enough resources for i^*
            Download i^*
            $b_{i^*j} \leftarrow 0, \forall j$
            for j that has just been deleted **do**
                $b_{ij} \leftarrow 0, \forall i$
        else
            Forward this request to the back-end cloud

2.9.1 Overview of the Trace

We use a data set from a Google cluster [15] to obtain the sequences of service requests.[6] This data set registers more than three million request arrivals over seven hours, and the average inter-arrival time is about $7\,\text{ms}$. Each request has a "ParentID" that identifies its service. In this data set, there are 9,218 unique services. The most popular service has 208,306 requests, while 5,150 services each only have one request. In all the simulations, we partition the data set into ten

[6] We are not aware of a public data set from deployed edge servers yet.

non-overlapping parts, and run policies on these ten parts individually. We then report the average performance of these ten parts.

2.9.2 Baseline Policies

In addition to RED/LED and OPTb, we also implement three different offline policies and one online policy. The three offline policies are derived from optimal solutions based on data caching, stochastic optimization, and dynamic programming, respectively. We describe the implementations of these policies for both homogeneous systems and heterogeneous systems.

Belady Modified. Belady's algorithm is known to be the optimal offline solution to the data caching problem in computer architecture. It minimizes cache misses by swapping in a new item and deleting the item which will be used in the furthest future. To adopt it in the service caching scenario, we make a small modification: Instead of always downloading a service that is not cached by the edge server, we forward the request when the next occurrence of the requested service is further in the future than those of existing edge services.[7] With this modification, our version of Belady's algorithm takes advantage of the possibility of forwarding without downloading. Belady Modified then achieves the optimal performance in the special case where $M = 1$ for homogeneous systems. For heterogeneous systems, Belady Modified keeps deleting services whose next request is furthest in the future until it has enough resources to accommodate the new service.

Offline Static. In homogeneous systems, Offline Static computes the frequency of all service requests and simply chooses the top K popular services to cache at the edge server. In heterogeneous systems, Offline Static caches a subset of services so that their total resource usage is no more than K, and the sum of their frequencies is maximized, which is a knapsack problem. When the arrivals of requests follow an i.i.d. stochastic process, most online policies that employ stochastic optimization will converge to Offline Static.

Offline Iterative. Given the complete trace, it is possible to compute the optimal solution, OPT, using dynamic programming. However, even for the homogeneous system, the complexity of dynamic programming is at least $O(\binom{|\mathbb{S}|}{K})$ per request. Even when K is as small as 5, our

[7] We say a service is an edge service if it is cached by the edge server.

implementation finds that dynamic programming cannot be completed within a reasonable amount of time. Therefore, we instead implement the following iterative policy for homogeneous systems: Since the edge server can cache K services, we say that the edge server has K slots, numbered as $L_1, L_2, \ldots, L_K$, and each of them can cache one service. Offline Iterative algorithm finds the edge service at each of the K slots iteratively. First, it uses dynamic programming to find services cached in L_1 so as to minimize the total cost assuming the capacity is one. Given the solutions for $L_1, L_2 \ldots, L_k$, the policy then uses dynamic programming to find the services cached in L_{k+1} so that the total cost is minimized assuming the capacity is $k+1$, and $L_1, \ldots, L_k$ are given. This policy achieves the optimal performance when $K = 1$. We only test this policy for homogeneous systems since it cannot be easily extended for heterogeneous systems.

Online Randomized. We consider an online baseline policy in addition to the above offline policies. Intuitively, a reasonable policy should download more often when the download cost is small. When a request whose service is not an edge service arrives, Online Randomized downloads the new service with probability $\frac{1}{M}$, or with probability $\frac{F_i}{M_i}$ in heterogeneous systems. To accommodate the new service, Online Randomized keeps deleting edge services uniformly at random until there are enough resources available. As Online Randomized is a randomized policy, we report its average performance over 10 i.i.d. simulation runs on each part of the data set.[8]

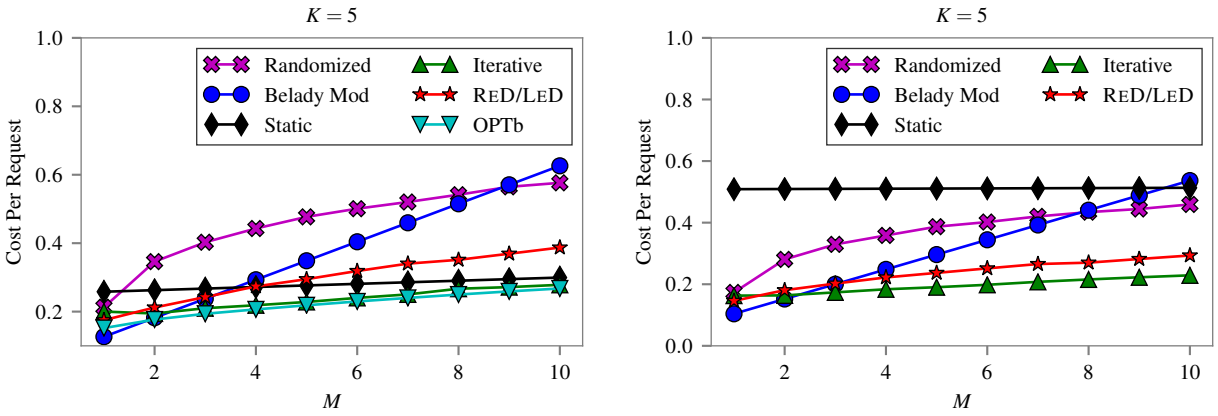
2.9.3 Performance Evaluations for Homogeneous Systems

We implement all the above policies and run the algorithms with different K and M for homogeneous systems. For each pair of K and M, we calculate the total costs over 10^3 requests and over 10^4 requests. The costs, normalized by the number of requests, are compared in Fig. 2.6 and Fig. 2.7. Due to excessive running time, OPTb is skipped for the case of 10^4 requests.[9]

Fig. 2.6 compares the costs of the above algorithms while fixing $K = 5$. RED/LED performs very well when compared with other policies. In most settings, OPTb and Offline Iterative are slightly better than RED/LED, but the difference is very limited. We note that OPTb and Offline

[8]The average performance over more runs remains the same.

[9]Over seven hours are required for a single sequence in our simulation.



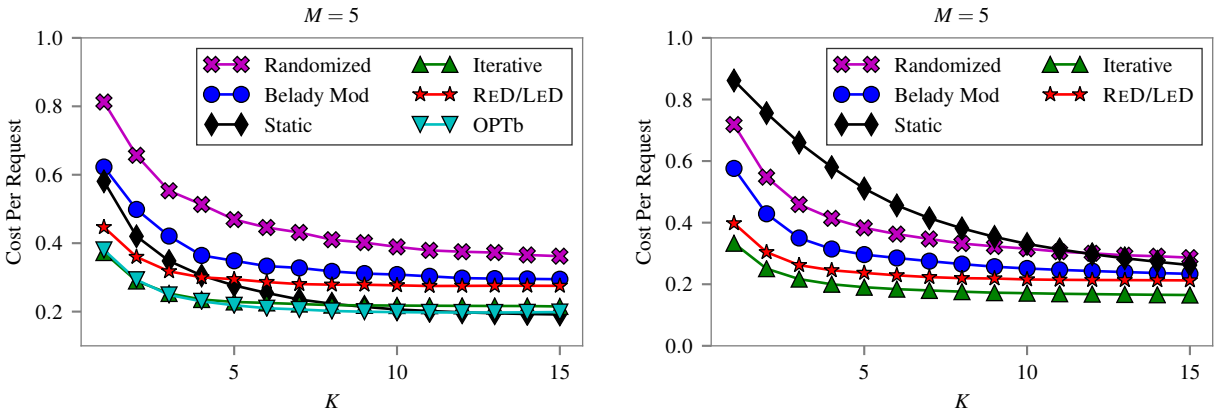
(a) Total cost over 10^3 requests.

(b) Total cost over 10^4 requests.

Figure 2.6: Cost comparison with different download costs M. Reprinted with permission from [33].

Iterative are very intelligent policies that require the knowledge of all future arrivals and have very high complexity. The result that our policy, being an online policy with low complexity, is only slightly worse than Offline Iterative suggests that it works well in practice. Also note that RED/LED has much better "real-world" performance than that the theoretical result guarantees since the competitive ratio is based on a worst-case analysis. Belady Modified achieves better performance than RED/LED when $M = 1$, as it is indeed the optimal policy, OPT, in such special case. However, as M becomes larger, it quickly becomes much worse than RED/LED. This highlights the difference of the service caching problem from the data caching problem. Offline Static can be better than RED/LED when we only evaluate it over 10^3 requests, but has worse performance than RED/LED when we evaluate it over 10^4 requests. With more requests, the system witnesses more variations, and therefore Offline Static becomes worse. Finally, Online Randomized performs poorly in all settings.

Fig. 2.7 shows the costs with different K with $M = 5$. Similar to Fig. 2.6, OPTb and Offline Iterative are only slightly better than RED/LED in all settings. Note that OPTb and Offline Iterative both minimize the total cost when $K = 1$. This result therefore shows that our RED/LED is close to OPT. Offline Static performs worse than our policy when we evaluate it over 10^4 requests. Both



(a) Total cost over 10^3 requests.

(b) Total cost over 10^4 requests.

Figure 2.7: Cost comparison with different K. Reprinted with permission from [33].

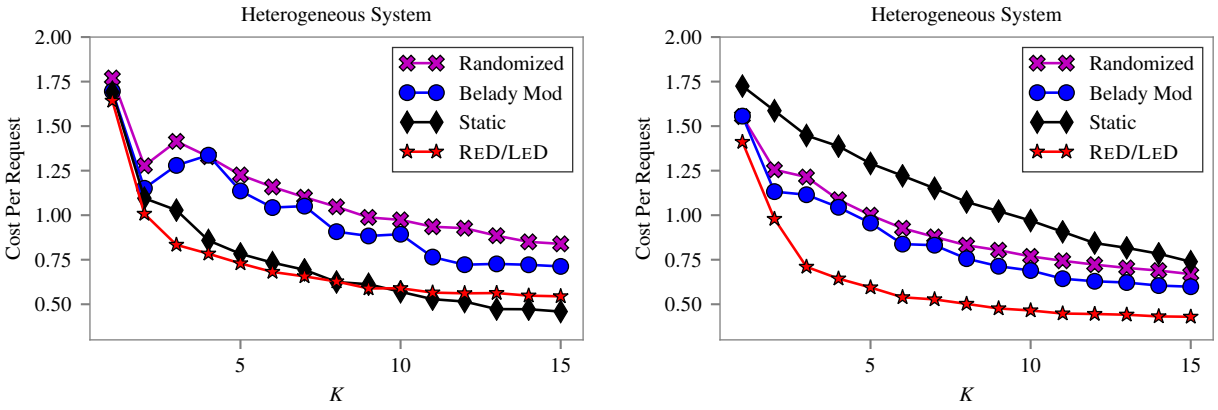
Belady Modified and Online Randomized are worse than RED/LED under all settings. Fig. 2.7a also exhibits the good performance of RED/LED with double capacity if we compare, for instance, the cost of RED/LED when $K = 10$ and the cost of OPTb when $K = 5$.

In addition, we also note that OPTb and Offline Iterative have very similar performance in all settings. This suggests that OPTb may be close to optimal in real-world scenarios.

2.9.4 Performance Evaluations for Heterogeneous Systems

We further evaluate the performance for heterogeneous systems. To create heterogeneity, we assign different forward costs $F_i = 1, 2, 3$, download costs $M_i = 5, 10, 15$, and resource requirements $W_i = 1, 2, 3$ to different services by their IDs. We again run the algorithms over 10^3 requests and over 10^4 requests, and calculate their total costs.

The normalized costs of different policies with different edge-server capacities K are shown in Fig. 2.8. We can see that RED/LED achieves the minimum cost among all policies under most settings. Although we only establish the competitiveness of RED/LED for homogeneous systems, this result suggests that RED/LED remains a desirable solution even in heterogeneous systems. Note that generally the curves are not smooth since K takes discrete values, and the non-monotonicity is due to specific request sequences, heterogeneity of costs and resource requirements, and download



(a) Total cost over 10^3 requests.

(b) Total cost over 10^4 requests.

Figure 2.8: Cost comparison in heterogeneous systems. Reprinted with permission from [33].

policies.

2.10 Conclusions

This chapter studies online algorithms for dynamic service caching at the edge. We introduce a model that captures the limited capacity of edge servers, the unknown arrival patterns of requests, and the operational costs of edge servers, including the cost of forwarding requests to the back-end cloud and the cost of downloading new services. We propose an online policy, RED/LED, that has a small total cost under any arbitrary sequence of arrivals. We evaluate the competitive ratio of RED/LED, and prove that it is at most $10K$, where K is the capacity of the edge server. Moreover, we prove that the competitive ratio of any deterministic online policy is at least $\Theta(K)$, and therefore RED/LED is asymptotically optimal with respect to K. In addition, we show that our policy is $(2, 10)$-OPTb-competitive and thus scalable. Furthermore, RED/LED can be easily extended to heterogeneous systems and maintains a low time complexity. The performance of RED/LED is further evaluated through simulations using traces from real-world data centers. Simulation results demonstrate that our RED/LED achieves better, or similar, performance compared to many intelligent policies in all scenarios.

There are several important directions for future research: First, RED/LED is only asymptoti-

cally optimal for homogeneous systems. It is of interest to study the optimal online algorithm for heterogeneous systems. Second, RED/LED is only proved to be asymptotically optimal among deterministic online policies, and it remains an open question whether RED/LED is still optimal when randomized policies are taken into account. Third, the scalability result of RED/LED is established by comparing it against OPTb, which is a more constrained policy than the optimal offline policy, OPT. Studying the scalability of online algorithms when compared against OPT can be important future work.

3. CONTENT CACHING AT THE EDGE*

Information Centric Edge is promising to provide better video streaming services to mobile users by provisioning computing and storage resources at the edge of the network for video contents, and forwarding packets directly based on video contents. However, due to the diversity of user interests, user devices, video versions or resolutions, cache sizes, network conditions, etc., it is challenging to decide where to place the video contents, and which cache and video version a mobile user device should select. In this chapter, we study the joint optimization of cache-version selection and content placement for adaptive video streaming in Information Centric Edge. We propose practical distributed algorithms that operate at each user device and each network cache to maximize the overall network utility. In addition to proving that our algorithms indeed achieve the optimal performance, we implement our algorithms as well as several baseline algorithms on ndnSIM, an ns-3 based Named Data Networking simulator. Simulation evaluations demonstrate that our algorithms significantly outperform conventional heuristic solutions.

3.1 Introduction

Video streaming has become the dominant application for modern Internet traffic. In order to provide better quality of service (QoS) and quality of experience (QoE) to mobile users, content delivery networks (CDNs) have been deployed to store popular videos at cache servers close to the users. This aligns with the trend of Information Centric Edge, where computing and storage resources are provisioned at the edge of the wireless network [25]. Meanwhile, as users are accessing videos from a variety of devices, ranging from smartphones to 4K televisions (TVs), adaptive video streaming, which encodes the same video content into multiple versions with different resolutions, has been widely used to deliver arguably the best video version to each user based on device types and network conditions.

*Part of this chapter is reprinted with permission from "Cache-version selection and content placement for adaptive video streaming in wireless edge networks" by A. Sasikumar, T. Zhao, I.-H. Hou, and S. Shakkottai, in *2019 17th Int. Symp. Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt '19)*, Copyright 2019 IFIP

In this chapter, we study the interplay between three important components for adaptive video streaming in Information Centric Edge: *cache selection*, where each user device determines which cache server to retrieve videos from, *version selection*, which determines the version that each user watches, and *content placement*, which entails the caching strategy of each cache server. We formulate CaVe-CoP, a Cache-Version selection and Content Placement problem that jointly optimizes these three components by taking into account the preferred video versions of users, the communication capacities of network links, and the storage capacities of cache servers. Our goal is to develop a new network algorithm for CaVe-CoP that is not only provably optimal, but also practical and implementable.

Our proposed solution is based on the observation that there is a practical timescale separation between cache-version selection (CaVe) and content placement (CoP), as the former can be updated much more frequently. Hence, we first solve the CaVe problem by fixing the solution to the CoP problem, and prove the optimality of our CaVe algorithms. We then solve the CoP problem by considering its influence to solution to the CaVe problem, and prove our CoP algorithms are optimal when fractional solutions are allowed.

While our algorithms can be practically implemented under the current Internet architecture with TCP/IP, we demonstrate that our algorithms can also be implemented in a distributed fashion on Named Data Networking (NDN) [34], a future Internet architecture designed with video streaming applications in mind. Since NDN forwards packets by content names instead of location IDs such as IP addresses, we present a distributed forwarding strategy that ensures user devices always obtain their selected video versions from their selected cache server. Moreover, we show that the overhead of our algorithms is negligible by exploiting local information and built-in caching. We evaluate our algorithms on ndnSIM [35], an ns-3 based NDN simulator. Simulation results depict that our algorithms significantly outperform baseline policies that employ conventional heuristic solutions and subsets of our algorithms.

The rest of the chapter is organized as follows. Section 3.2 introduces our system model and the formulation of CaVe-CoP. Solutions to the two problems CaVe and CoP are introduced in Sec-

tion 3.3 and 3.4, respectively. In Section 3.5, we discuss the implementation of our algorithms in NDN. Section 3.6 demonstrates the simulation results. Section 3.7 reviews some related literature. Finally, Section 3.8 concludes the chapter.

3.2 System Model

We consider an Information Centric Edge network where a group of network caches jointly host a set of videos and serve a set of video streaming users.[1] Fig. 3.1 illustrates an example of such a network, which is consistent with the YouTube video delivery system [36]. We use $\mathbb{C}$ to denote the set of network caches, $\mathbb{S}$ to denote the set of users, and $\mathbb{L}$ to denote the set of communication links that connect the network caches, routers, and users. We assume there is a route[2] between each user s and each network cache c, and define $H^l_{s,c}$ as the indicator function that link l is on the route between s and c.

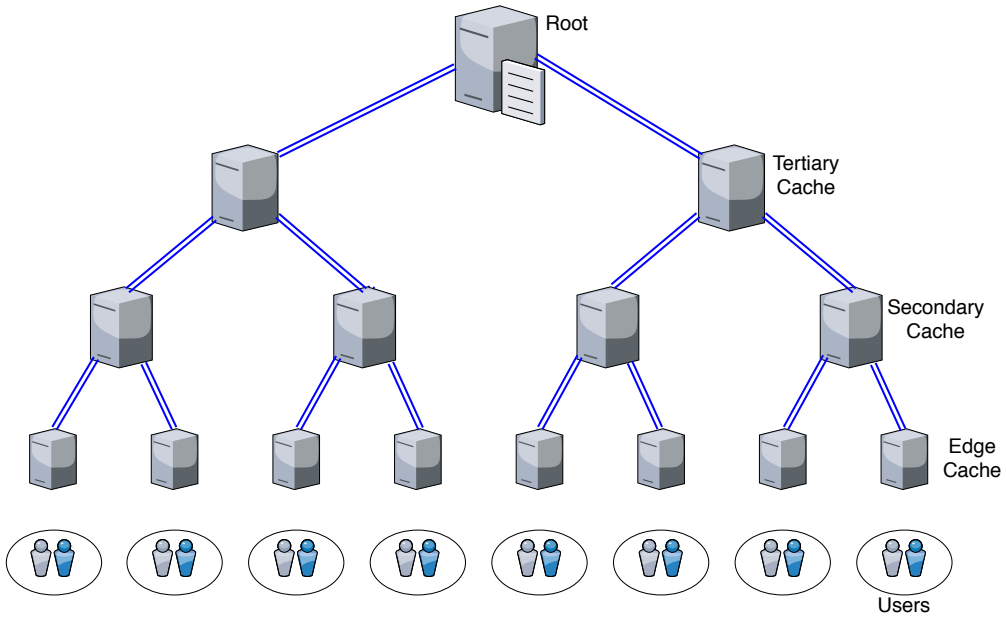


Figure 3.1: An Information Centric Edge network. The root node holds all videos and there are three layers of caches. Each edge cache serves a group of users with different user devices. Reprinted with permission from [37].

[1] The terms "user" and "user device" are used interchangeably.
[2] Our model and algorithms can be generalized to the multi-route scenario.

We consider multi-version video streaming where each video is encoded into multiple versions for different resolutions of the same video content. We use $\mathbb{V}$ to denote the set of all versions of all videos. For each video version $v \in \mathbb{V}$, we use X_v to denote the average bit rate of v and Y_v to denote the file size of v, i.e. the product of X_v and the duration of the video. For the ease of theoretical analysis, we assume that there exists a *null version* v_0 with $X_{v_0} = 0$ and $Y_{v_0} = 0$. If a user decides not to watch any video, then we say that the user watches the null version v_0. With the introduction of the null version, we can assume that each user always watches a video version.

Each network cache $c \in \mathbb{C}$ has a storage of size B_c to store some video versions. Specifically, let $p_{c,v}$ be the indicator function that v is present in the storage of c, then we have $\sum_v Y_v p_{c,v} \leq B_c$, for all c. Each network cache c determines which video versions to store, and thereby determines the values of $p_{c,v}$, subject to its storage constraint. We assume that there exists at least a network cache c with infinite storage $B_c = \infty$ and stores all video versions, which we call the root node. Such an assumption is to ensure that at least one copy of each video version exists in the network. We refer to the problem of determining $p_{c,v}$ as the *content placement (CoP) problem*.

At the user end, each user s is interested in watching a video. Let $\mathbb{I}_s$ be the set of video versions that correspond to the interested video of user s. Each user device s needs to determine which video version to watch, as well as which network cache to obtain the video version from. Let $z_{s,c,v}$ be the indicator function that user s decides to watch video version v, and to obtain it from network cache c. We refer to the problem of determining $z_{s,c,v}$ as the *cache-version selection (CaVe) problem*. Since user s needs to obtain exactly one video version, we require that $\sum_{c,v \in \mathbb{I}_s} z_{s,c,v} = 1$, for all s. Moreover, user s can only obtain video version v from network cache c if c indeed stores v, that is, $p_{c,v} = 1$. Hence, we also need $z_{s,c,v} \leq p_{c,v}$, for all s, c, v.

Recall that the bit rate of video version v is X_v and $H^l_{s,c} = 1$ if link l is on the route between s and c. When user s obtains v from c, it incurs an amount of X_v traffic on each link along the route between s and c. The total amount of traffic on link l can then be expressed as $\sum_{s,c,v} X_v H^l_{s,c} z_{s,c,v}$. We consider that each link l has a finite capacity of R_l, and hence we require that $\sum_{s,c,v} X_v H^l_{s,c} z_{s,c,v} \leq R_l$, for all $l \in \mathbb{L}$.

Finally, each user obtains some utility based on its perceived video quality. In particular, we consider that each user s obtains a utility of $U_s(X_v)$ when watching a video version with bit rate X_v. We assume that $U_s(\cdot)$ is a non-decreasing and concave function. Different users may have different utility functions since they may be watching videos on different types of devices. For example, users watching videos on smartphones typically enjoy lower utility than those watching videos on TVs.

We aim to maximize the total utility of all users in the network by choosing the optimal $\boldsymbol{p} := [p_{c,v}]$ and $\boldsymbol{z} := [z_{s,c,v}]$, subject to all aforementioned constraints. Formally, we have the following CaVe-CoP optimization problem.[3]

CaVe-CoP

$$\text{maximize} \quad \sum_{s,c,v \in \mathbb{I}_s} U_s(X_v) z_{s,c,v} \tag{3.1a}$$

$$\text{subject to} \quad \sum_{v} Y_v p_{c,v} \leq B_c, \qquad \forall c \in \mathbb{C}, \tag{3.1b}$$

$$\sum_{c,v \in \mathbb{I}_s} z_{s,c,v} = 1, \qquad \forall s \in \mathbb{S}, \tag{3.1c}$$

$$z_{s,c,v} \leq p_{c,v}, \qquad \forall s \in \mathbb{S}, c \in \mathbb{C}, v \in \mathbb{V}, \tag{3.1d}$$

$$\sum_{s,c,v} X_v H^l_{s,c} z_{s,c,v} \leq R_l, \qquad \forall l \in \mathbb{L}, \tag{3.1e}$$

$$p_{c,v} \in \{0,1\}, z_{s,c,v} \in \{0,1\}, \quad \forall s \in \mathbb{S}, c \in \mathbb{C}, v \in \mathbb{V}. \tag{3.1f}$$

While the utility maximization problem studied in this chapter may look similar to many existing studies on network utility maximization (NUM), we note that there are two major challenges that distinguish our problem from other NUM problems: First, most existing studies on NUM problems assume that the source and destination of each flow is fixed and given. In contrast, multiple network caches may store the same video version depending on the solution to the content placement problem. Hence, not only does a user have multiple choices of network caches to ob-

[3] In practice the CaVe-CoP problem will be solved repeatedly over time with different parameters to cope with network changes.

tain the video version from, but the problem of selecting cache is fundamentally intertwined with the problem of content placement. Second, although the problem of version selection may seem to be a special case of the rate control problem, we note that the problem of version selection is fundamentally intertwined with the problem of selecting cache since each cache may only store a subset of versions for a given video. The possibility of placing different versions of the same video at different caches also distinguishes this work from some recent studies on throughput-optimal algorithms with caches. Araldo *et al.* [38] studied a similar problem to ours. However, they only derived heuristics without meaningful performance guarantees.

The decision variables in CaVe-CoP are $\boldsymbol{p}$ and $\boldsymbol{z}$. We note that there is a practical timescale separation between the update for $\boldsymbol{p}$ and that for $\boldsymbol{z}$. When a user device changes its values for $\boldsymbol{z}$ due to e.g. network congestion, it simply requests new packets from a different network cache and/or with a different video version. Hence, $\boldsymbol{z}$ can be updated rather frequently, for example, once every 100 milliseconds. On the other hand, when a network cache changes its values for $p_{c,v}$, it needs to obtain all video versions with $p_{c,v} = 1$. Hence, $\boldsymbol{p}$ can only be updated infrequently.

Our proposed solution for CaVe-CoP is based on the observation of the timescale separation between the update for $\boldsymbol{p}$ and that for $\boldsymbol{z}$. In Section 3.3, we will first consider the CaVe problem by finding the optimal $\boldsymbol{z}$ for given $\boldsymbol{p}$. Next, in Section 3.4, we will consider the CoP problem. In order to find the optimal $\boldsymbol{p}$, we will introduce pseudo-variables $\boldsymbol{z}' := [z'_{s,c,v}]$ and $\boldsymbol{p}' := [p'_{c,v}]$ that are updated at the same frequency as $\boldsymbol{p}$ to address the issue with timescale separation.

Finally, we note that CaVe-CoP is an integer programming problem since $p_{c,v}$ and $z_{s,c,v}$ are integers. To obtain tractable results, we will relax (3.1f) and allow $p_{c,v}$ and $z_{s,c,v}$ to be any real number between 0 and 1. As we will demonstrate in Section 3.3, our solution to the CaVe problem will always yield integer values for $z_{s,c,v}$. We will also discuss how to obtain integer solutions for $p_{c,v}$ in Section 3.4.

3.3 The Cache-Version Selection Problem (CaVe)

In this section, we study the CaVe problem. We consider that the contents that each network cache store are given and fixed, and aims to determine both the video version to watch and the

network cache to obtain contents from for each user. In terms of the optimization problem (3.1a)–(3.1f), we focus on finding the optimal $\boldsymbol{z} := [z_{s,c,v}]$ to maximize total utility in the network when $\boldsymbol{p} := [p_{c,v}]$ is given and fixed.

3.3.1 Overview of the Solution

We begin by rewriting the optimization problem (3.1a)–(3.1f) for the CaVe problem. Since $\boldsymbol{p}$ is given and fixed, constraint (3.1b) no longer applies. Further, we relax the constraint (3.1f) by allowing $z_{s,c,v}$ to be any real number between 0 and 1. The resulting optimization problem, which we call CaVe-Primal, can then be described as follows:

CaVe-Primal

$$\text{maximize} \quad \sum_{s,c,v \in \mathbb{I}_s} U_s(X_v) z_{s,c,v} \tag{3.2a}$$

$$\text{subject to} \quad \sum_{c,v \in \mathbb{I}_s} z_{s,c,v} = 1, \quad \forall s \in \mathbb{S}, \tag{3.2b}$$

$$z_{s,c,v} \leq p_{c,v}, \forall s \in \mathbb{S}, c \in \mathbb{C}, v \in \mathbb{V}, \tag{3.2c}$$

$$\sum_{s,c,v} X_v H^l_{s,c} z_{s,c,v} \leq R_l, \ \ \forall l \in \mathbb{L}, \tag{3.2d}$$

$$0 \leq z_{s,c,v} \leq 1, \quad \forall s \in \mathbb{S}, c \in \mathbb{C}, v \in \mathbb{V}. \tag{3.2e}$$

We will consider a dual problem to CaVe-Primal. We associate a Lagrange multiplier, λ_l, for each link capacity constraint (3.2d), for all $l \in \mathbb{L}$. Let $\boldsymbol{\lambda} := [\lambda_l]$ be the vector of Lagrange multipliers. The Lagrangian is obtained as follows:

$$\begin{aligned} & L(\boldsymbol{z}, \boldsymbol{\lambda}) \\ & := \sum_{s,c,v \in \mathbb{I}_s} U_s(X_v) z_{s,c,v} - \sum_l \lambda_l \left(\sum_{s,c,v} z_{s,c,v} H^l_{s,c} X_v - R_l \right) \end{aligned} \tag{3.3}$$

The dual objective, $D(\boldsymbol{\lambda})$, is defined as the maximum value of $L(\boldsymbol{z}, \boldsymbol{\lambda})$ over $\boldsymbol{z}$ subject to the constraints (3.2b), (3.2c), and (3.2e). We call the underlying optimization problem CaVe-

Lagrangian. It can be written as follows:

CaVe-Lagrangian

$$\text{maximize} \quad L(\boldsymbol{z}, \boldsymbol{\lambda}) \tag{3.4a}$$

$$\text{subject to} \quad \sum_{c,v \in \mathbb{I}_s} z_{s,c,v} = 1, \qquad \forall s \in \mathbb{S}, \tag{3.4b}$$

$$z_{s,c,v} \leq p_{c,v}, \quad \forall s \in \mathbb{S}, c \in \mathbb{C}, v \in \mathbb{V}, \tag{3.4c}$$

$$0 \leq z_{s,c,v} \leq 1, \qquad \forall s \in \mathbb{S}, c \in \mathbb{C}, v \in \mathbb{V}. \tag{3.4d}$$

Remark 1. *In defining the CaVe-Lagrangian problem, we only relax the link capacity constraint (3.2d), and keep other constraints (3.2b), (3.2c) and (3.2e) intact. This is because the link capacity constraint (3.2d) can be temporarily violated as packets that cannot be served immediately can be queued in the buffer. On the other hand, constraints (3.2b) and (3.2c) need to be satisfied at all time in practical systems.*

The dual problem is to minimize $D(\boldsymbol{\lambda})$ while ensuring that all Lagrange multipliers λ_l are non-negative. We call this the CaVe-Dual and mathematically write it as:

CaVe-Dual

$$\text{minimize} \quad D(\boldsymbol{\lambda}) \tag{3.5a}$$

$$\text{subject to} \quad \lambda_l \geq 0, \quad \forall \lambda_l \in \mathbb{L}. \tag{3.5b}$$

Theorem 6 (Strong Duality)**.** *CaVe-Primal and CaVe-Dual have the same optimal value.*

Proof. The objective function of CaVe-Primal is a linear function of $\boldsymbol{z}$, and hence is concave. The set of $\boldsymbol{z}$ that satisfies the three unrelaxed constraints, namely, (3.2b), (3.2c), and (3.2e), is nonempty and convex.

Furthermore, the relaxed constraint (3.2d) is linear and thus convex. To get strict inequalities in (3.2d), we can set $z_{s,c,v}$ to be 1 if c is the root node and v is the null version, and 0 otherwise.

It is straightforward to see that (3.2b), (3.2c), and (3.2e) are satisfied, while (3.2d) is satisfied with strict inequalities.

Hence, this theorem holds following Theorem 6.2.4 (Strong Duality Theorem) in [39]. □

Based on Theorem 6, we can solve the CaVe-Primal problem by solving CaVe-Dual. Solving CaVe-Dual involves two steps: First, for a given vector $\boldsymbol{\lambda}$, we need to find $D(\boldsymbol{\lambda})$ by solving CaVe-Lagrangian. Second, we need to find the optimal $\boldsymbol{\lambda}$ to solve CaVe-Dual. We introduce our solutions to these two steps below.

3.3.2 The Solution to CaVe-Lagrangian

We rewrite (3.3) as:

$$\begin{aligned} & L(\boldsymbol{z}, \boldsymbol{\lambda}) \\ &= \sum_{s,c,v \in \mathbb{I}_s} U_s(X_v) z_{s,c,v} - \sum_l \lambda_l \left(\sum_{s,c,v} z_{s,c,v} H^l_{s,c} X_v - R_l \right) \\ &= \sum_s \sum_{c,v \in \mathbb{I}_s} z_{s,c,v} \big(U_s(X_v) - X_v \sum_{l: H^l_{s,c}=1} \lambda_l \big) + \sum_l \lambda_l R_l \end{aligned} \tag{3.6}$$

We note that the above expression provides a natural user-by-user decomposition. Specifically, by defining $\boldsymbol{z}_s$ as the vector containing all $[z_{s,c,v}]$ for a given s, and defining

$$L_s(\boldsymbol{z_s}, \boldsymbol{\lambda}) := \sum_{c,v \in \mathbb{I}_s} z_{s,c,v} \big(U_s(X_v) - X_v \sum_{l: H^l_{s,c}=1} \lambda_l \big), \tag{3.7}$$

we have

$$L(\boldsymbol{z}, \boldsymbol{\lambda}) = \sum_s L_s(\boldsymbol{z_s}, \boldsymbol{\lambda}) + \sum_l \lambda_l R_l. \tag{3.8}$$

As $\boldsymbol{\lambda}$ is given in CaVe-Lagrangian, the last term $\sum_l \lambda_l R_l$ is a constant. Hence, $L(\boldsymbol{z}, \boldsymbol{\lambda})$ is maximized if one can maximize $L_s(\boldsymbol{z_s}, \boldsymbol{\lambda})$ for each user s. Moreover, recall that $p_{c,v}$ is the indicator function that network cache c stores video version v. Therefore, the constraint (3.4c) is equivalent to saying that $z_{s,c,v}$ needs to be 0 if $p_{c,v} = 0$. We can now define CaVe-User$_s$ as follows:

CaVe-User$_s$

$$\text{maximize} \quad \sum_{c,v:v\in\mathbb{I}_s,p_{c,v}=1} z_{s,c,v}\big(U_s(X_v) - X_v \sum_{l:H^l_{s,c}=1} \lambda_l\big) \tag{3.9a}$$

$$\text{subject to} \quad \sum_{c,v:v\in\mathbb{I}_s,p_{c,v}=1} z_{s,c,v} = 1, \tag{3.9b}$$

$$0 \le z_{s,c,v} \le 1, \quad \forall c \in \mathbb{C}, v \in \mathbb{V}. \tag{3.9c}$$

It is clear that the optimal vector $\boldsymbol{z}$ that solves CaVe-User$_s$, for all s, is also the optimal vector that solves CaVe-Lagrangian. To solve CaVe-User$_s$, note that the only decision variable in CaVe-User$_s$ is the vector $\boldsymbol{z}_s$, while $U_s(X_v)$, X_v, and λ_l are all constants. Hence, the following algorithm solves CaVe-User$_s$: First, find (c^*, v^*) that has the maximum value of $U_s(X_v) - X_v \sum_{l:H^l_{s,c}=1} \lambda_l$ among all (c, v) with $v \in \mathbb{I}_s$ and $p_{c,v} = 1$. Ties can be broken arbitrarily. Second, set $z_{s,c^*,v^*} = 1$, and $z_{s,c,v} = 0$ for all other (c, v). Alg. 2 summarizes the algorithm. We note that, even though we have relaxed the constraint and allowed $z_{s,c,v}$ to be any real number between 0 and 1, the optimal solution produced by Alg. 2 is always an integer one. Besides, note that c^* and v^* are updated iteratively as $\boldsymbol{\lambda}$ is updated. It means the cache-version selection of each user is dynamic and adaptive to the network congestion.

Algorithm 2 CaVe-User$_s$ Algorithm

Obtain $\boldsymbol{p}$ and $\boldsymbol{\lambda}$
$z_{s,c,v} \leftarrow 0, \forall c, v$
$(c^*, v^*) \leftarrow \text{argmax}_{c,v\in\mathbb{I}_s:p_{c,v}=1} U_s(X_v) - X_v \sum_{l:H^l_{s,c}=1} \lambda_l$
$z_{s,c^*,v^*} \leftarrow 1$

3.3.3 The Solution to CaVe-Dual

Our solution to CaVe-Dual is shown in Alg. 3, where each link l updates its own λ_l. We have the following lemma and theorem.

Algorithm 3 CaVe-Link$_l$ Algorithm

$t \leftarrow 0$, $\lambda_l \leftarrow 0$
while true **do**
  Obtain $\boldsymbol{z}$ from Alg. 2
  $\lambda_l \leftarrow \left[\lambda_l + h_t(\sum_{s,c,v} X_v H^l_{s,c} z_{s,c,v} - R_l)\right]^+$
  $t \leftarrow t + 1$

Lemma 5. *Given $\boldsymbol{\lambda}$, let $\boldsymbol{z}^*$ be the vector that solves CaVe-User$_s$. Then $\boldsymbol{g} := [g_l] := [R_l - \sum_{s,c,v} X_v H^l_{s,c} z^*_{s,c,v}]$ is a subgradient of $D(\boldsymbol{\lambda})$.*

Proof. $\boldsymbol{z}^*$ solves CaVe-User$_s$ and thus solves CaVe-Lagrangian. By definition, $D(\boldsymbol{\lambda}) = L(\boldsymbol{z}^*, \boldsymbol{\lambda})$ for the given $\boldsymbol{\lambda}$. Therefore, for any $\tilde{\boldsymbol{\lambda}} := [\tilde{\lambda}_l]$ where $\tilde{\lambda}_l \geq 0$,

$$\begin{aligned} D(\tilde{\boldsymbol{\lambda}}) - D(\boldsymbol{\lambda}) &\geq L(\boldsymbol{z}^*, \tilde{\boldsymbol{\lambda}}) - L(\boldsymbol{z}^*, \boldsymbol{\lambda}) \\ &= -\sum_l (\tilde{\lambda}_l - \lambda_l) \left(\sum_{s,c,v} z^*_{s,c,v} H^l_{s,c} X_v - R_l \right) \\ &= (\tilde{\boldsymbol{\lambda}} - \boldsymbol{\lambda})^T \boldsymbol{g}. \end{aligned}$$

Hence, $\boldsymbol{g}$ is a subgradient of $D(\boldsymbol{\lambda})$. □

Theorem 7. *Let $\{h_t\}$ be a sequence of non-negative numbers with $\sum_{t=0}^{\infty} h_t = \infty$ and $\lim_{t\to\infty} h_t = 0$, then Alg. 3 solves CaVe-Dual.*

Proof. Note that the objective function of CaVe-Dual is convex in $\boldsymbol{\lambda}$, and the feasible region is a nonempty, convex, closed subset of $\mathbb{R}^{|\mathbb{L}|}$. With Lemma 5 and the step size sequence specified in the theorem, Alg. 3 solves CaVe-Dual following Theorem 8.9.2 in [39]. □

3.3.4 The Solution to CaVe-Primal

We have the following theorem regarding the optimality of $\boldsymbol{z}$ obtained by running Alg. 2 and Alg. 3 iteratively:

Theorem 8. *Let $\boldsymbol{z}^*$ be the vector that solves CaVe-Primal, $\boldsymbol{\lambda}^k$ be the vector produced by Alg. 3 after k iterations, and $\boldsymbol{z}^k$ be the vector produced by Alg. 2 when $\boldsymbol{\lambda} = \boldsymbol{\lambda}^k$. Let $\bar{\boldsymbol{z}}^t$ be the weighted*

average of $\boldsymbol{z}^k$ after the first t iterations, i.e. $\bar{\boldsymbol{z}}^t := \lim_{T\to\infty} \frac{\sum_{k=t+1}^{t+T} h_k \boldsymbol{z}^k}{\sum_{k=t+1}^{t+T} h_k}$. Then, for any $\varepsilon > 0$, there exists an integer K such that for every $t > K$,

1. *$\bar{\boldsymbol{z}}^t$ satisfies all CaVe-Primal constraints;*

2. *$\sum_{s,c,v} U_s(X_v) z^*_{s,c,v} - \sum_{s,c,v} U_s(X_v) \bar{z}^t_{s,c,v} \le \varepsilon$.*

Proof. For 1), first it is straightforward to see that $\boldsymbol{z}^k$ for any k satisfies the constraints (3.2b), (3.2c), and (3.2e), since they are not relaxed when formulating CaVe-Lagrangian and CaVe-User$_s$. Hence, it is easy to show that $\bar{\boldsymbol{z}}^t$ satisfies these three constraints. As for the remaining constraint (3.2d), Theorem 7 shows that for any $\varepsilon > 0$, there exists an integer K such that for every integer $t > K$ and for all l, $|\lambda_l^t - \lambda_l^*| < \varepsilon/2$, where $\boldsymbol{\lambda}^*$ is the optimal point of CaVe-Dual. Hence, for any integer $T > 0$, $\lambda_l^{t+T+1} < \lambda_l^{t+1} + \varepsilon$. Meanwhile, we know from Alg. 3 that $\lambda_l^{k+1} \ge \lambda_l^k + h_k(\sum_{s,c,v} X_v H^l_{s,c} z^k_{s,c,v} - R_l)$. Therefore, $\lambda_l^{t+T+1} \ge \lambda_l^{t+1} + \sum_{k=t+1}^{t+T} h_k(\sum_{s,c,v} X_v H^l_{s,c} z^k_{s,c,v} - R_l)$. So we have

$$\frac{\sum_{k=t+1}^{t+T} h_k(\sum_{s,c,v} X_v H^l_{s,c} z^k_{s,c,v} - R_l)}{\sum_{k=t+1}^{t+T} h_k} < \frac{\varepsilon}{\sum_{k=t+1}^{t+T} h_k}.$$

That is,

$$\sum_{s,c,v} X_v H^l_{s,c} \frac{\sum_{k=t+1}^{t+T} h_k z^k_{s,c,v}}{\sum_{k=t+1}^{t+T} h_k} < R_l + \frac{\varepsilon}{\sum_{k=t+1}^{t+T} h_k}.$$

Let $T \to \infty$, and we have

$$\sum_{s,c,v} X_v H^l_{s,c} \bar{z}^t_{s,c,v} \le R_l,$$

for all l. Hence, $\bar{\boldsymbol{z}}^t$ satisfies the constraint (3.2d), and 1) is proved. Since we also have $\lambda_l^{t+T+1} > \lambda_l^{t+1} - \varepsilon$, we know $\sum_{s,c,v} X_v H^l_{s,c} \bar{z}^t_{s,c,v} \ge R_l$, and thus $\sum_{s,c,v} X_v H^l_{s,c} \bar{z}^t_{s,c,v} = R_l$.

To prove $\sum_{s,c,v} U_s(X_v) z^*_{s,c,v} - \sum_{s,c,v} U_s(X_v) \bar{z}^t_{s,c,v} \le \varepsilon$, we note that based on Theorem 6, $\sum_{s,c,v} U_s(X_v) z^*_{s,c,v} = D(\boldsymbol{\lambda}^*)$. Because of Theorem 7, for any $\varepsilon > 0$, there exists an integer K such

that for every integer $t > K$ and for all l, $|D(\boldsymbol{\lambda}^t) - D(\boldsymbol{\lambda}^*)| < \varepsilon$. By definition,

$$\begin{aligned} D(\boldsymbol{\lambda}^t) &= \sum_{s,c,v} U_s(X_v) z^t_{s,c,v} - \sum_l \lambda^t_l \left(\sum_{s,c,v} X_v H^l_{s,c} z^t_{s,c,v} - R_l \right) \\ &= \sum_{s,c,v} U_s(X_v) z^t_{s,c,v} - \sum_l \lambda^*_l \left(\sum_{s,c,v} X_v H^l_{s,c} z^t_{s,c,v} - R_l \right) \\ &\quad + \sum_l (\lambda^*_l - \lambda^t_l) \left(\sum_{s,c,v} X_v H^l_{s,c} z^t_{s,c,v} - R_l \right) \end{aligned}$$

We know that $D(\boldsymbol{\lambda}^t) \geq D(\boldsymbol{\lambda}^*) - \varepsilon$, $|\lambda^t_l - \lambda^*_l| < \varepsilon/2$, and $\sum_{s,c,v} X_v H^l_{s,c} z^t_{s,c,v} - R_l$ is bounded for all l, t since $0 \leq z^t_{s,c,v} \leq 1$. Let $B := \frac{|\mathbb{L}|}{2} \max_{\boldsymbol{z}^t, l, t} |\sum_{s,c,v} X_v H^l_{s,c} z^t_{s,c,v} - R_l| + 1$. We then have

$$\begin{aligned} D(\boldsymbol{\lambda}^*) \leq & \sum_{s,c,v} U_s(X_v) z^t_{s,c,v} \\ & - \sum_l \lambda^*_l \left(\sum_{s,c,v} X_v H^l_{s,c} z^t_{s,c,v} - R_l \right) + \varepsilon B. \end{aligned}$$

Taking weighted average of both sides from $k = t+1$ to $t+T$, and then letting $T \to \infty$, we have

$$\begin{aligned} D(\boldsymbol{\lambda}^*) \leq & \sum_{s,c,v} U_s(X_v) \bar{z}^t_{s,c,v} \\ & - \sum_l \lambda^*_l \left(\sum_{s,c,v} X_v H^l_{s,c} \bar{z}^t_{s,c,v} - R_l \right) + \varepsilon B. \end{aligned}$$

Note that $\sum_{s,c,v} X_v H^l_{s,c} \bar{z}^t_{s,c,v} = R_l$. Therefore,

$$\sum_{s,c,v} U_s(X_v) z^*_{s,c,v} = D(\boldsymbol{\lambda}^*) \leq \sum_{s,c,v} U_s(X_v) \bar{z}^t_{s,c,v} + \varepsilon B,$$

and this concludes the proof of 2). □

3.4 The Content Placement Problem (CoP)

We now discuss the content placement (CoP) problem, which entails deciding $p_{c,v}$, the indicator function that network cache c stores video version v, for all c and v. As discussed in Section 3.2, a

major challenge to our optimization problem (3.1a)–(3.1f) is that the vector $\boldsymbol{p}$ needs to be updated much less frequently than the vector $\boldsymbol{z}$. To address this challenge, we introduce pseudo-variables $\boldsymbol{z}' := [z'_{s,c,v}]$ and $\boldsymbol{p}' := [p'_{s,c,v}]$, which can be updated much more frequently than $\boldsymbol{p}$, to replace $\boldsymbol{z}$ and $\boldsymbol{p}$.[4] We only update $\boldsymbol{p}$, the real content placement, after $\boldsymbol{p}'$ converges. Also, we relax (3.1f) by allowing $p'_{c,v}$ and $z'_{s,c,v}$ to be any real number between 0 and 1. We can now rewrite (3.1a)–(3.1f) as:

CoP-Primal

$$\text{maximize} \quad \sum_{s,c,v\in\mathbb{I}_s} U_s(X_v) z'_{s,c,v} \tag{3.10a}$$

$$\text{subject to} \quad \sum_{v} Y_v p'_{c,v} \leq B_c, \qquad \forall c \in \mathbb{C}, \tag{3.10b}$$

$$\sum_{c,v\in\mathbb{I}_s} z'_{s,c,v} = 1, \qquad \forall s \in \mathbb{S}, \tag{3.10c}$$

$$z'_{s,c,v} \leq p'_{c,v}, \qquad \forall s, c, v, \tag{3.10d}$$

$$\sum_{s,c,v} X_v H^l_{s,c} z'_{s,c,v} \leq R_l, \qquad \forall l \in \mathbb{L}, \tag{3.10e}$$

$$0 \leq p'_{c,v} \leq 1, 0 \leq z'_{s,c,v} \leq 1, \quad \forall s, c, v. \tag{3.10f}$$

3.4.1 Overview of the Solution

Similar to our solution to the CaVe problem, we will consider a dual problem to the CoP-Primal problem. Let $\boldsymbol{\mu}' := [\mu'_{s,c,v}]$, and $\boldsymbol{\lambda}' := [\lambda'_l]$ be the vectors of Lagrange multipliers associated with

[4]The pseudo-variables carry state information that needs to be shared between user applications and the network in the implementation.

each constraint in (3.10d) and (3.10e) respectively. The Lagrangian is then

$$
\begin{aligned}
&L'(\boldsymbol{p}', \boldsymbol{z}', \boldsymbol{\lambda}', \boldsymbol{\mu}') \\
&:= \sum_{s,c,v \in \mathbb{I}_s} U_s(X_v) z'_{s,c,v} - \sum_l \lambda'_l \big(\sum_{s,c,v} X_v H^l_{s,c} z'_{s,c,v} - R_l \big) \\
&\quad - \sum_{s,c,v} \mu'_{s,c,v} (z'_{s,c,v} - p'_{c,v}).
\end{aligned} \tag{3.11}
$$

The dual objective, $D'(\boldsymbol{\lambda}', \boldsymbol{\mu}')$, is defined as the maximum value of $L'(\boldsymbol{p}', \boldsymbol{z}', \boldsymbol{\lambda}', \boldsymbol{\mu}')$ over $\boldsymbol{p}'$ and $\boldsymbol{z}'$ subject to constraints (3.10b), (3.10c) and (3.10f). We call the optimization problem CoP-Lagrangian:

CoP-Lagrangian

$$
\begin{aligned}
\text{maximize} \quad & L'(\boldsymbol{p}', \boldsymbol{z}', \boldsymbol{\lambda}', \boldsymbol{\mu}') && && \text{(3.12a)} \\
\text{subject to} \quad & \sum_v Y_v p'_{c,v} \leq B_c, && \forall c \in \mathbb{C}, && \text{(3.12b)} \\
& \sum_{c,v \in \mathbb{I}_s} z'_{s,c,v} = 1, && \forall s \in \mathbb{S}, && \text{(3.12c)} \\
& 0 \leq p'_{c,v} \leq 1,\ 0 \leq z'_{s,c,v} \leq 1, && \forall s, c, v. && \text{(3.12d)}
\end{aligned}
$$

Remark 2. *We note that an important difference between CoP-Lagrangian and CaVe-Lagrangian is that CoP-Lagrangian relaxes the constraint (3.10d) as well. Since the pseudo-variable $z'_{s,c,v}$ in CoP-Primal bears no physical meaning, this constraint can now be temporarily violated in practice.*

The dual problem, which we call CoP-Dual, is to find the Lagrange multipliers that minimize $D'(\boldsymbol{\lambda}', \boldsymbol{\mu}')$:

CoP-Dual

$$\text{minimize} \quad D'(\boldsymbol{\lambda}', \boldsymbol{\mu}') \tag{3.13a}$$

$$\text{subject to} \quad \lambda'_l \geq 0, \quad \forall l \in \mathbb{L}, \tag{3.13b}$$

$$\mu'_{s,c,v} \geq 0, \quad \forall s \in \mathbb{S}, c \in \mathbb{C}, v \in \mathbb{V}. \tag{3.13c}$$

It is straightforward to show the following theorem:

Theorem 9. *CoP-Primal and CoP-Dual have the same optimal value.*

Proof. We use the same justification as in the previous section. The objective function of CoP-Primal is a linear function, and hence is concave. The set of $\boldsymbol{z}'$ and $\boldsymbol{p}'$ that satisfies the three unrelaxed constraints, namely, (3.10b), (3.10c), and (3.10f), is nonempty and convex.

Furthermore, the relaxed constraints (3.10d) and (3.10e) are linear and thus convex. To get strict inequalities in (3.10d) and (3.10e), we i) choose $0 < \varepsilon < \min\{\frac{\min_l R_l}{|\mathbb{S}||\mathbb{V}|}\frac{\min_s |\mathbb{I}_s|-1}{\max_v X_v}, \frac{\min_c B_c}{2\sum_v Y_v}\}$; ii) set $p'_{c,v}$ to be 1 if c is the root node and v is the null version, and 2ε otherwise; iii) set $z'_{s,c,v}$ to be $1-\varepsilon$ if c is the root node and v is the null version, and $\frac{\varepsilon}{|\mathbb{C}|(|\mathbb{I}_s|-1)}$ otherwise. It is straightforward to see that (3.10b), (3.10c), and (3.10f) are satisfied, while (3.10d) and (3.10e) are satisfied with strict inequalities.

Hence, this theorem holds following Theorem 6.2.4 (Strong Duality Theorem) in [39]. □

We will solve CoP-Primal by solving CoP-Dual. We discuss our solutions to CoP-Lagrangian and CoP-Dual below.

3.4.2 The Solution to CoP-Lagrangian

We first rewrite $L'(\boldsymbol{p}', \boldsymbol{z}', \boldsymbol{\lambda}', \boldsymbol{\mu}')$ as:

$$\begin{aligned} & L'(\boldsymbol{p}', \boldsymbol{z}', \boldsymbol{\lambda}', \boldsymbol{\mu}') \\ & = \sum_{s} \sum_{c,v} z'_{s,c,v} \left(U_s(X_v) - X_v \sum_{l:H^l_{s,c}=1} \lambda'_l - \mu'_{s,c,v} \right) \\ & \quad + \sum_{c} \sum_{v} p'_{c,v} \sum_{s} \mu'_{s,c,v} + \sum_{l} \lambda'_l R_l. \end{aligned} \tag{3.14}$$

Let $\boldsymbol{z}'_s$ be the vector containing all $[z'_{s,c,v}]$ for a given s and $\boldsymbol{p}'_c$ be the vector containing all $[p'_{c,v}]$ for a given c. Also, let $\bar{L}_s(\boldsymbol{z}'_s, \boldsymbol{\lambda}', \boldsymbol{\mu}') := \sum_{c,v} z'_{s,c,v}[U_s(X_v) - X_v \sum_{l:H^l_{s,c}=1} \lambda'_l - \mu'_{s,c,v}]$, $\hat{L}_c(\boldsymbol{p}'_c, \boldsymbol{\mu}') := \sum_v p'_{c,v}(\sum_s \mu'_{s,c,v})$, and $B(\boldsymbol{\lambda}') := \sum_l \lambda_l R_l$. Then, we have

$$\begin{aligned} & L'(\boldsymbol{p}', \boldsymbol{z}', \boldsymbol{\lambda}', \boldsymbol{\mu}') \\ & = \sum_{s} \bar{L}_s(\boldsymbol{z}'_s, \boldsymbol{\lambda}', \boldsymbol{\mu}') + \sum_{c} \hat{L}_c(\boldsymbol{p}'_c, \boldsymbol{\mu}') + B(\boldsymbol{\lambda}'), \end{aligned} \tag{3.15}$$

which gives rise to a natural decomposition among all users and network caches. Specifically, consider the two subproblems, namely, CoP-User$_s$ and CoP-Cache$_c$, below. For fixed vectors $\boldsymbol{\lambda}'$ and $\boldsymbol{\mu}'$, CoP-Lagrangian can be solved by solving CoP-User$_s$ for each s and CoP-Cache$_c$ for each c.

CoP-User$_s$

$$\text{maximize} \quad \sum_{c,v} z'_{s,c,v} \big(U_s(X_v) - X_v \sum_{l:H^l_{s,c}=1} \lambda'_l - \mu'_{s,c,v} \big) \tag{3.16a}$$

$$\text{subject to} \quad \sum_{c,v \in \mathbb{I}_s} z'_{s,c,v} = 1, \tag{3.16b}$$

$$0 \le z'_{s,c,v} \le 1, \quad \forall c \in \mathbb{C}, v \in \mathbb{V}. \tag{3.16c}$$

CoP-Cache$_c$

$$\text{maximize} \quad \sum_v p'_{c,v} \sum_s \mu'_{s,c,v} \tag{3.17a}$$

$$\text{subject to} \quad \sum_v Y_v p'_{c,v} \leq B_c, \tag{3.17b}$$

$$0 \leq p'_{c,v} \leq 1, \quad \forall v \in \mathbb{V}. \tag{3.17c}$$

CoP-User$_s$ can be solved by the following algorithm: First, find (c^*, v^*) that has the maximum value of $U_s(X_v) - X_v \sum_{l:H^l_{s,c}=1} \lambda'_l - \mu'_{s,c,v}$ among all (c, v) with $v \in \mathbb{I}_s$. Ties can be broken arbitrarily. Second, set $z'_{s,c^*,v^*} = 1$, and $z'_{s,c,v} = 0$ for all other (c, v). Alg. 4 shows the algorithm.

On the other hand, CoP-Cache$_c$ can be solved by the following greedy algorithm: First, sort all video versions v in decreasing order of $\frac{\sum_s \mu'_{s,c,v}}{Y_v}$ so that $\frac{\sum_s \mu'_{s,c,1}}{Y_1} \geq \frac{\sum_s \mu'_{s,c,2}}{Y_2} \geq \ldots$. Second, starting from $v = 1$, set $p_{c,v}$ to be the largest possible value without violating any constraints. Specifically, set $p'_{c,v} = \min\{1, (B_c - \sum_{v'<v} Y_{v'} p'_{c,v'})/Y_v\}$. It is straightforward to verify that this greedy algorithm achieves the optimal solution for CoP-Cache$_c$, since it is a fractional knapsack problem.

Remark 3. *Recall that $p_{c,v}$ is the indicator function that c stores v, which needs to be an integer. The optimal solution to CoP-Cache$_c$ may not be integer. However, from the description of our greedy algorithm, it is obvious that, for each c, there is at most one v with non-integer $p_{c,v}$. In practice, we make each network cache c store only video versions with $p_{c,v} = 1$. Since all but one version have integer $p_{c,v}$, this approach is close to optimal.*

3.4.3 The Solution to CoP-Dual

The CoP-Dual problem involves two Lagrange multipliers, $\boldsymbol{\lambda}'$ and $\boldsymbol{\mu}'$. They are updated as in Alg. 5 and 6. The following lemma and theorem, whose proofs are omitted due to space constraint, show that these algorithms solve CoP-Dual.

Lemma 6. *Given $\boldsymbol{\lambda}'$ and $\boldsymbol{\mu}'$, let $\boldsymbol{z}'^*$ and $\boldsymbol{p}'^*$ be the vectors that solve CoP-User$_s$ and CoP-Cache$_c$. Then the vector $\boldsymbol{g}' := [[R_l - \sum_{s,c,v} X_v H^l_{s,c} z'^*_{s,c,v}], [p'^*_{c,v} - z'^*_{s,c,v}]]$ is a subgradient of $D'(\boldsymbol{\lambda}', \boldsymbol{\mu}')$.*

Proof. Since $\boldsymbol{z}'^*$ and $\boldsymbol{p}'^*$ solves CoP-User$_s$ and CoP-Cache$_c$ respectively, they jointly solve CoP-Lagrangian for the given $\boldsymbol{\lambda}'$ and $\boldsymbol{\mu}'$. That is, $D'(\boldsymbol{\lambda}', \boldsymbol{\mu}') = L'(\boldsymbol{p}'^*, \boldsymbol{z}'^*, \boldsymbol{\lambda}', \boldsymbol{\mu}')$. Therefore, for any $\tilde{\boldsymbol{\lambda}}' := [\tilde{\lambda}'_l]$ and $\tilde{\boldsymbol{\mu}}' := [\tilde{\mu}'_{s,c,v}]$, where $\tilde{\lambda}_l \geq 0$ and $\tilde{\mu}'_{s,c,v} \geq 0$,

$$
\begin{aligned}
&D'(\tilde{\boldsymbol{\lambda}}', \tilde{\boldsymbol{\mu}}') - D'(\boldsymbol{\lambda}', \boldsymbol{\mu}') \\
&\quad \geq L'(\boldsymbol{p}'^*, \boldsymbol{z}'^*, \tilde{\boldsymbol{\lambda}}', \tilde{\boldsymbol{\mu}}') - L'(\boldsymbol{p}'^*, \boldsymbol{z}'^*, \boldsymbol{\lambda}', \boldsymbol{\mu}') \\
&\quad = -\sum_l (\tilde{\lambda}'_l - \lambda'_l) \left(\sum_{s,c,v} z'^*_{s,c,v} H^l_{s,c} X_v - R_l \right) \\
&\qquad - \sum_{s,c,v} (\tilde{\mu}'_{s,c,v} - \mu'_{s,c,v})(z'^*_{s,c,v} - p'^*_{c,v}) \\
&\quad = [\tilde{\boldsymbol{\lambda}}' - \boldsymbol{\lambda}', \tilde{\boldsymbol{\mu}}' - \boldsymbol{\mu}']^T \boldsymbol{g}'.
\end{aligned}
$$

Hence, $\boldsymbol{g}'$ is a subgradient of $D'(\boldsymbol{\lambda}', \boldsymbol{\mu}')$. □

Theorem 10. *Let $\{h_t\}$ be a sequence of non-negative numbers with $\sum_{t=0}^{\infty} h_t = \infty$ and $\lim_{t\to\infty} h_t = 0$, then Alg. 5 and 6 together solve CoP-Dual.*

Proof. The proof is virtually the same as that of Theorem 7. Note that the objective function of CoP-Dual is convex in $\boldsymbol{\lambda}'$ and $\boldsymbol{\mu}'$, and the feasible region is a nonempty, convex, closed subset of $\mathbb{R}^{|\mathbb{L}|+|\mathbb{S}||\mathbb{C}||\mathbb{V}|}$. With Lemma 6 and the step size sequence specified in the theorem, Alg. 5 and 6 together solve CaVe-Dual following Theorem 8.9.2 in [39]. □

Algorithm 4 CoP-User$_s$ Algorithm

1: Obtain $\boldsymbol{\mu}'$ and $\boldsymbol{\lambda}'$
2: $z'_{s,c,v} \leftarrow 0, \forall c, v$
3: $(c^*, v^*) \leftarrow \text{argmax}_{c,v \in \mathbb{I}_s} U_s(X_v) - X_v \sum_{l:H^l_{s,c}=1} \lambda'_l - \mu'_{s,c,v}$
4: $z'_{s,c^*,v^*} \leftarrow 1$

Algorithm 5 CoP-Link$_l$ Algorithm

$t \leftarrow 0$, $\lambda'_l \leftarrow 0$
while true **do**
    Obtain $\boldsymbol{z}'$ from Alg. 4
    $\lambda'_l \leftarrow \left[\lambda'_l + h_t(\sum_{s,c,v} X_v H^l_{s,c} z'_{s,c,v} - R_l)\right]^+$
    $t \leftarrow t + 1$

Algorithm 6 CoP-Cache$_c$ Algorithm

$t \leftarrow 0$, $\mu'_{s,c,v} \leftarrow 0$
while true **do**
    Obtain $\boldsymbol{z}'$ from Alg. 4
    $\mu'_{s,c,v} \leftarrow \left[\mu'_{s,c,v} + h_t(z'_{s,c,v} - p'_{c,v})\right]^+ \forall s, v$
    Sort all versions so that $\frac{\sum_s \mu'_{s,c,1}}{Y_1} \geq \frac{\sum_s \mu'_{s,c,2}}{Y_2} \geq \ldots$
    $B' \leftarrow B_c$
    for $v = 1 \rightarrow |\mathbb{V}|$ **do**
        $p'_{c,v} \leftarrow \min\{1, \frac{B'}{Y_v}\}$
        $B' \leftarrow B' - Y_v p'_{c,v}$
    $t \leftarrow t + 1$

3.4.4 The Solution to CoP-Primal

We have the following theorem regarding the optimality of $\boldsymbol{z}'$ obtained by running Alg. 4, Alg. 5, and Alg. 6 iteratively:

Theorem 11. *Let $\boldsymbol{z}'^*$ be the vector that solves CoP-Primal, $\boldsymbol{\lambda}'^k$ be the vector produced by Alg. 5 after k iterations, $\boldsymbol{\mu}'^k$ be the vector produced by Alg. 6 after k iterations, and $\boldsymbol{z}'^k$ be the vector produced by Alg. 4 when $\boldsymbol{\lambda}' = \boldsymbol{\lambda}'^k$ and $\boldsymbol{\mu}' = \boldsymbol{\mu}'^k$. Let $\bar{\boldsymbol{z}}'^t$ be the weighted average of $\boldsymbol{z}^k$ after the first t iterations, i.e. $\bar{\boldsymbol{z}}'^t := \lim_{T\to\infty} \frac{\sum_{k=t+1}^{t+T} h_k \boldsymbol{z}'^k}{\sum_{k=t+1}^{t+T} h_k}$. Then, for any $\varepsilon > 0$, there exists an integer K such that for every $t > K$,*

1. *$\bar{\boldsymbol{z}}'^t$ satisfies all CoP-Primal constraints;*
2. *$\sum_{s,c,v} U_s(X_v) z'^*_{s,c,v} - \sum_{s,c,v} U_s(X_v) \bar{z}'^t_{s,c,v} \leq \varepsilon$.*

Proof. The proof is virtually the same as that of Theorem 8 and thus omitted. □

3.5 Implementation on Named Data Networking

In this section, we discuss the implementation of our algorithms on Named Data Networking (NDN). We first introduce the NDN architecture briefly, and then show how we implement our algorithms following the NDN philosophy.

3.5.1 NDN Architecture

NDN is a future Internet architecture where every piece of data is associated with a unique hierarchical name. When a user wants to obtain a piece of named data, the user device sends out an *interest packet* with the name of the data. Note that usually the interest packet does not specify the destination location. NDN routers have built-in caches. When a router receives an interest packet, it first checks whether the named data is cached or not. If cached, it directly replies with the corresponding data packet. Otherwise, it forwards the interest packet to the next hop according to the employed forwarding strategy. The content producer e.g. video service provider is responsible for generating data packets for a certain name space. The data packet follows the reverse route of the interest packet to the user.

3.5.2 Placement of Data

In our implementation, there are three types of data: packets of video contents, decision variables ($z'_{s,c,v}$ and $p_{c,v}$), and Lagrange multipliers (λ_l, λ'_l, and $\mu'_{s,c,v}$). We assign each of them a unique name. For example, a video version has a name prefix such as `/r/file1/v1`, and $\mu'_{1,2,3}$ has `/mu2/1_3`. Each prefix is appended a sequence number to uniquely identify video packets and variables in different iterations. Naturally, video contents are placed at network caches according to the video versions.[5] Decision variables $z'_{s,c,v}$ are stored and updated at the corresponding user s. Decision variables $p_{c,v}$ and Lagrange multipliers $\mu'_{s,c,v}$ are stored and updated at the corresponding network cache c. Finally, Lagrange multipliers λ_l and λ'_l of link l from node A to B are stored and updated at node A that is closer to the cache.

[5] Videos are cached in full rather than at the packet level.

3.5.3 Implementation of User Algorithms

From Alg. 2 and 4, we can see that each user s needs to know the values of $p_{c,v}$, λ_l, λ'_l, and $\mu'_{s,c,v}$. Each user periodically sends out interest packets for the named data of these variables. Since the names of these data indicate the entities that store them, routers can easily route the interest packets to the correct destinations. Further, as data packets traverse in the reverse route of their corresponding interest packets, each router can cache all latest values of $p_{c,v}$ and λ_l that pass through it.

With the information of $p_{c,v}$ and λ_l, each user s can find the best video version v^* and cache c^* via Alg. 2. User s then sends out interest packets for video version v^* at a rate indicated by X_{v^*}. Note that these interest packets only contain information about the video version v^*, and not the destination c^*. Nevertheless, the following forwarding strategy ensures the interest packet will be eventually forwarded to c^* assuming no link failure or topology change: When a router receives an interest packet for video version v^*, it finds the network cache $c^\dagger$ that has the smallest *cost*, where the cost is defined as $\sum_l \lambda_l$ over all link l on the path to the network cache c, among those that store v^*, i.e., $p_{c,v^*} = 1$. It then forwards the interest packet to the next router on the path toward $c^\dagger$. Note that routers store all values of $p_{c,v}$ and λ_l that pass through it and thus do not need additional message passing.

With the information of $p_{c,v}$, λ'_l and μ'_l, each user s can decide the video version v^* and network cache c^* such that $z'_{s,c^*,v^*} = 1$ via Alg. 4. Each user s then sends out a pseudo-interest packet with the name of z'_{s,c^*,v^*}. We call it a pseudo-interest packet since it is used to inform the caches the changes of $z_{s,c,v}$ instead of requesting information. The replied data packet from cache c^* carries no meaning payload and is ignored.

3.5.4 Implementations for Routers and Caches

We now discuss the implementations of Alg. 3, 5, and 6. In Alg. 3, each router needs to know $\sum_{s,c,v} X_v H^l_{c,v} z_{s,c,v}$ to update λ_l for its links. We note that $\sum_{s,c,v} X_v H^l_{c,v} z_{s,c,v}$ can be estimated by the product of the rate of interest packets going through the opposite link to l and video data

packet size. As the router knows the rate of interest packets going through l, it can update λ_l directly without requesting additional information. Likewise, Alg. 5, and 6 can be carried out if one knows $z'_{s,c,v}$. This is achieved by user s sending out a pseudo-interest packet as explained in Section 3.5.3. Besides, R_l, the maximum achievable video bit rate of link l, is obtained from stress tests. Table 3.1 gives some examples of R_l for certain labeled link capacities.

Table 3.1: Maximum achievable video bit rates.

Labeled capacity (Mbps)	R_l (Mbps)
25	23.722
100	94.900
1000	947.279

3.6 Evaluations

We present our simulation evaluation results in this section. All simulations are conducted on ndnSIM [35], an ns-3 based NDN simulator.

We consider the Information Centric Edge network in Figure 3.1 for evaluation. Same as in [36], the topology of network caches follows the three-tier hierarchy of the YouTube video delivery system. There are 15 network routers with caches in total, including the root node and 8 edge caches. Each edge cache serves 20 users who have different types of devices and are interested in different videos.

We consider a catalog of 200 different videos, each with 5 different versions. The popularity of these videos follows the Zipf distribution with the shape parameter equal to 1. The 5 versions correspond to video resolutions of 360p, 720p, 1080p, 1440p (2K), and 2160p (4K) respectively.[6] The data rate of streaming each video version is set based on measurement results for YouTube videos with H.264 codec [40]. The access link capacities between users and edge caches are $25\,\mathrm{Mbps}$ each so that one can stream a 4K video. The capacities of links between caches and the

[6]The aspect ratio is assumed to be $16:9$ as in YouTube. For example, a 720p video has a resolution of 1280×720.

root node are $100\,\mathrm{Mbps}$ each so that the number of concurrent 4K streams is low. We assume each video is one-hour long, and the file sizes of video versions are calculated accordingly. The root node holds all video versions. Each edge (or primary), secondary, and tertiary cache is assumed to be able to hold all versions of one, two, and four videos respectively.

As for user utilities, we categorizes user devices into three types: smartphones, laptops or tablets, and TVs. The utility function of each user has the form $U(X_v) = \alpha \ln \min(X_v, \bar{X})$, where α is a scaling factor capturing the effect of the screen size, X_v is the data rate of video version v in Mbps, and $\bar{X}$ is a cutoff rate reflecting the limit of the device resolution. For the three types, we set a scaling factor of $20, 40, 60$ and a cutoff rate corresponding to a 1080p, 2K, 4K video respectively. Besides, we set $U(0) = -100$, which is much smaller than all regular utilities.

To evaluate the performance of our algorithms, we implement and compare the following four policies:

- **Optimal**: This policy tries to find the optimal solution to the CaVe-CoP problem by solving the integer program numerically via the GLPK toolbox. Note that it is a centralized policy and involves solving a high-dimensional problem.

- **CaVe-CoP**: This refers to our algorithms Alg. 2–6.

- **CaVe-CAV**: In this policy, each user employs our algorithms for CaVe. For content placement, if a network cache decides to store a video version, it needs to cache all versions (CAV) of the same video. We note that this content placement strategy is consistent with design practices in commercial CDNs. As a result, each network cache simply stores the most popular videos, subject to its storage constraint.

- **Greedy-CoP**: In this policy, each user chooses the version that matches its cutoff rate. Network caches employ our algorithms for CoP.

For each simulation, we use the video contents that each user actually receives to calculate the total utility of all users and the average % stall time, i.e. the percentage of time that video

streaming stalls[7], of all users. The metrics are calculated at each CaVe iteration, i.e. every $0.1\,\mathrm{s}$. We run CoP iterations every $0.2\,\mathrm{s}$, and apply content placement results at $20\,\mathrm{s}$.

Fig. 3.2 and Fig. 3.3 present our simulation results. For our simulated scenario, Optimal cannot find the exact integer solution for utility. Instead, it reports a upper bound from linear programming (LP) relaxation, and a lower bound by integer programming (IP) heuristics. Note that Optimal reports ideal utility instead of perceived utility. We can observe that our Cave-CoP policy achieves near-optimal utility, significantly outperforming the two baseline policies, even though they involve subsets of our algorithms. Besides, our policy approaches zero stall time. Note that the jumps near $20\,\mathrm{s}$ in the figures are due to applying content placement results.

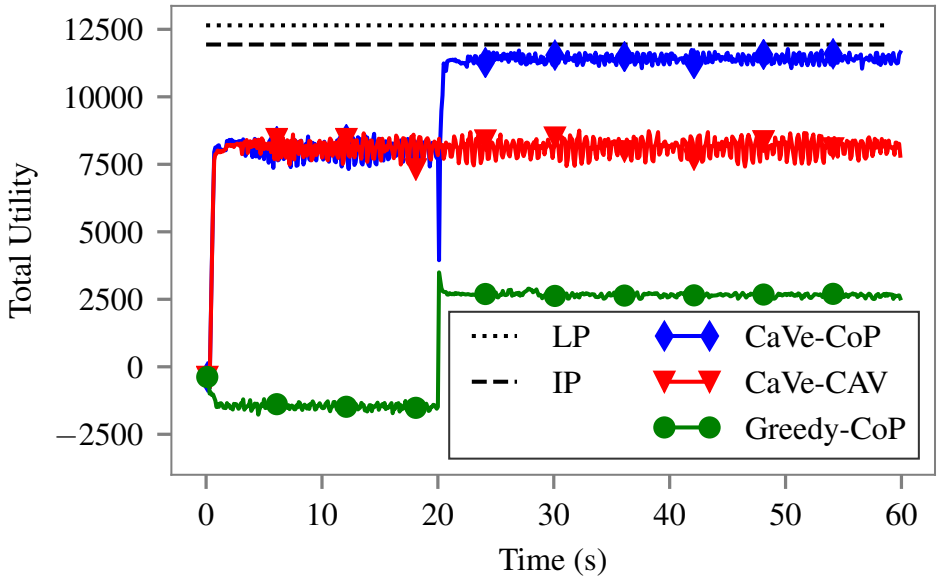


Figure 3.2: Comparison of total utility. Reprinted with permission from [37].

3.7 Related Work

There has been rich literature on adaptive video streaming. An early work identified a cross layer framework for adaptive video streaming in IP networks [41]. More recently, experiment-

[7]Video streaming stalls when all received video contents are consumed.

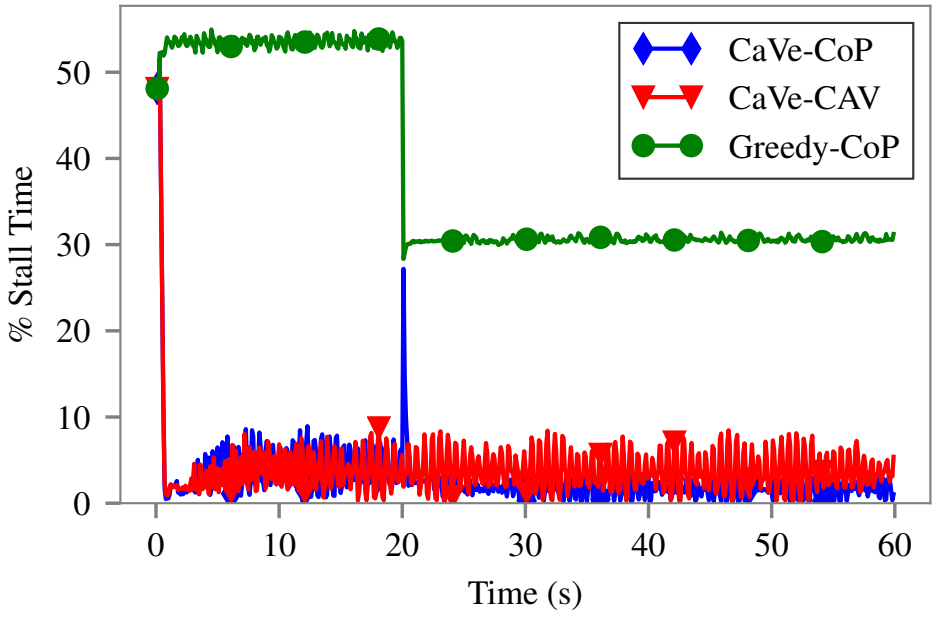


Figure 3.3: Comparison of % stall time. Reprinted with permission from [37].

based investigations have been conducted on the YouTube video delivery system [42] as well as CDNs run by Akamai [43] and Netflix [44]. Liu *et al.* [45] made a case for a coordinated control plane across CDNs for video streaming to provide high quality of experience. The performance has also been investigated on NDN [46]. Wireless edge networks are promising to enhance the benefits of CDNs for adaptive video streaming [47], where the user scheduling problem has been studied [48].

Content caching is crucial for practical adaptive video streaming in Information Centric Edge networks. Considering distributed caches, Ramadan *et al.* [36] proposed the abstraction of "BIG" cache to effectively utilize the resources. Applegate *et al.* [49] studied optimal content placement of videos with a focus on scalability. There have been many studies on joint optimization of content caching and packet routing. Yeh *et al.* [50] proposed a framework for joint forwarding and caching in NDN. Wang *et al.* [51] employed stochastic network utility maximization and developed a distributed forwarding and caching algorithm. Ioannidis and Yeh [52] studied the routing cost minimization problem of joint routing and caching, where the cost is incurred per link. These

studies are not directly applicable to multi-version video streaming since different versions of the same video can be stored in different caches.

Our work formulates the joint cache-version selection and content placement problem as a network utility maximization (NUM) problem, and uses the well-known primal dual approach and dual decomposition [53]. However, there are notable differences between our work and traditional NUM research. Existing studies have explored various scenarios including time varying channel with delay constraints [54], delay sensitive fairness [55], multiple flow classes [56], multiple protocols [57] and so on, while assuming a static source-destination pair per user (flow). In contrast, in our work, a user could obtain its desired content from in-network caches as well as the content producer.

3.8 Conclusion

In this chapter, we have studied the CaVe-CoP problem, i.e. the joint optimization of cache-version selection and content placement, for adaptive video streaming in Information Centric Edge networks. Realizing that there is a practical timescale separation between CaVe and CoP, we have proposed a set of algorithms that provably optimize CaVe and CoP respectively. Further, we show that our algorithms can be practically implemented on NDN in a distributed fashion. Simulation evaluations on ndnSIM demonstrate that our policy significantly outperforms baseline policies with conventional heuristics.

4. EFFICIENT COST ALLOCATION*

This chapter proposes a practical non-monetary mechanism that induces the efficient solution to the optimal rate control problem, where each client optimizes its request arrival rate to maximize its own net utility individually, and at the Nash Equilibrium the total net utility of the system is also maximized. Existing mechanisms typically rely on monetary exchange which requires additional infrastructure that is not always available. Instead, the proposed mechanism is based on efficient cost allocation, where the cost is in terms of non-monetary metric such as average delay or request loss rate. Specifically, we present an efficient cost allocation rule for the server to determine the target cost of each client. We then propose an intelligent policy for the server to control the costs of the clients to achieve the efficient allocation. Furthermore, we design a distributed rate control protocol with provable convergence to the Nash Equilibrium of the system. The effectiveness of our mechanism is extensively evaluated via simulations of both delay allocation and loss rate allocation against baseline mechanisms with classic control policies.

4.1 Introduction

The mobile Internet market has been enjoying an unprecedented growth in recent years. It is predicted that the trend will continue, and the global mobile data traffic will increase sevenfold between 2016 and 2021 [58]. With the growing market, it is of great interest to understand the economics of the network. In this chapter, we are interested in finding a practical mechanism to induce the efficient solution to the optimal rate control problem in a network system of multiple selfish and strategic clients. We consider systems where a server processes requests from multiple clients, and each client can dynamically adjust its own request arrival rate. Each client obtains some utility based on its request arrival rate and its own utility function, but also suffers from some disutility based on some cost such as its experienced delay or request losses. Each client optimizes

its request arrival rate to maximize its own net utility individually. The server's goal is to ensure that the total net utility is maximized at the Nash Equilibrium. Our system model can be applied to a wide range of networks. For example, the clients might be smartphones, wearable devices, tablets and so on, and the server can be a cellular base station (e.g. LTE eNodeB) or a Wi-Fi hotspot which provides Internet services to the clients. Each request corresponds to an LTE subframe or an IP packet.

The optimal rate control problem, which entails maximizing the total net utility in the system, is typically convex, and it is thus easy to solve when one has complete information of all the individual utility functions. In practice, however, the utility functions are often private information of clients, and a strategic client that aims to maximize its own net utility may not reveal its true utility function. Further, request rates are directly controlled by clients, instead of the server. Most existing work employs some auction or pricing scheme that ensures strategic clients reveal their true functions and follow the assigned rates from the server [54, 59]. However, these schemes involve additional monetary exchange between clients and the server, which requires additional infrastructure that is not always available.

In this chapter, we propose a novel non-monetary mechanism for optimal rate control to address this issue. Note that each client suffers from some disutility based on its experienced delay or request loss rate, and the server can indirectly adjust such disutility experienced by each client through its employed control policy. Therefore, the server can potentially steer request rates of strategic clients toward the optimal point through its control policy. Effectively, the server uses "delay" or "loss rate" as a kind of "currency."

In economic terms, there are negative externalities from a client increasing its request rate, since this increases the overall cost, in the form of delay or loss rate, of all clients. This is an analogy to a public goods problem [60], in which one client's consumption choice affects the utility and payoffs of the other clients. As such, the server's objective is to design an allocation scheme such that each client internalizes these negative externalities, thereby leading to efficient consumption of resources.

In designing the non-monetary mechanism, we make the following contributions:

1. First, for both the cost of delay and the cost of loss rate, we propose efficient cost allocation rules through which the server can determine the cost to be allocated to each client.

2. We then design control policies used by the server to allocate costs and adjust disutilities experienced by the clients. For the cost of delay, we propose a simple scheduling algorithm and proves that it achieves the efficient delay allocation in the heavy traffic regime.[1] For the cost of loss rate, we propose a simple policy that determines which request to drop when the server's buffer is full.

3. Furthermore, we present a distributed rate control protocol where clients update their request rates based on their experienced costs. The protocol is scalable and lightweight, and is proved to converge to the Nash Equilibrium where the total net utility of the system is also maximized.

Altogether, they form our non-monetary mechanism for optimal rate control through efficient cost allocation.

The rest of the chapter is organized as follows. Section 4.2 reviews the literature related to our work. Section 4.3 introduces our system model and problem formulation, using delay allocation as an example. Section 4.4, 4.5, and 4.6 present the efficient delay allocation rule, the efficient delay scheduling policy, and the distributed rate control protocol for delay allocation respectively. Section 4.7 extends the non-monetary mechanism to loss rate allocation. Simulation study is described in Section 4.8, and we conclude our chapter in Section 4.9.

4.2 Related Work

There has been a considerable amount of literature that studies networks from the respect of economics. Altman et al. gave a comprehensive survey on networking games [61]. Specifically for rate control, Kelly et al. analyzed the stability and fairness of pricing based rate control

[1] Heavy traffic means the total request rate approaches the service rate.

algorithms [59]. Alpcan and Başar gave a utility-based congestion control scheme for cost minimization and showed its stability for a general network topology [62]. Hou and Kumar presented a truthful and utility-optimal auction for wireless networks with per-packet deadline constraints [54]. Gupta et al. studied network utility maximization where flows are aggregated into flow classes [56]. Ramaswamy et al. considered the case when a client can choose from a number of congestion control protocols [57]. Despite the rich literature, most existing mechanisms require additional monetary exchange between clients and the server, and infrastructure for monetary exchange is thus necessary. However, such infrastructure is not always available in wireless networks, which in turn limits the applicability of these monetary mechanisms. In contrast, our non-monetary mechanism exploits existing wireless network properties such as delay or loss rate to realize optimal rate control. The main advantage is that no additional infrastructure for monetary exchange needs to be set up or maintained, which can be a substantial cost saving.

The intellectual foundation of our research comes from economics. The early literature began with problems of creating incentives to reduce free riding in teams, such as in Groves [63]. This research uses much of the similar logic as our method on the behavior of other agents in a strategic game. Baldenius et al. [64], Moulin and Shenker [65], and Rajan [66] studied the problem of cost allocation, namely, how to allocate a common cost to separate corporate departments. Our contribution is combining a framework that is well utilized in economics and applying it to the optimal rate control problem in wireless networks. The application to distributed networks is new to our knowledge.

Besides, our work shares a similar spirit as the standard loss-based TCP congestion control and delay-based TCP variants, such as TCP Vegas [67], TCP Westwood+ [68, 69], and FAST TCP [70], in the sense that loss or delay is used as the signal for the clients to adjust their request rates. However, our mechanism includes not only a rate update protocol but also an efficient cost allocation rule and a control policy to enforce such rule for optimal rate control.

4.3 System Model for Delay Allocation

Starting from this section, we first focus on the delay allocation problem for ease of presentation. As will be shown in Section 4.7, the system model and mechanism design can be easily extended to the loss rate allocation problem.

Consider a system with N clients and a server. Each client i generates requests by some predefined random process, such as Poisson random process, but it can dynamically adjust its average request rate, denoted by λ_i. We use $\boldsymbol{\lambda} := [\lambda_i]$ to denote the vector containing the average request rates of all clients, and λ_{-i} to denote the vector of average request rates of all clients other than i.

On the other hand, the server employs some scheduling policy to determine which request to process. Unserved requests are queued in the system. This corresponds to real systems with sufficiently large buffers, for example, campus Wi-Fi networks. The processing time of each request is a random variable with mean $\frac{1}{\mu}$. If the server's scheduling policy is work-conserving, which never idles as long as there is at least one request available for processing, then the average delay of all requests is a function of the total average request arrival rate, $\Lambda := \sum_i \lambda_i$, regardless of the employed scheduling policy. The average delay function $\bar{C}(\Lambda)$ is smooth, strictly increasing, and strictly convex. We assume that the average delay $\bar{C}(\Lambda)$ can be well fitted by a low-order polynomial function $C(\Lambda)$ via, for example, Chebyshev least squares approximation.

Suppose each client obtains some *utility* based on its request rate λ_i and suffers from *disutility* for every unit delay experienced by each of its request. Specifically, the utility of client i is $U_i(\lambda_i)$, where $U_i(\cdot)$ is a smooth, strictly increasing, and strictly concave function. Let $D_i(\lambda_i, \lambda_{-i})$ be the average delay that client i experiences for all its requests. The disutility of client i is $\lambda_i D_i(\lambda_i, \lambda_{-i})$. Client i aims to maximize its *net utility*, $U_i(\lambda_i) - \lambda_i D_i(\lambda_i, \lambda_{-i})$, by choosing its request rate λ_i.

The server aims to maximize the total net utility in the system, which can be written as $\sum_i (U_i(\lambda_i) - \lambda_i D_i(\lambda_i, \lambda_{-i}))$. Since the average delay of *all* requests is the weighted average $\frac{\sum_i \lambda_i D_i(\lambda_i, \lambda_{-i})}{\Lambda} \approx C(\Lambda)$, we say that the server aims to maximize $\sum_i U_i(\lambda_i) - \Lambda C(\Lambda)$. The system model is illustrated in Figure 4.1.

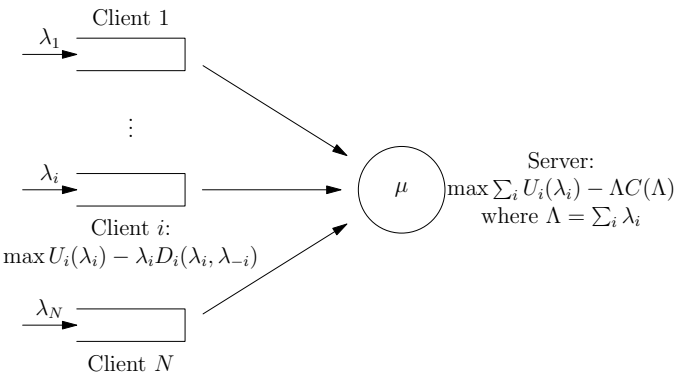


Figure 4.1: An illustration of the system model. Reprinted with permission from [71].

Note that the average delay of all requests is always infinite when the system is overloaded with $\Lambda \geq \mu$. To simplify discussions, we assume that $\boldsymbol{\lambda}$ has the properties that $\Lambda = \sum_i \lambda_i \leq (1-\epsilon)\mu$, where $\epsilon > 0$ is a predetermined value known to the server. We further assume that $\lambda_i \geq \lambda_\delta$ for all i, for some predetermined $\lambda_\delta > 0$ known to the server. These assumptions are not restrictive since we can choose ϵ and λ_δ arbitrarily close to 0. Let $\mathcal{S}_{\boldsymbol{\lambda}} := \{\boldsymbol{\lambda} \mid \Lambda \leq (1-\epsilon)\mu, \lambda_i \geq \lambda_\delta\}$ be the feasible region of $\boldsymbol{\lambda}$. The server's optimization problem is thus formally:

$$\max_{\boldsymbol{\lambda} \in \mathcal{S}_{\boldsymbol{\lambda}}} \sum_{i=1}^{N} U_i(\lambda_i) - \Lambda C(\Lambda). \tag{4.1}$$

Since $U_i(\cdot)$ is concave, $C(\cdot)$ is convex, and $\mathcal{S}_{\boldsymbol{\lambda}}$ is a convex set, the problem of maximizing the total net utility can be easily solved when one has complete information of all these functions. In practice, however, the function $U_i(\cdot)$ is the private information of client i, and a strategic client may not reveal its true $U_i(\cdot)$. Now consider a game where, given $\boldsymbol{\lambda}$, the server determines the average delay experienced by each client i, $D_i(\lambda_i, \lambda_{-i})$, with the constraint that $\sum_i \lambda_i D_i(\lambda_i, \lambda_{-i}) \geq \Lambda C(\Lambda)$. On the other hand, given λ_{-i} and $U_i(\cdot)$, each client i aims to maximize its own net utility by solving

$$\tilde{\lambda}_i = \operatorname*{argmax}_{\lambda_i} U_i(\lambda_i) - \lambda_i D_i(\lambda_i, \lambda_{-i}). \tag{4.2}$$

Note that we allow $\sum_i \lambda_i D_i(\lambda_i, \lambda_{-i})$ to be strictly larger than $\Lambda C(\Lambda)$, which can be achieved by

employing a policy that is not work-conserving and may arbitrarily delay, or drop, requests.

We say that the system reaches a Nash Equilibrium if no client in the system can improve its own net utility unilaterally.

Definition 9. *A vector* $\tilde{\boldsymbol{\lambda}} := [\tilde{\lambda}_i]$ *is said to be a Nash Equilibrium if* $\tilde{\lambda}_i = \text{argmax}_{\lambda_i} U_i(\lambda_i) - \lambda_i D_i(\lambda_i, \tilde{\lambda}_{-i}), \forall i$.

Let $\boldsymbol{\lambda}^* := [\lambda_i^*]$ be the vector that maximizes the total net utility. We assume $\boldsymbol{\lambda}^*$ lies in the interior of $\mathcal{S}_{\boldsymbol{\lambda}}$ to simplify the analysis. This assumption is not restrictive by choosing ϵ and λ_δ sufficiently small. The server's problem is to find the rule that allocates delays, $[D_i(\cdot)]$, to induce optimal choices of $[\lambda_i]$.

Definition 10. *A rule of allocating delays,* $[D_i(\cdot)]$*, is said to be* efficient *if* $\boldsymbol{\lambda}^*$ *is the only Nash Equilibrium.*

4.4 Efficient Delay Allocation

In this section, we propose the first building block of our non-monetary mechanism, an efficient delay allocation rule. The rule will be used by the server to determine how much delay should be allocated to each client given their request rates $\boldsymbol{\lambda}$.

We first study some basic properties of the optimal vector $\boldsymbol{\lambda}^* = [\lambda_i^*]$ that maximizes total net utility $\sum_i U_i(\lambda_i) - \Lambda C(\Lambda)$. We have

$$\frac{\partial}{\partial \lambda_i}\left[\sum_i U_i(\lambda_i^*) - \Lambda^* C(\Lambda^*)\right] = 0. \tag{4.3}$$

Hence,

$$U_i'(\lambda_i^*) = \frac{\partial}{\partial \lambda_i}\Lambda^* C(\Lambda^*). \tag{4.4}$$

On the other hand, if $\boldsymbol{\lambda}^*$ is also the Nash Equilibrium under some delay allocation rule $[D_i(\cdot)]$, then λ_i^* maximizes $U_i(\lambda_i) - \lambda_i D_i(\lambda_i, \lambda_{-i}^*)$, and we have

$$\frac{\partial}{\partial \lambda_i}[U_i(\lambda_i^*) - \lambda_i^* D_i(\lambda_i^*, \lambda_{-i}^*)] = 0. \tag{4.5}$$

Hence,

$$U_i'(\lambda_i^*) = \frac{\partial}{\partial \lambda_i} \lambda_i^* D_i(\lambda_i^*, \lambda_{-i}^*). \tag{4.6}$$

Combining the above equations yields

$$\frac{\partial}{\partial \lambda_i} [\Lambda^* C(\Lambda^*) - \lambda_i^* D_i(\lambda_i^*, \lambda_{-i}^*)] = 0. \tag{4.7}$$

Eq. (4.7) suggests that an efficient rule of delay allocation should ensure that $\Lambda C(\Lambda) - \lambda_i D_i(\lambda_i, \lambda_{-i})$ is only determined by λ_{-i}, and is not influenced by λ_i. It means the sum of the disutilities of all clients but i should not depend on the request rate of client i. This implication has indeed been formally stated and proved in [60]:

Proposition 1. *$[D_i(\cdot)]$ is efficient if and only if there exists functions $R_i : \mathbb{R}^{N-1} \to \mathbb{R}$ such that for all i,*

$$\lambda_i D_i(\lambda_i, \lambda_{-i}) = \Lambda C(\Lambda) - R_i(\lambda_{-i}), \tag{4.8}$$

and

$$\sum_{i=1}^{N} \lambda_i D_i(\lambda_i, \lambda_{-i}) = \Lambda C(\Lambda). \tag{4.9}$$

Recall that $C(\Lambda)$ is a low-order polynomial. Therefore, $\Lambda C(\Lambda)$ is also a low-order polynomial, and can be expressed as $\Lambda C(\Lambda) = c_1 \Lambda + c_2 \Lambda^2 + \cdots + c_m \Lambda^m$.

We now define some helpful terminology. First define the sets

$$P^j := \Big\{ \boldsymbol{p} = [p_i] \;\Big|\; p_i \text{ is a nonnegative integer}, \sum_{i=1}^{N} p_i = j \Big\}, \tag{4.10}$$

$$P_i^j := \{ \boldsymbol{p} \in P^j \Big|\; p_i = 0 \}, \tag{4.11}$$

for $j = 1, \ldots, m$ and $i = 1, \ldots, N$. Next, for $\boldsymbol{p} \in P^j$, let $G(\boldsymbol{p})$ be the number of nonzero coordinates of $\boldsymbol{p}$: $G(\boldsymbol{p}) := \big|\{ l \;\big|\; p_l \neq 0 \}\big|$. Note that $G(\boldsymbol{p})$ is at most j, for all $\boldsymbol{p} \in P^j$. Finally, define $\binom{j}{\boldsymbol{p}} := \frac{j!}{p_1! \cdots p_N!}$.

By the multinomial expansion theorem, it holds that

$$(\lambda_1 + \cdots + \lambda_N)^j = \sum_{\boldsymbol{p} \in P^j} \binom{j}{\boldsymbol{p}} \lambda_1^{p_1} \cdots \lambda_N^{p_N}. \tag{4.12}$$

We now introduce our delay allocation rule. Let

$$\beta_i^j = c_j \sum_{\boldsymbol{p} \in P_i^j} \frac{N-1}{N - G(\boldsymbol{p})} \binom{j}{\boldsymbol{p}} \lambda_1^{p_1} \cdots \lambda_N^{p_N}, \tag{4.13}$$

for $j = 1, \ldots, m$. We then choose $R_i(\lambda_{-i})$ as

$$R_i(\lambda_{-i}) = \sum_{j=1}^{m} \beta_i^j, \tag{4.14}$$

and

$$\lambda_i D_i(\lambda_i, \lambda_{-i}) = \Lambda C(\Lambda) - R_i(\lambda_{-i}). \tag{4.15}$$

(4.15) ensures that $R_i(\lambda_{-i})$ is the sum of the disutilities of all clients but i. (4.14) guarantees it does not depend on λ_i, which is consistent with the aforementioned implication.

Theorem 12. *The rule of delay allocation $[D_i(\cdot)]$ as defined by Eq. (4.14) and (4.15) is efficient.*

Proof. Since $p_i = 0$ for all $\boldsymbol{p} \in P_i^j$, it is obvious that $R_i(\lambda_{-i}) = \sum_{j=1}^m \beta_i^j$ is not influenced by λ_i.

Next, we check the condition $\sum_i \lambda_i D_i(\lambda_i, \lambda_{-i}) = \Lambda C(\Lambda)$. By Eq. (4.13), for every $\boldsymbol{p} \in P^j$, the term $\frac{N-1}{N-G(\boldsymbol{p})}\binom{j}{\boldsymbol{p}}\lambda_1^{p_1} \cdots \lambda_N^{p_N}$ appears in β_i^j if and only if $p_i = 0$, and there are $(N - G(\boldsymbol{p}))$ different i with $p_i = 0$. Therefore, the term $\frac{N-1}{N-G(\boldsymbol{p})}\binom{j}{\boldsymbol{p}}\lambda_1^{p_1} \cdots \lambda_N^{p_N}$ appears in $[\beta_i^j]$ a total number of

$(N - G(\boldsymbol{p}))$ times. We then have

$$\begin{aligned}\sum_{i=1}^{N} R_i(\lambda_{-i}) &= \sum_{i=1}^{N}\sum_{j=1}^{m} \beta_i^j \\ &= \sum_{j=1}^{m} c_j \sum_{\boldsymbol{p}\in P^j} (N-1)\binom{j}{\boldsymbol{p}} \lambda_1^{p_1}\cdots\lambda_N^{p_N} \\ &= (N-1)\Lambda C(\Lambda), \end{aligned} \tag{4.16}$$

and

$$\sum_i \lambda_i D_i(\lambda_i, \lambda_{-i}) = N\Lambda C(\Lambda) - \sum_{i=1}^{N} R_i(\lambda_{-i}) = \Lambda C(\Lambda). \tag{4.17}$$

Therefore, by Proposition 1, the rule of delay allocation $[D_i(\cdot)]$ as defined by Eq. (4.14) and (4.15) is efficient. □

Next, we briefly discuss the time complexity of calculating efficient delay allocation using the above rule. The most time consuming part is obtaining all the elements of the set P^j, whose size is no more than $O(N^j)$, for all $j = 1, \ldots, m$. We can obtain P_i^j as well as $G(\boldsymbol{p})$ and $\binom{j}{\boldsymbol{p}}$ while obtaining the elements of P^j. Therefore, the total time complexity is $O(N^m)$, where m is a small constant.

Remark: We note that the allocated delays of some clients following the efficient delay allocation rule $[D_i(\cdot)]$ as in Eq. (4.15) might be unachievable (e.g. negative) in practice, especially when their request rates are too small compared with others. We call those clients "VIP", since their allocated delays are among the smallest. Note there is always at least one non-VIP client in the system. The above delay allocation rule is efficient only when there are no VIP clients in practical systems. In the following theoretical analysis, we will focus on the case where all clients in the system are non-VIP. We will present preliminary simulation studies on VIP clients in Section 4.8.1.3.

4.5 Efficient Delay Scheduling

In this section, we propose an online scheduling policy used by the server to ensure that the actual delay experienced by each client is the same as its allocated delay, as described in Eq. (4.14)

and (4.15).

As mentioned before, we focus on non-VIP clients, and assume that $g_i := \lambda_i D_i > 0$ for all i. According to Little's law, g_i can be interpreted as the target average queue length (i.e. number of requests in the system) of client i, which is known to the server. Based on this observation, we propose the following maximum-relative-queue-length (MRQ) policy:

Definition 11 (MRQ)**.** *Let $Q_i(t)$ be the queue length of client i at time t. At time t, the MRQ policy schedules the client with the largest relative queue length, defined as $Q_i(t)/g_i$, breaking ties by scheduling the client with the lowest ID.*

The intuition behind MRQ is that by always scheduling the client with the largest relative queue length, eventually all relative queue lengths are equal on average in steady state, or equivalently, the average queue length of each client is roughly the same as its target queue length.

Below we will show that the MRQ policy indeed achieves the desirable efficient delay allocation in the heavy traffic regime.[2] In particular, we show that the deviation of the actual average delay from the target delay is bounded for each client i, regardless of the difference between the total request rate Λ and the service rate μ. When Λ approaches μ, the actual average delay goes to infinity, and therefore the deviation becomes negligible compared to the actual average delay. Our technical approach is similar to the state space collapse results in the queueing theory literature [72].

Let $\boldsymbol{g} := [g_i]$ be the vector of target queue lengths for all clients. Let $\hat{\boldsymbol{g}} := \boldsymbol{g}/\sum_i g_i$ be the normalized vector of $\boldsymbol{g}$ such that $\hat{g}_i > 0$ is the fraction of target queue length for client i and $\sum_i \hat{g}_i = 1$. Define the weighted inner product of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ by:

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle := \sum_{i=1}^{N} \frac{x_i y_i}{\hat{g}_i},$$

[2]On the other hand, if the traffic is light and queues are not built up, it is not quite necessary to employ an advanced scheduling policy. Nevertheless, MRQ can still be used in light traffic and simulation results suggest that it works reasonably well. See also Section 4.8.1.2.

and the norm of a vector $\boldsymbol{x}$ by:

$$\|\boldsymbol{x}\| := \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle}.$$

Note that $\|\hat{\boldsymbol{g}}\| = 1$ and thus $\hat{\boldsymbol{g}}$ is the unit vector in the direction of $\boldsymbol{g}$.

Let $\boldsymbol{Q}(t)$, $\boldsymbol{A}(t)$, and $\boldsymbol{S}(t)$ be the vector of queue lengths, arrivals, and services respectively for all clients at time t. To simplify discussions, we assume that time is slotted and the duration of a time slot is τ. Moreover, in each time slot, each client can generate at most one request, and the server can serve at most one request. This assumption is not restrictive as we can set τ to be arbitrarily small. Next we define the generalized projection of $\boldsymbol{Q}(t)$ onto $\boldsymbol{g}$, denoted by $\boldsymbol{Q}_{\parallel}(t)$, as follows:

$$\boldsymbol{Q}_{\parallel}(t) := \langle \boldsymbol{Q}(t), \hat{\boldsymbol{g}} \rangle \hat{\boldsymbol{g}} = \sum_{i=1}^{N} Q_i(t) \hat{\boldsymbol{g}}.$$

Since the total queue length is $\sum_i Q_i(t)$, the queue length of each client i is exactly the i-th element of $\boldsymbol{Q}_{\parallel}(t)$ if we allocate queue lengths proportionally to $\boldsymbol{g}$. Therefore, $\boldsymbol{Q}_{\parallel}(t)$ can be thought of as the vector of target queue lengths of all clients under perfect state space collapse.

The deviation $\boldsymbol{Q}_{\perp}(t)$ of actual queue lengths $\boldsymbol{Q}(t)$ from the target queue lengths $\boldsymbol{Q}_{\parallel}(t)$ is defined as:

$$\boldsymbol{Q}_{\perp}(t) := \boldsymbol{Q}(t) - \boldsymbol{Q}_{\parallel}(t).$$

Now we introduce a helpful lemma to prove the state space collapse property. Our proof is based on the Lyapunov drift techniques. First, define the following Lyapunov functions:

$$V_{\perp}(t) := \|\boldsymbol{Q}_{\perp}(t)\|, W(t) := \|\boldsymbol{Q}(t)\|^2, W_{\parallel}(t) := \|\boldsymbol{Q}_{\parallel}(t)\|^2.$$

The respective drifts are defined as follows:

$$\begin{aligned}
\Delta V_{\perp}(t) &:= V_{\perp}(t+\tau) - V_{\perp}(t) \\
\Delta W(t) &:= W(t+\tau) - W(t) \\
\Delta W_{\parallel}(t) &:= W_{\parallel}(t+\tau) - W_{\parallel}(t)
\end{aligned}$$

The following lemma, adapted from Lemma 7 in [72], shows that the drift $\Delta V_\perp(t)$ can be bounded by $\Delta W(t)$ and $\Delta W_\parallel(t)$, and absolutely bounded.

Lemma 7. *We have*

$$\Delta V_\perp(t) \le \frac{1}{2\|\boldsymbol{Q}_\perp(t)\|}(\Delta W(t) - \Delta W_\parallel(t)), \tag{4.18}$$

and

$$|\Delta V_\perp(t)| \le 2\sqrt{\frac{N}{\hat{g}_{\min}}}, \tag{4.19}$$

where $\hat{g}_{\min} := \min_i \hat{g}_i$.

Proof. See Appendix B.1. □

Since we are considering a single server system, it is easy to see our MRQ policy stabilizes the queues of all clients as long as $\Lambda < \mu$. Therefore, $\boldsymbol{Q}(t)$ converges to a limiting random vector $\bar{\boldsymbol{Q}}$ in steady state.

Consider the following limiting queueing process: fix a vector $\hat{\boldsymbol{g}}$ of unit length with $\hat{g}_i > 0$, we consider all systems whose allocated delays satisfy $\boldsymbol{g}/\sum_i g_i = \hat{\boldsymbol{g}}$. Each system is indexed by $\varepsilon := \mu - \Lambda^{(\varepsilon)}$, where $\Lambda^{(\varepsilon)}$ is the total request arrival rate of the system. We use $\bar{\boldsymbol{Q}}^{(\varepsilon)}$ to denote the random vector of queue lengths in steady state for the system, and use $\bar{\boldsymbol{Q}}_\perp^{(\varepsilon)}$ to denote the deviation in steady state. The efficiency of MRQ is formally stated in the following theorem:

Theorem 13. *The efficient delay allocation rule is enforced by the MRQ scheduling policy in the heavy traffic regime. That is, there exists a sequence of finite integers* $\{N_r\}$ *such that* $\mathbb{E}\left[\left\|\bar{\boldsymbol{Q}}_\perp^{(\varepsilon)}\right\|^r\right] \le N_r$ *for all* $r = 1, 2, \ldots$ *and for all* $\varepsilon > 0$.

Proof. Below the superscript $^{(\varepsilon)}$ is omitted for brevity. By [72, Lemma 1], we only need to show the Lyapunov drift $\Delta V_\perp(t)$ is *1.* negative when $\|\boldsymbol{Q}_\perp(t)\|$ is sufficiently large, and *2.* absolutely bounded. Lemma 7 has shown that *2)* is satisfied. Moreover, *1)* can be reduced to bound $\Delta W(t)$ and $\Delta W_\parallel(t)$.

Consider $\mathbb{E}[\Delta W(t) \mid \boldsymbol{Q}] := \mathbb{E}[\Delta W(t) \mid \boldsymbol{Q}(t) = \boldsymbol{Q}]$.

$$\begin{aligned}
&\mathbb{E}[\Delta W(t) \mid \boldsymbol{Q}] \\
&= \mathbb{E}\big[\|\boldsymbol{Q}(t+\tau)\|^2 - \|\boldsymbol{Q}(t)\|^2 \,\big|\, \boldsymbol{Q}\big] \\
&= \mathbb{E}\big[\|(\boldsymbol{Q}(t) + \boldsymbol{A}(t) - \boldsymbol{S}(t))^+\|^2 - \|\boldsymbol{Q}(t)\|^2 \,\big|\, \boldsymbol{Q}\big] \\
&\le \mathbb{E}\big[\|\boldsymbol{Q}(t) + \boldsymbol{A}(t) - \boldsymbol{S}(t)\|^2 - \|\boldsymbol{Q}(t)\|^2 \,\big|\, \boldsymbol{Q}\big] \\
&\le 2\mathbb{E}[\langle \boldsymbol{Q}(t), \boldsymbol{A}(t) - \boldsymbol{S}(t)\rangle \mid \boldsymbol{Q}] + K_1,
\end{aligned} \tag{4.20}$$

where $(\cdot)^+ := \max\{0, \cdot\}$ and K_1 is a bounded constant. Below we will omit (t) in the derivation for brevity.

Given a request rate vector $\boldsymbol{\lambda}$, define a hypothetical service rate vector $\boldsymbol{\mu} := \boldsymbol{\lambda} + \varepsilon \hat{\boldsymbol{g}}$, where $\varepsilon > 0$. Note that $\mu_\Sigma := \sum_i \mu_i = \Lambda + \varepsilon = \mu$. Recall μ is the service rate the server can provide.

Next, we bound the term $\mathbb{E}[\langle \boldsymbol{Q}, \boldsymbol{A} - \boldsymbol{S}\rangle \mid \boldsymbol{Q}]$ in Eq. (4.20). Without loss of generality, suppose at time t, client 1 has the largest relative queue length, that is $Q_1(t)/g_1 \ge Q_i(t)/g_i$ for all i. Note that by the definition of the MRQ scheduling policy,

$$\langle \boldsymbol{Q}, \mathbb{E}[\boldsymbol{S} \mid \boldsymbol{Q}]\rangle = \frac{Q_1}{\hat{g}_1}\mu \ge \frac{Q_i}{\hat{g}_i}\mu.$$

Therefore,

$$\begin{aligned}
\mathbb{E}[\langle \boldsymbol{Q}, \boldsymbol{A} - \boldsymbol{S}\rangle \mid \boldsymbol{Q}] &= \langle \boldsymbol{Q}, \boldsymbol{\lambda} - \boldsymbol{\mu}\rangle + \langle \boldsymbol{Q}, \boldsymbol{\mu} - \mathbb{E}[\boldsymbol{S} \mid \boldsymbol{Q}]\rangle \\
&= -\varepsilon\|\boldsymbol{Q}_\parallel\| - \sum_{i=1}^N \mu_i \left|\frac{Q_i}{\hat{g}_i} - \frac{Q_1}{\hat{g}_1}\right| \\
&\le -\varepsilon\|\boldsymbol{Q}_\parallel\| - \mu_{\min} \sum_{i=1}^N \left|\frac{Q_i}{\hat{g}_i} - \frac{Q_1}{\hat{g}_1}\right|,
\end{aligned} \tag{4.21}$$

where $\mu_{\min} := \min_i \mu_i$.

Since $0 < \hat{g}_i < 1$ for all i, we know $\hat{g}_i^2 < \hat{g}_i$, and thus

$$\sum_{i=1}^{N} \left| \frac{Q_i}{\hat{g}_i} - \frac{Q_1}{\hat{g}_1} \right| \geq \sqrt{\sum_{i=1}^{N} \left(\frac{Q_i}{\hat{g}_i} - \frac{Q_1}{\hat{g}_1} \right)^2} \geq \left\| \boldsymbol{Q} - \frac{Q_1}{\hat{g}_1} \hat{\boldsymbol{g}} \right\|.$$

Further, we know $\|\boldsymbol{Q} - t\hat{\boldsymbol{g}}\| \geq \|\boldsymbol{Q}_\perp\|$ for all $t \in \mathbb{R}$. Hence,

$$\begin{aligned} \mathbb{E}[\langle \boldsymbol{Q}, \boldsymbol{A} - \boldsymbol{S} \rangle \mid \boldsymbol{Q}] &\leq -\varepsilon \|\boldsymbol{Q}_\parallel\| - \mu_{\min} \left\| \boldsymbol{Q} - \frac{Q_1}{\hat{g}_1} \hat{\boldsymbol{g}} \right\| \\ &\leq -\varepsilon \|\boldsymbol{Q}_\parallel\| - \mu_{\min} \|\boldsymbol{Q}_\perp\| \\ &\leq -\varepsilon \|\boldsymbol{Q}_\parallel\| - \delta \|\boldsymbol{Q}_\perp\|, \end{aligned} \tag{4.22}$$

for any δ such that $0 < \delta < \min_i \lambda_i$.

Substituting Eq. (4.22) to Eq. (4.20), we get

$$\mathbb{E}[\Delta W(t) \mid \boldsymbol{Q}] \leq -2\varepsilon \|\boldsymbol{Q}_\parallel\| - 2\delta \|\boldsymbol{Q}_\perp\| + K_1. \tag{4.23}$$

Next, we obtain a lower bound of $\Delta W_\parallel(t)$. Consider $\mathbb{E}\left[\Delta W_\parallel(t) \,\middle|\, \boldsymbol{Q}\right] := \mathbb{E}\left[\Delta W_\parallel(t) \,\middle|\, \boldsymbol{Q}(t) = \boldsymbol{Q}\right]$. Let $\boldsymbol{\Psi}(t)$ be the unused service at time t such that $\boldsymbol{Q}(t+1) = \boldsymbol{Q}(t) + \boldsymbol{A}(t) - \boldsymbol{S}(t) + \boldsymbol{\Psi}(t)$. Note that $0 \leq \psi_i \leq 1$ for all i.

$$\begin{aligned} \mathbb{E}\left[\Delta W_\parallel(t) \,\middle|\, \boldsymbol{Q}\right] =& \mathbb{E}\left[\langle \hat{\boldsymbol{g}}, \boldsymbol{Q} + \boldsymbol{A} - \boldsymbol{S} + \boldsymbol{\Psi} \rangle^2 - \langle \hat{\boldsymbol{g}}, \boldsymbol{Q} \rangle^2 \,\middle|\, \boldsymbol{Q}\right] \\ =& \mathbb{E}\left[2\langle \hat{\boldsymbol{g}}, \boldsymbol{Q} \rangle \langle \hat{\boldsymbol{g}}, \boldsymbol{A} - \boldsymbol{S} \rangle + \langle \hat{\boldsymbol{g}}, \boldsymbol{A} - \boldsymbol{S} \rangle^2 \right. \\ & \left. + 2\langle \hat{\boldsymbol{g}}, \boldsymbol{Q} + \boldsymbol{A} - \boldsymbol{S} \rangle \langle \hat{\boldsymbol{g}}, \boldsymbol{\Psi} \rangle + \langle \hat{\boldsymbol{g}}, \boldsymbol{\Psi} \rangle^2 \boldsymbol{Q} \right] \\ \geq& 2\langle \hat{\boldsymbol{g}}, \boldsymbol{Q} \rangle \langle \hat{\boldsymbol{g}}, \boldsymbol{\lambda} - \mathbb{E}[\boldsymbol{S} \mid \boldsymbol{Q}] \rangle \\ & - 2\mathbb{E}[\langle \hat{\boldsymbol{g}}, \boldsymbol{S} \rangle \langle \hat{\boldsymbol{g}}, \boldsymbol{\Psi} \rangle \mid \boldsymbol{Q}] \\ \geq& 2\langle \hat{\boldsymbol{g}}, \boldsymbol{Q} \rangle \langle \hat{\boldsymbol{g}}, \boldsymbol{\lambda} - \mathbb{E}[\boldsymbol{S} \mid \boldsymbol{Q}] \rangle - K_2, \end{aligned} \tag{4.24}$$

where $K_2 := 2N^2$ considering $S_i \leq 1$ and $\psi_i \leq 1$ for all i. The first term can be further reduced

as follows:

$$2\langle \hat{\boldsymbol{g}}, \boldsymbol{Q}\rangle\langle \hat{\boldsymbol{g}}, \boldsymbol{\lambda} - \mathbb{E}[\boldsymbol{S} \mid \boldsymbol{Q}]\rangle = 2\|\boldsymbol{Q}_{\|}\|(\Lambda - \mu) = -2\varepsilon\|\boldsymbol{Q}_{\|}\|.$$

Therefore,

$$\mathbb{E}\left[\Delta W_{\|}(t) \,\middle|\, \boldsymbol{Q}\right] \geq -2\varepsilon\|\boldsymbol{Q}_{\|}\| - K_2. \tag{4.25}$$

By taking expectation of Eq. (4.18), and substituting Eq. (4.23) and (4.25) into it, we have

$$\mathbb{E}[\Delta V_{\perp}(t) \mid \boldsymbol{Q}] \leq -\delta + \frac{K_1 + K_2}{2\|\boldsymbol{Q}_{\perp}\|},$$

which establishes the negative drift of $\mathbb{E}[\Delta V_{\perp}(t) \mid \boldsymbol{Q}]$. Along with the absolute boundness provided by Lemma 7, we can conclude that the conditions for Lemma 1 of [72] are satisfied, and thus there exists a sequence of finite integers $\{N_r\}$ such that $\mathbb{E}\left[\left\|\bar{\boldsymbol{Q}}_{\perp}^{(\varepsilon)}\right\|^r\right] \leq N_r$ for all $r = 1, 2, \ldots$. □

Remark: Since the constants in these bounds are all independent of ε, the deviation of the limiting queue length vector $\bar{\boldsymbol{Q}}^{(\varepsilon)}$ from the target queue length vector $\boldsymbol{g}$ becomes negligible as $\varepsilon \to 0$. Therefore, we observe the state space collapse behavior of relative queue lengths, and the efficient delay allocation rule is enforced by our MRQ scheduling policy in the heavy traffic regime.

4.6 Distributed Rate Control Protocol

Theorem 12 has shown that our proposed delay allocation rule in Section 4.4 is efficient. That is, suppose there is a unique vector $\boldsymbol{\lambda}^* = [\lambda_i^*]$ that maximizes total net utility $\sum_i U_i(\lambda_i) - \Lambda C(\Lambda)$ in Eq. (4.1), then $\boldsymbol{\lambda}^*$ is also the unique vector of Nash Equilibrium under our delay allocation rule. Theorem 13 further proves that our MRQ scheduling policy enforces the delay allocation rule, that is each client experiences its own allocated delay in the heavy traffic regime. In this section, we propose a distributed rate control protocol for clients to dynamically adjust their rates so as to converge to the Nash Equilibrium.

Our protocol is based on the projected gradient method [73], a simple yet effective method to solve convex optimization problems. The projected gradient method consists of two steps: initialization and iterative update. In the initialization step, the method arbitrarily chooses a vector

$\boldsymbol{\lambda}(0) \in \mathcal{S}_{\boldsymbol{\lambda}}$. Recall that $\mathcal{S}_{\boldsymbol{\lambda}}$ is the feasible region for $\boldsymbol{\lambda}$. In each subsequent iteration k, the projected gradient method updates $\boldsymbol{\lambda}$ by:

$$\hat{\boldsymbol{\lambda}}(k+1) = \boldsymbol{\lambda}(k) + \kappa(k)\nabla \left[\sum_{i=1}^{N} U_i(\lambda_i) - \Lambda C(\Lambda)\right],$$
$$\boldsymbol{\lambda}(k+1) = \mathrm{P}(\hat{\boldsymbol{\lambda}}(k+1)),$$

where $\kappa(k) > 0$ is the step size at the k-th iteration, and P is the projection to the convex set $\mathcal{S}_{\boldsymbol{\lambda}}$. Note that the index k of iteration should not be confused with the time slot for scheduling. We assume a *time scale separation*, where rate update happens in a more coarse time scale than scheduling, so that there is sufficient time for the scheduling policy to steer the clients and enforce the efficient delay allocation rule. [73] has shown that the projected gradient method converges to the unique optimal solution, and therefore also converges to the Nash Equilibrium.

Proposition 2. *If $\kappa(k)$ satisfies $\sum_{k=0}^{\infty} \kappa(k) = \infty$ and $\sum_{k=0}^{\infty} \kappa^2(k) < \infty$, then the projected gradient method either stops at some iteration k, or the infinite sequence $\{\boldsymbol{\lambda}(k)\}$ generated by the method converges to the optimal point.*

Note that stopping at some iteration k means the method reaches the optimality in finite steps. However, the projected gradient method is a centralized algorithm. In particular, calculating the projection $\boldsymbol{\lambda}(k+1) = \mathrm{P}(\hat{\boldsymbol{\lambda}}(k+1))$ requires the knowledge of all elements in $\hat{\boldsymbol{\lambda}}(k+1)$. Below, we propose a distributed rate control protocol that is inspired by the projected gradient method.

Since

$$\frac{\partial}{\partial \lambda_i}[\Lambda C(\Lambda)] = \frac{\mathrm{d}[\Lambda C(\Lambda)]}{\mathrm{d}\Lambda}\frac{\partial \Lambda}{\partial \lambda_i} = \frac{\mathrm{d}}{\mathrm{d}\Lambda}[\Lambda C(\Lambda)],$$

$\hat{\boldsymbol{\lambda}}(k+1)$ can be acquired by each client updating its own request rate:

$$\hat{\lambda}_i(k+1) = \lambda_i(k) + \kappa(k)\left[U_i'(\lambda_i(k)) - \frac{\mathrm{d}[\Lambda C(\Lambda)]}{\mathrm{d}\Lambda}\right].$$

Note that, to facilitate the update, the server only needs to broadcast the value of $\kappa(k)$ and $\frac{\mathrm{d}[\Lambda C(\Lambda)]}{\mathrm{d}\Lambda}$

in each iteration to all clients.

To ensure that $\boldsymbol{\lambda}(k+1)$ satisfies $\Lambda(k+1) \leq (1-\epsilon)\mu$ and $\lambda_i(k+1) \geq \lambda_\delta$, each client i further chooses

$$\lambda_i(k+1) = \min\{\max\{\hat{\lambda}_i(k+1), \lambda_\delta\}, \lambda_i(k)\frac{(1-\epsilon)\mu}{\Lambda(k)}\}.$$

This step ensures that $\lambda_\delta \leq \lambda_i(k+1) \leq \lambda_i(k)\frac{(1-\epsilon)\mu}{\Lambda(k)}$, and therefore $\Lambda(k+1) \leq \Lambda(k)\frac{(1-\epsilon)\mu}{\Lambda(k)} = (1-\epsilon)\mu$. We also note that, to facilitate this step, the server only needs to broadcast the value of $\Lambda(k)$ in each iteration. Figure 4.2 illustrates the different projection behaviors of the centralized projected gradient method and our distributed rate control protocol. Note that distributed projection requires the constraints of the optimization problem are either decoupled for each client or in a summation form, while centralized projection works with a general convex set as the feasible region.

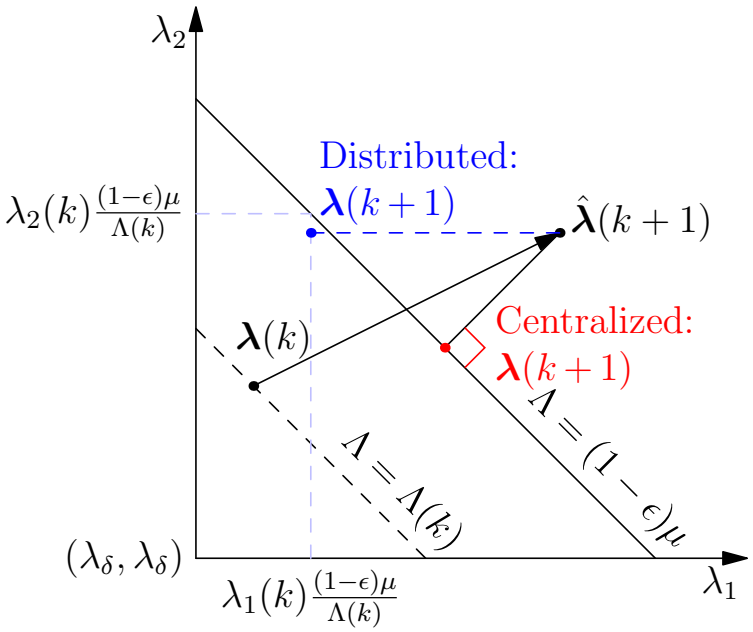


Figure 4.2: Centralized vs. distributed projection. Reprinted with permission from [71].

The complete distributed protocol is summarized in Protocol 1. Compared with the centralized method, our distributed protocol is more scalable and lightweight, since it utilizes the broadcast

nature of wireless channel and requires less resource of the server and the channel.

Protocol 1: Distributed rate control protocol

Server: on convergence of relative queue lengths:

1. $k \leftarrow k+1$
2. Broadcast $\Lambda(k)$, $\kappa(k)$, and $\frac{\mathrm{d}[\Lambda C(\Lambda)]}{\mathrm{d}\Lambda}$

Client i: on reception of server broadcast message:

1. Update: $\hat{\lambda}_i \leftarrow \lambda_i + \kappa(k)\left[U_i'(\lambda_i) - \frac{\mathrm{d}[\Lambda C(\Lambda)]}{\mathrm{d}\Lambda}\right]$
2. Projection: $\lambda_i \leftarrow \min\{\max\{\hat{\lambda}_i, \lambda_\delta\}, \lambda_i \frac{(1-\epsilon)\mu}{\Lambda}\}$

We can prove that our distributed protocol also converges to the Nash Equilibrium. This property will also be verified by simulations in Section 4.8.

Theorem 14. *If $\kappa(k)$ satisfies $\sum_{k=0}^{\infty}\kappa(k) = \infty$ and $\sum_{k=0}^{\infty}\kappa^2(k) < \infty$, then the distributed rate control protocol either stops at some iteration k, or the infinite sequence $\{\boldsymbol{\lambda}(k)\}$ generated by the protocol converges to the Nash Equilibrium of the system.*

Proof. See Appendix B.2. □

4.7 Non-Monetary Protocol with Efficient Loss Rate Allocation

Our non-monetary mechanism can be extended to deal with different costs other than delay itself. In this section, we consider loss rate allocation in a finite-buffer system as an example. This is more practical for real systems with only small buffers where packet losses are more common, for example, mobile hotspots set up by cellphones.

4.7.1 System Model for the Loss Rate Allocation Problem

Similar to Section 4.3, suppose that there are N clients and a server in the system. Each client i controls its request arrival rate λ_i, and the service time needed by each request is a sequence

of i.i.d. random variables with mean $\frac{1}{\mu}$. On the other hand, we assume that the server serves all requests in a first-in-first-out (FIFO) fashion, and that the server only has a finite buffer that can hold B unfinished requests, including the one being served. When the buffer is full and there is another request arrival, the server needs to drop a request to accommodate the new request, and the corresponding client experiences a loss.[3]

Since the service times of all requests have the same probability distribution, the request loss rate, defined as the average number of dropped requests per unit time, is a function of total request arrival rate, $\Lambda = \sum_i \lambda_i$. We denote the request loss rate by $\bar{L}(\Lambda)$, and note that $\bar{L}(\Lambda) = \Lambda P_B(\Lambda)$, where $P_B(\Lambda)$ is the blocking probability of the queueing system. We assume that $\bar{L}(\Lambda)$ can be well fitted by a low-order polynomial function $L(\Lambda)$, which is strictly increasing and strictly convex.

Each client obtains some utility $U_i(\lambda_i)$ based on its own request rate, and suffers from some disutility that equals its own loss rate. We use $l_i(\lambda_i, \lambda_{-i})$ to denote the loss rate of client i. Hence, the net utility of client i is $U_i(\lambda_i) - l_i(\lambda_i, \lambda_{-i})$.

Obviously, we have $\sum_i l_i(\lambda_i, \lambda_{-i}) = \bar{L}(\Lambda) \approx L(\Lambda)$. The goal of the server is to maximize the total net utility in the system, which can be approximated by $\sum_i U_i(\lambda_i) - L(\Lambda)$, while each client i aims to maximize its own net utility $U_i(\lambda_i) - l_i(\lambda_i, \lambda_{-i})$. The server can allocate the loss rate $l_i(\lambda_i, \lambda_{-i})$ of each client i through its policy of dropping requests, subject to the constraint that $\sum_i l_i(\lambda_i, \lambda_{-i}) = L(\Lambda)$.

Similar to delay allocation, we can define Nash Equilibrium and efficient allocation rule for loss rate allocation as follows:

Definition 12. *A vector $\tilde{\boldsymbol{\lambda}} := [\tilde{\lambda}_i]$ is said to be a Nash Equilibrium for loss rate allocation if $\tilde{\lambda}_i = \text{argmax}_{\lambda_i} U_i(\lambda_i) - l_i(\lambda_i, \tilde{\lambda}_{-i}), \forall i$.*

Definition 13. *A rule of allocating loss rates, $[l_i(\cdot)]$, is said to be* efficient *if $\boldsymbol{\lambda}^*$ is the only Nash Equilibrium.*

[3]The dropped request can be the newly arriving one, or some request already in the buffer.

4.7.2 Mechanism Design for Efficient Loss Rate Allocation

Our results of efficient allocation rule in Section 4.4 can be easily extended to loss rate allocation. In particular, we have the following proposition:

Proposition 3. $[l_i(\cdot)]$ *is efficient if and only if there exists functions* $R_i : \mathbb{R}^{N-1} \to \mathbb{R}$ *such that for all* i*,*

$$l_i(\lambda_i, \lambda_{-i}) = L(\Lambda) - R_i(\lambda_{-i}), \tag{4.26}$$

and

$$\sum_{i=1}^{N} l_i(\lambda_i, \lambda_{-i}) = L(\Lambda). \tag{4.27}$$

For the allocation rule, redefine c_i to be the coefficients of $L(\Lambda)$ instead of $\Lambda C(\Lambda)$ in Section 4.4. Then setting

$$l_i(\lambda_i, \lambda_{-i}) = L(\Lambda) - R_i(\lambda_{-i}), \tag{4.28}$$

is efficient, where $R_i(\lambda_{-i})$ has the same form as in (4.14).

Next, we discuss how to design a policy that ensures the actual perceived loss rate of each client i is close to the desirable $l_i(\lambda_i, \lambda_{-i})$. Suppose at time t, the server's buffer is full and one more client request arrives. Let $\bar{l}_i(t)$ be the perceived loss rate of client i till time t for all i. On the other hand, l_i is the allocated loss rate according to the above allocation rule. We propose the following drop-smallest-relative-loss-rate (DropSRLR) policy:

Definition 14 (DropSRLR)**.** *Suppose the server's buffer is full and a new request arrives at time* t*, the DropSRLR policy drops a request from the client with the smallest relative loss rate, defined as* $\bar{l}_i(t)/l_i$*, breaking ties by choosing the client with the lowest ID.*

The intuition of our dropping policy is that by always selecting the client with the smallest relative loss rate, over a long term all relative loss rates tend to be the same, which is equivalent to say each client obtains a loss rate as allocated. The efficiency of the policy will be demonstrated in the simulations in Section 4.8.

Moreover, we can extend our distributed rate control protocol to loss rate allocation. The complete distributed protocol is summarized in Protocol 2. Note that there is no upper limit for the total request rate to make the finite-buffer system stable. Therefore, the distributed protocol is essentially the same as its centralized counterpart, and its convergence is straightforward to show.

Protocol 2: Distributed rate control protocol for loss rate allocation

Server: on convergence of relative loss rates:

1. $k \leftarrow k+1$
2. Broadcast $\kappa(k)$ and $L'(\Lambda(k))$

Client i: on reception of server broadcast message:

1. Update: $\hat{\lambda}_i \leftarrow \lambda_i + \kappa(k)\left[U_i'(\lambda_i) - L'(\Lambda(k))\right]$
2. Projection: $\lambda_i \leftarrow \max\{\hat{\lambda}_i, \lambda_\delta\}$

4.8 Simulations

In this section, we evaluate the performance of our overall design via simulations. We will present the simulations for delay allocation and loss rate allocation respectively.

4.8.1 Simulations of Delay Allocation

For delay allocation, we validate the polynomial approximation assumption for the average delay function, the state space collapse behavior of relative queue lengths through the MRQ scheduling policy, and the convergence to the Nash Equilibrium of our distributed rate control protocol. For comparison, we also consider a baseline mechanism with the classic FIFO policy for scheduling and centralized projected gradient method for rate control. Note that with FIFO scheduling, each client experiences the same average delay, i.e. $D_i(\lambda_i, \lambda_{-i}) = C(\Lambda)$.

In our simulations, we consider two systems each with $N = 10$ clients and one server. Both systems have Poisson arrivals of requests from all clients. The service time distribution of one system is exponential, and the other is deterministic. Hence, the two systems correspond to an

M/M/1 queue and an M/D/1 queue respectively. Each system has an average service rate $\mu = 1 \times 10^3\,\mathrm{s}^{-1}$ and an initial total average request rate $\Lambda = 0.95\mu = 0.95 \times 10^3\,\mathrm{s}^{-1}$. Request rates will be updated by clients over time. We round up all inter-arrival times between two consecutive requests and service times of requests to the nearest microsecond. Given the above average service rate, we make about 10^3 scheduling decisions every second.

4.8.1.1 Polynomial Approximation of Average Delay Function

First, we evaluated the assumption that the average delay function can be well approximated by a polynomial $C(\Lambda)$. There are two methods to obtain the average delay function: One is via the theoretical formula, and the other is via simulations. Here, we use the first method. For the M/M/1 queue, the theoretical average delay function is:

$$\bar{C}(\Lambda) = \frac{1}{\mu - \Lambda}.$$

For the M/D/1 queue, it is:

$$\bar{C}(\Lambda) = \frac{1}{\mu} + \frac{\Lambda}{2\mu(\mu - \Lambda)}.$$

In our simulations, we fit $\bar{C}(\Lambda)$ with ten samples in our most interested heavy traffic region, where $\Lambda/\mu \in [0.9, 0.99]$, to get the polynomial $C(\Lambda)$. Recall that the total disutility in terms of total average queue length is $\Lambda C(\Lambda)$. The total disutility functions before and after approximation are compared in Figure 4.3, labeled as "Theory" and "Approx" respectively. We can observe that the polynomial approximation fits the theoretical functions very well. In fact, the order of the polynomial $C(\Lambda)$ is as small as six, and the largest relative error of the approximation is only about 2.66%.

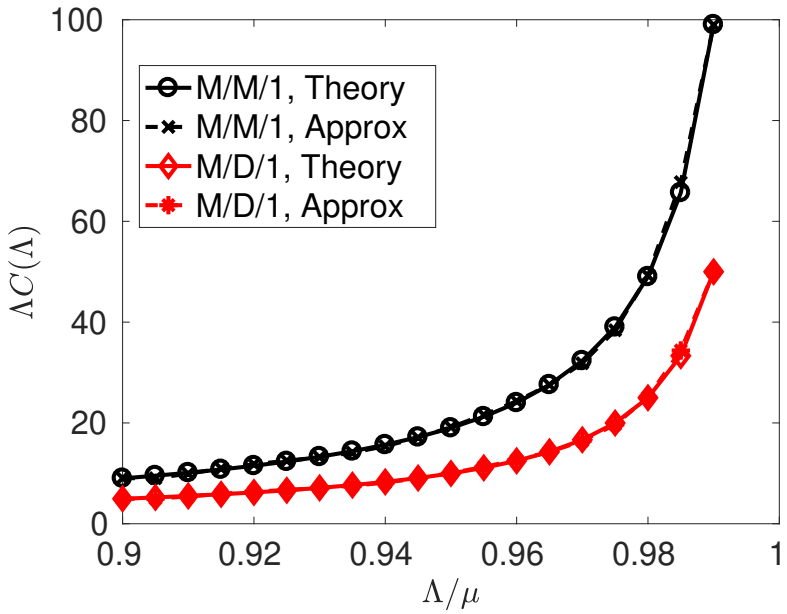


Figure 4.3: Polynomial approximation of total disutility functions $\Lambda C(\Lambda)$. Reprinted with permission from [71].

4.8.1.2 Scheduling Policy

We implemented our MRQ scheduling policy and validated the state space collapse behavior in the simulations. We use a new metric, *the relative difference of queue lengths*, defined as:

$$\left(\max_i \frac{Q_i(t)}{g_i} - \min_i \frac{Q_i(t)}{g_i}\right) \Big/ \sum_i \frac{Q_i(t)}{g_i}$$

to evaluate the state space collapse performance. Theorem 13 has shown that, given the target queue length g_i of each client i, our MRQ policy ensures that the relative difference of queue lengths converges to 0 in the heavy traffic regime.

Figure 4.4 shows the evolution of the relative difference of queue lengths for both systems for two sets of initial request rates, "Same rate" and "Diff rates". "Same rate" means all ten clients have the same request rate $\lambda = \Lambda/N = 95\,\mathrm{s}^{-1}$, while in "Diff rates" we have two groups of request rates: $\lambda_i = 95.6\,\mathrm{s}^{-1}$ for $i = 1, 2, \ldots, 5$ and $94.4\,\mathrm{s}^{-1}$ for $i = 6, 7, \ldots, 10$. We initialize the queue length of client i to be i^2 to exhibit the convergence of relative queue lengths more clearly. We can

see that the relative difference of queue lengths converges to 0 quickly for each scenario.

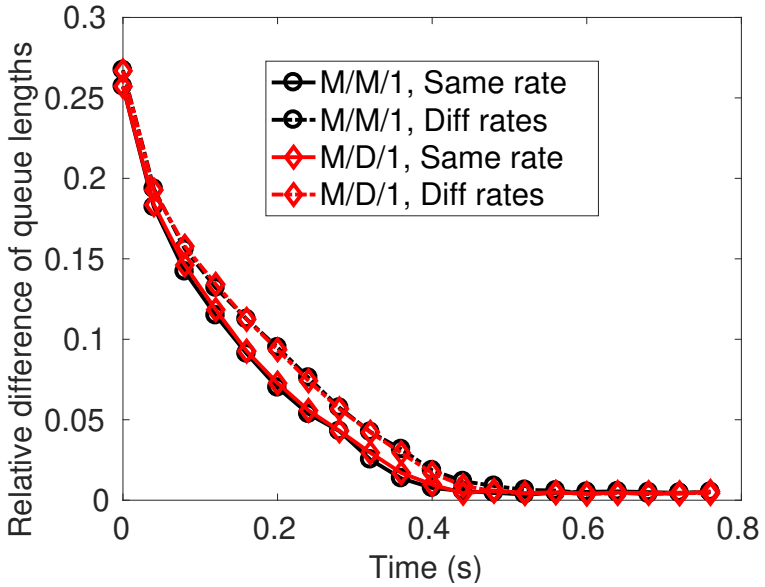


Figure 4.4: State space collapse of relative queue lengths. Reprinted with permission from [71].

Figure 4.5 depicts the state space collapse behavior in light traffic, where the total load of each system is only 0.1. Here "Same rate" means all ten clients have the same request rate $\lambda = \Lambda/N = 10\,\mathrm{s}^{-1}$, while in "Diff rates" the two groups of request rates are: $\lambda_i = 10.6\,\mathrm{s}^{-1}$ for $i = 1, 2, \ldots, 5$ and $9.4\,\mathrm{s}^{-1}$ for $i = 6, 7, \ldots, 10$. Similar to Figure 4.4, the queue length of client i is initialized to be i^2. We observe that the relative difference of queue lengths also quickly decreases to a low level initially where there are enough requests to schedule. However, there is no further decrease afterwards since too few requests are in the system to achieve exact allocation of queue lengths and delays.

4.8.1.3 Nash Equilibrium

Furthermore, we evaluated our distributed rate control protocol in the simulations. We set the utility functions for both systems to be $U_i(\lambda_i) = \alpha w_i \log \lambda_i$, where $\alpha = 100$ is the common scaling coefficient for all clients, and w_i's are different weights for different clients. We set the weights

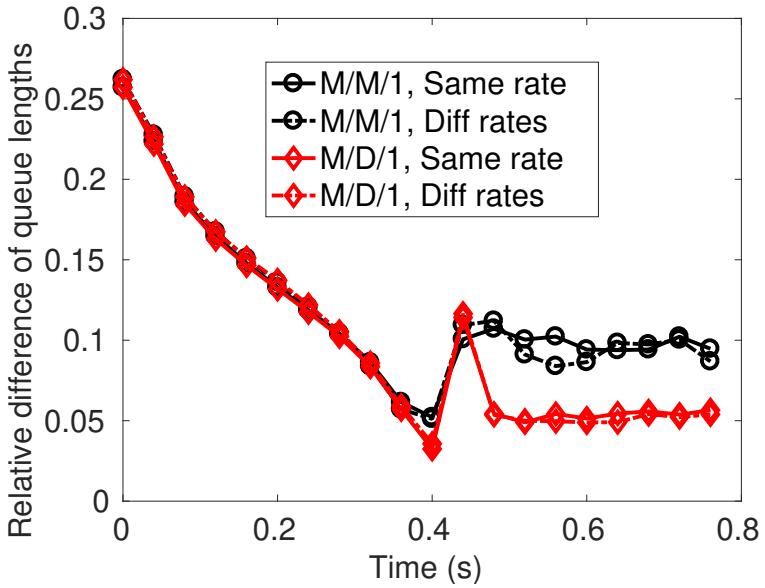


Figure 4.5: State space collapse in light traffic. Reprinted with permission from [71].

to be in two groups: $w_i = 0.99$ for $i = 1, 2, \ldots, 5$ and 1.01 for $i = 6, 7, \ldots, 10$. Therefore, the evolution of request rates of all the clients can be captured by those of Client 1 and Client 10. For the step size, we let $\kappa(k) = 10/k$ for all k.

Figure 4.6 shows the rate convergence performance for the two systems respectively. We can see that for each system, the request rates converge to two distinct values after tens of iterations. Observe that the distributed rate control protocol ("Dist" in the figure) has almost the same rate updates as the projected centralized gradient method ("Cent" in the figure). It validates that the request rates converge to the optimal rates $\boldsymbol{\lambda}^*$, and the distributed rate control protocol achieves the Nash Equilibrium of the system.

Figure 4.7 shows the convergence performance in terms of total net utility for the two systems. The total net utility settles down quickly with our distributed protocol ("MRQ, Dist" in the figures), and the evolution is again almost the same as the centralized method ("MRQ, Cent" in the figures). It means the total net utility converges to the optimal value of the optimization problem, and confirms the convergence of our distributed rate control protocol. In these figures we also plot the performance of the baseline mechanism with the FIFO scheduling policy for comparison. We

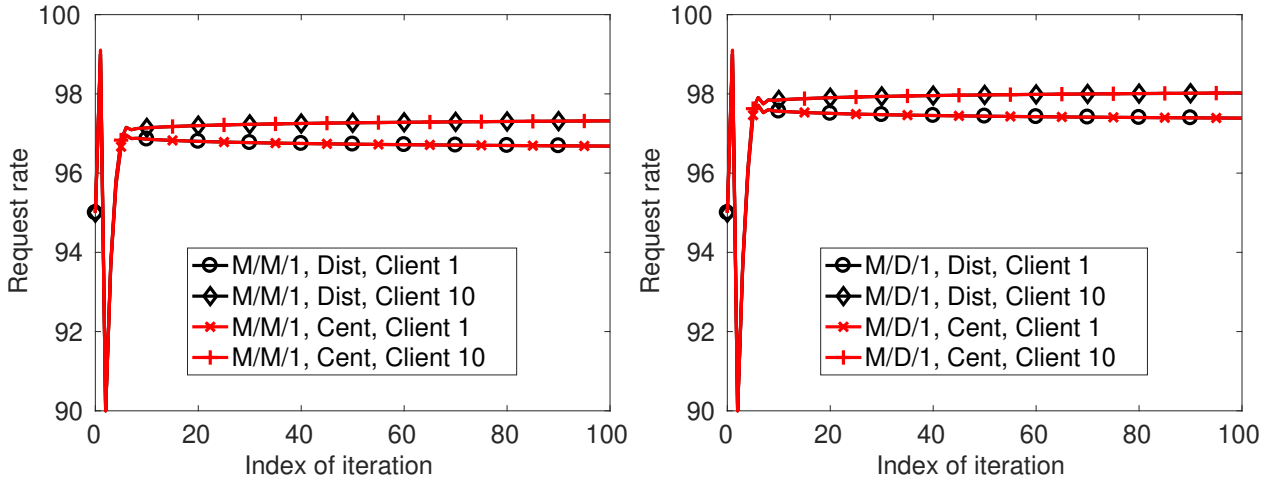
can see that under the baseline mechanism, the total net utility converges to a suboptimal value. It indicates that the delay allocation rule of the baseline mechanism is not efficient.

We also conduct preliminary studies on the impact of VIP clients via simulations. We assume VIP clients experience zero delay and update their request rates accordingly. We consider two scenarios where VIP clients exist. One is that there are VIP clients at the Nash Equilibrium. In the simulation, we set the weights in the utility functions to be $w_i = 0.7$ for $i = 1, 2, \ldots, 5$ and 1.3 for $i = 6, 7, \ldots, 10$ so that Clients 1–5 will be VIP at the Nash Equilibrium. Figure 4.8 depicts the evolution of total net utility for the M/M/1 system in this scenario. Observe that under our protocol, the total net utility oscillates over time. However, our mechanism still outperforms the baseline mechanism. The other scenario uses the same utility functions as those in Figure 4.7, but sets the initial request rates to be $\lambda_i = 100\,\mathrm{s}^{-1}$ for $i = 1, 2, \ldots, 5$ and $90\,\mathrm{s}^{-1}$ for $i = 6, 7, \ldots, 10$ so that initially Clients 6–10 are VIP. We find that Clients 6–10 remain VIP clients in the first two iterations. However, all clients are non-VIP afterwards. Figure 4.9 shows the convergence performance of total net utility for the M/M/1 system. Note that it converges to the same optimal value as in Figure 4.7a under our mechanism. Therefore, in this case the system eventually converges to the original optimal Nash Equilibrium.

4.8.2 Simulations of Loss Rate Allocation

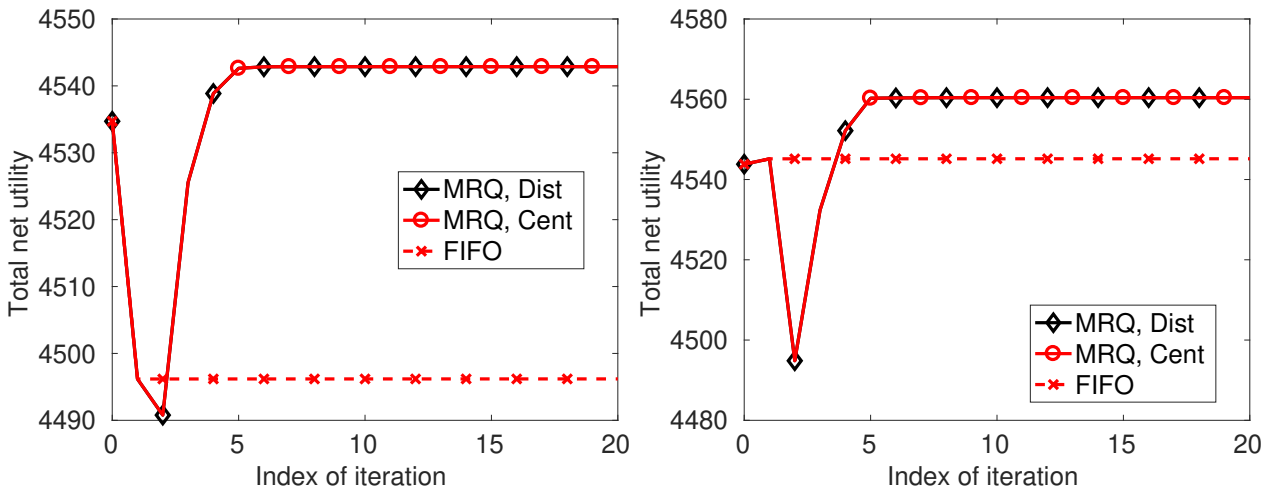
For loss rate allocation, we will show the validity of polynomial approximation for the loss rate function, the convergence of relative loss rates under our DropSRLR policy, and the convergence of the distributed rate control protocol in Protocol 2. As for the baseline mechanism, we use the well-known DropTail policy that always drops the newly arriving request when the buffer is full. Note that under DropTail, each client has the same blocking probability, and thus $l_i(\lambda_i, \lambda_{-i}) = \lambda_i L(\Lambda)/\Lambda$.

Similar to delay allocation, we simulate two systems each with $N = 10$ clients and one server for loss rate allocation. The two systems correspond to an M/M/1/B queue and an M/D/1/B queue respectively. That is to say, the request arrival processes are both Poisson, while the service time distributions are exponential and deterministic respectively. The buffer size B is fixed to be 10 for



(a) M/M/1 system. (b) M/D/1 system.

Figure 4.6: Convergence performance of request rates for delay allocation. Reprinted with permission from [71].



(a) M/M/1 system. (b) M/D/1 system.

Figure 4.7: Convergence performance of total net utility for delay allocation. Reprinted with permission from [71].

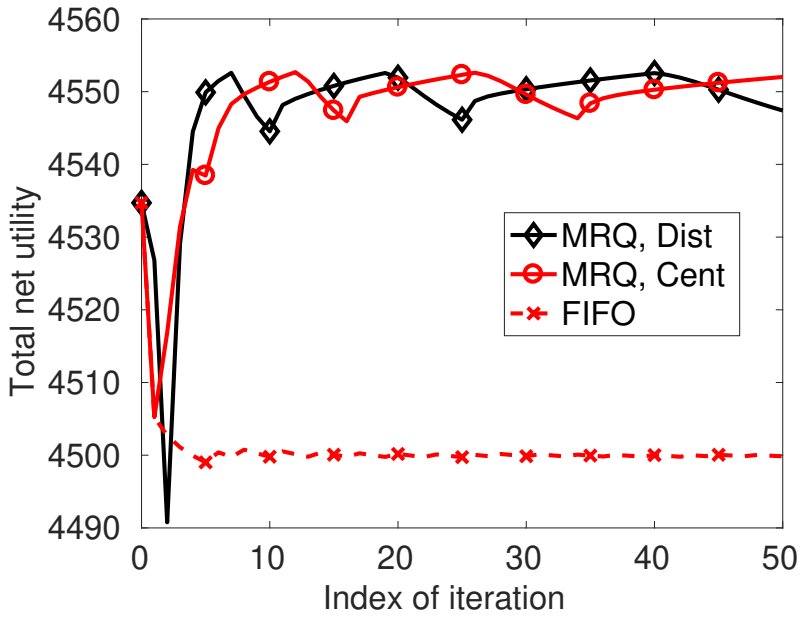


Figure 4.8: Total net utility evolution with VIP clients at the Nash Equilibrium. Reprinted with permission from [71].

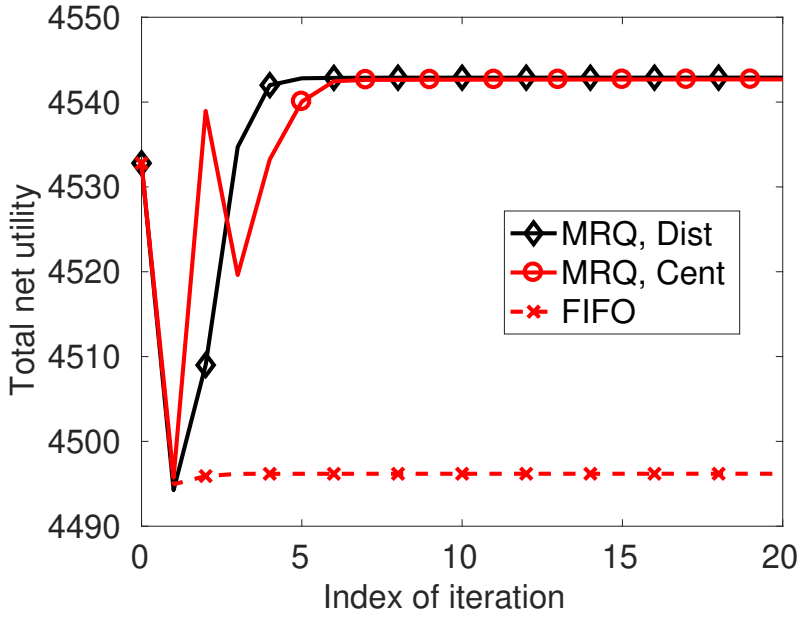


Figure 4.9: Total net utility convergence with VIP clients at initial arrival. Reprinted with permission from [71].

each system. Besides, we set the average service rate $\mu = 1 \times 10^3\,\mathrm{s}^{-1}$, and the initial total average request rate $\Lambda = 0.99\mu = 0.99 \times 10^3\,\mathrm{s}^{-1}$.

4.8.2.1 Polynomial Approximation of Loss Rate Function

First, we evaluated the assumption that the loss rate function can be well approximated by a polynomial $L(\Lambda)$. Similar to delay allocation, we use theoretical results to obtain the loss rate function $\bar{L}(\Lambda)$. Recall that $\bar{L}(\Lambda) = \Lambda P_B(\Lambda)$. For the M/M/1/B queue, the blocking probability $P_B(\Lambda)$ is given by the following formula:

$$P_B(\Lambda) = \frac{(\frac{\Lambda}{\mu})^B}{\sum_{i=0}^{B}(\frac{\Lambda}{\mu})^i}.$$

For the M/D/1/B queue, $P_B(\Lambda)$ can be calculated by the procedure described in [74]. Therefore, we can get $P_B(\Lambda)$ and $\bar{L}(\Lambda)$ for any given Λ.

In our simulations, we fit $P_B(\Lambda)$ with ten samples where $\Lambda/\mu \in [0.7, 1.3]$ to be a 6-order polynomial. The total disutility functions in terms of loss rate $L(\Lambda)$ before and after approximation are compared in Figure 4.10, labeled as "Theory" and "Approx" respectively. Similar to delay allocation, the polynomial approximation can be observed to match the theoretical functions very well. The largest relative error is only about 1.57%.

4.8.2.2 Dropping Policy

We implemented our DropSRLR dropping policy for loss rate allocation and validated the convergence of relative loss rates via simulations. To quantify the convergence performance, we introduce *the relative difference of loss rates*, defined as

$$\left(\max_i \frac{\bar{l}_i(t)}{l_i} - \min_i \frac{\bar{l}_i(t)}{l_i}\right) \Big/ \sum_i \frac{\bar{l}_i(t)}{l_i}.$$

Figure 4.11 shows the evolution of the relative difference of loss rates for both systems for two sets of initial request rates. Similar to delay allocation, "Same rate" means all ten clients have the same request rate $\lambda = \Lambda/N = 99\,\mathrm{s}^{-1}$. On the other hand, for "Diff rates" in loss rate allocation we set

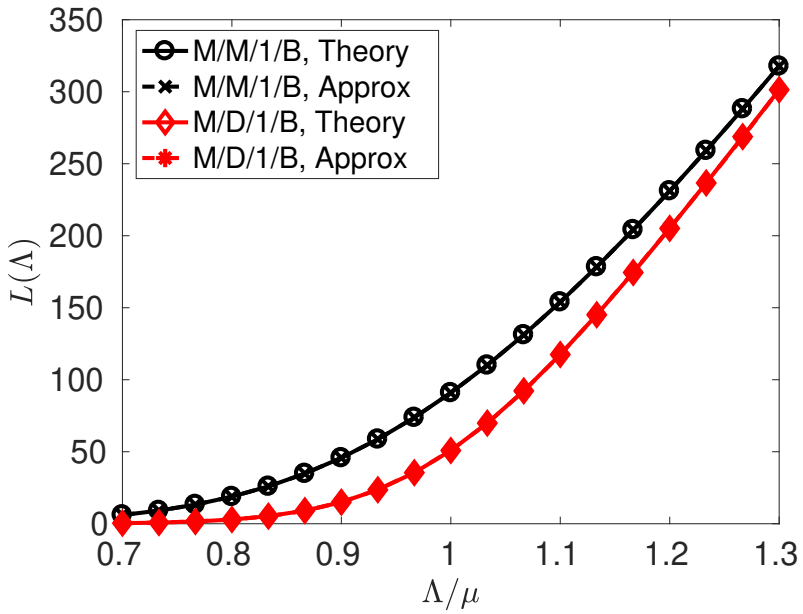


Figure 4.10: Polynomial approximation of loss rate functions. Reprinted with permission from [71].

$\lambda_i = 100\,\mathrm{s}^{-1}$ for $i = 1, 2, \ldots, 5$ and $98\,\mathrm{s}^{-1}$ for $i = 6, 7, \ldots, 10$. The initial loss rate of client i is set to be i. From the figure, we can see that the relative difference of loss rates converges to 0 quickly for both systems and both sets of initial request rates. It shows that our DropSRLR dropping policy ensures that the loss rates experienced are as allocated and the policy is thus efficient.

4.8.2.3 Distributed Protocol

We also validated the convergence of our distributed rate control protocol for loss rate allocation, Protocol 2, in our simulations. Similar to delay allocation, the utility function of client i is $U_i(\lambda_i) = \alpha w_i \log \lambda_i$. We set $\alpha = 50$ as the common scaling coefficient for all clients. We set the weights to be in two groups: $w_i = 1 - 5 \times 10^{-3}$ for $i = 1, 2, \ldots, 5$ and $1 + 5 \times 10^{-3}$ for $i = 6, 7, \ldots, 10$. For the step size, we let $\kappa(k) = 80/k$ for all k.

Figure 4.12 shows the rate convergence performance for the two finite-buffer systems. In our setup, the rate evolution of Client 1 and Client 10 depicts the rate evolution of all the ten clients. We can see that for each system, the request rates converge to two distinct values after tens of iterations. Figure 4.13 shows the convergence performance in terms of total net utility for the two

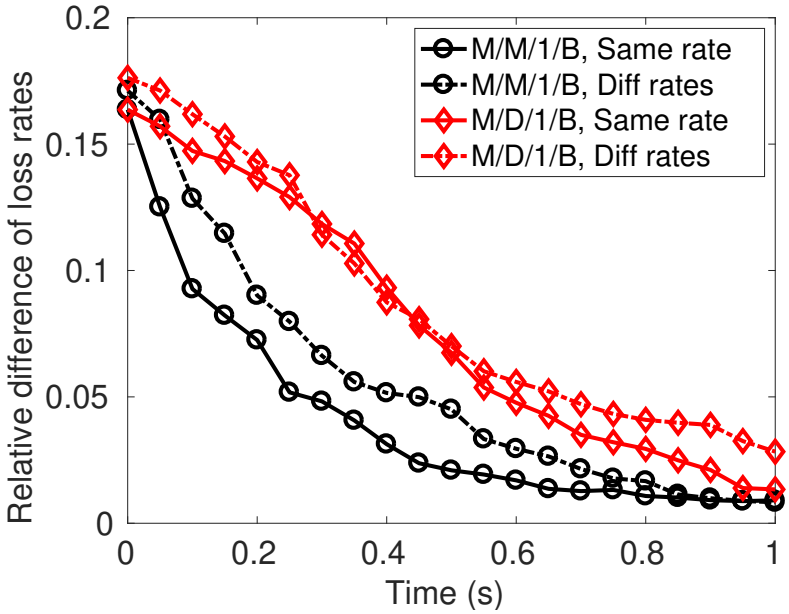


Figure 4.11: Convergence performance of relative loss rates. Reprinted with permission from [71].

systems. The total net utility settles down quickly with our distributed protocol ("DropSRLR" in the figure). Note that the centralized method is omitted since it is essentially the same as the distributed protocol for loss rate allocation. Therefore, under our distributed protocol, the request rates of all clients converge to the Nash Equilibrium, and the total net utility converges to the optimal value of the rate control problem. On the other hand, under the baseline mechanism with the DropTail dropping policy the total net utility converges to a suboptimal value for each system. Hence, the loss rate allocation of the baseline mechanism is not efficient.

We also conduct sensitivity analysis on the buffer size B via simulations. The results are plotted in Figure 4.14, and they clearly show diminishing returns. The total net utility, i.e. the objective value of the rate control problem, increases as the buffer size B increases. This is consistent with the intuition that larger buffer size leads to smaller loss rates. However, the marginal increase in total net utility decreases and the total net utility becomes saturated when B is large.

4.9 Conclusions

We have presented our non-monetary mechanism for optimal rate control through efficient cost allocation to address the challenge of strategic users in Information Centric Edge. First, we

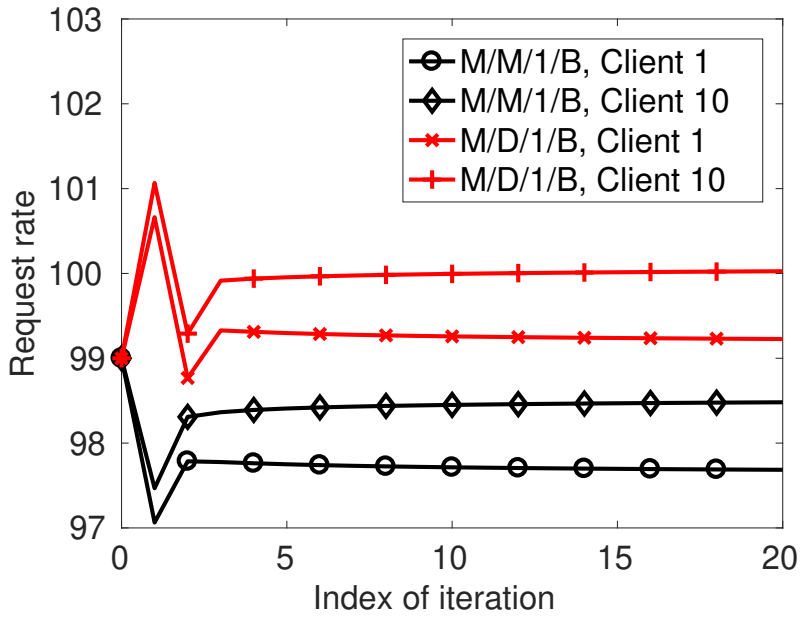


Figure 4.12: Convergence performance of request rates for loss rate allocation. Reprinted with permission from [71].

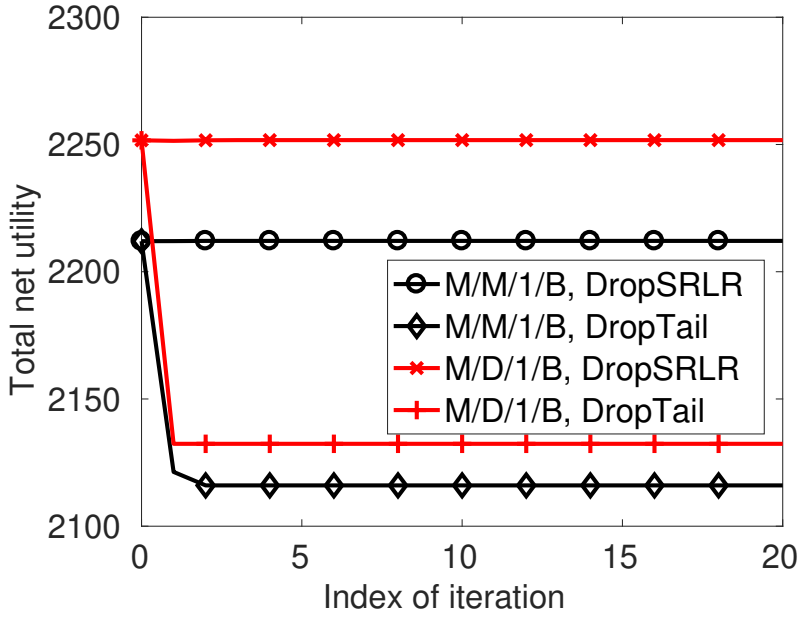


Figure 4.13: Convergence performance of total net utility for loss rate allocation. Reprinted with permission from [71].

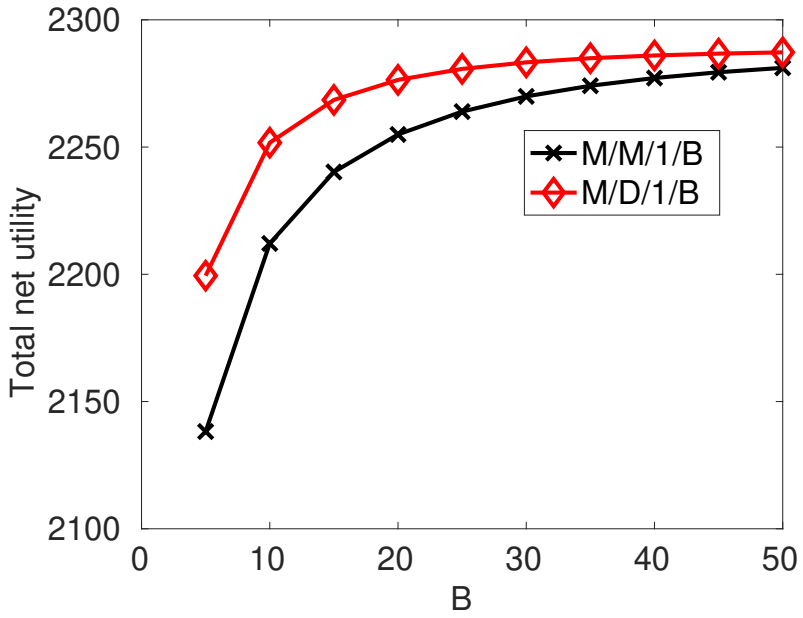


Figure 4.14: Sensitivity on the buffer size B. Reprinted with permission from [71].

focus on delay allocation. We give our delay allocation rule and prove its efficiency based on multinomial expansion. Then we propose our MRQ scheduling policy that can enforce the delay allocation rule effectively in the heavy traffic regime. Besides, we design a distributed rate control protocol which can lead the system to the Nash Equilibrium. Furthermore, we show that our non-monetary mechanism can be extended to handle loss rate allocation as well. Finally, simulation results depict the effectiveness of our mechanism. We will conduct further study on VIP clients for future work. We would like to obtain nontrivial sufficient conditions for clients to become VIPs and for our mechanism to still achieve efficient cost allocation considering VIP clients.

5. EFFICIENT REVENUE DISTRIBUTION*

This chapter presents a revenue distribution method to encourage manufacturing device suppliers to participate in an Internet-based platform enabled by cyber infrastructure. Considering a group of suppliers (e.g. 3D printing service providers) sign up to an e-commerce platform to fulfill the order of customers, it is very much desirable for the platform operator to distribute the revenue in a way that the total net utility of the platform and the net utility of the individual suppliers are optimized. Based on the game theory approach, a nonlinear revenue distribution rule is designed to achieve Nash Equilibrium of the system. Simulation studies show that, compared to the proportional and equal distribution rules, the proposed distribution rule indeed yields the highest net utility for all. While efficient algorithms to reduce computational costs can be further investigated, the present work demonstrates an approach to promote broader participation of manufacturing activities.

5.1 Introduction

Manufacturing is typically a capital and skill intensive industry. With the advances in manufacturing technologies and cyber infrastructure, the barrier-to-entry into manufacturing, however, has been significantly eased. Today, it is possible for a company to establish a manufacturing facility with computer numerical control machines (CNCs) or 3D printers, conduct business through Internet, and supply parts to customers across the country and around the world. Therefore the development of a pervasive manufacturing environment consists of providers and customers of manufacturing technologies, services, and products can be enabled by an Internet-based platform. The platform can be perceived as the *Uber*, *Airbnb* or *Amazon* of manufacturing for time sharing of manufacturing equipment (e.g. 3D printers) and transaction of manufactured goods.

For the success of a pervasive manufacturing environment described above, it is critical that the

*Part of this chapter is reprinted with permission from "An efficient revenue distribution method in a cyber-enabled manufacturing system" by T. Zhao, I.-H. Hou, V. J. Leon, K. Ray, and J. Wang, *Manufacturing Letters*, Copyright 2018 Elsevier

customers seeking for manufacturing services and the participating manufacturers find it beneficial and profitable to use the cyber-enabled platform. This chapter focuses on the interaction between the numerous manufacturers and the platform. One way the platform can encourage manufacturers' participation would be to provide each participant with a suitable portion of the revenue obtained from selling to customers. Specifically, a revenue sharing methodology is presented that illustrates how the platform owner could optimally distribute a portion of its revenue among all manufacturing service suppliers to maximize total net utility achieved by the platform and the manufacturers while at the same time optimizing the individual net utility for each manufacturer.

The literature on revenue-sharing is extensive for traditional supply chains and more recently for supply chains under e-commerce [75, 76, 77, 78]. However, there is notable lack of theoretic publications on this topic with explicit consideration to characteristics unique to manufacturing service suppliers, and even less in the context of cyber-enabled supply chains. This chapter presents initial results aimed at filling this research gap.

5.2 Theory

Assuming convexity, the problem of maximizing the total net utility can be easily solved by convex optimization toolboxes when one has complete information of all the reward and cost functions. In practice, however, the cost functions are private information of suppliers, and a strategic supplier may not reveal its true cost function. Therefore, game theoretical approach is necessary.

Consider a cloud platform that connects a massive number of designers/customers and n manufactured part suppliers, as shown in Fig. 5.1. The platform assigns the arriving tasks to different suppliers to fully utilize available manufacturing equipment/resources. Let μ_i denote the performance of the i-th supplier, which can be measured as the number of basic parts that can be manufactured in unit time. Let $\boldsymbol{\mu} := [\mu_i]$ denote the performance vector of all suppliers, and μ_{-i} denote the performance vector of all suppliers other than the i-th supplier.

From the perspective of the cloud platform, the overall system performance is $\mu_\Sigma := \sum_i \mu_i$. We assume the overall system performance is bounded so that $\mu_\Sigma \in [0, \bar{\mu}_\Sigma]$. Suppose such performance generates a revenue of $R_{\text{tot}}(\mu_\Sigma) := R_0(\mu_\Sigma) + R(\mu_\Sigma)$ to the cloud platform, where $R_0(\mu_\Sigma)$

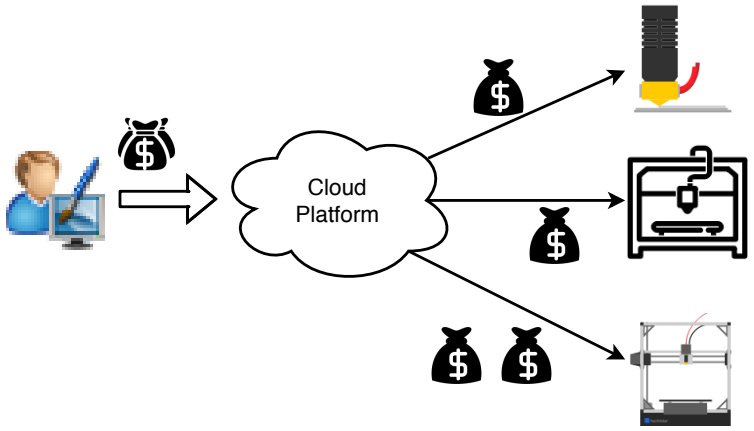


Figure 5.1: Cyber-enabled manufacturing platform.

is the net profit of the platform itself, and $R(\mu_\Sigma)$ is the revenue to be distributed to the n suppliers. For instance, the platform can choose $R_0(\mu_\Sigma)$ to be a certain percentage of $R_{\text{tot}}(\mu_\Sigma)$. It is assumed that the function $R(\cdot)$ is strictly concave, non-decreasing, satisfies $R(0) = 0$, and can be well-fitted by an $(n-1)$-order polynomial $\widehat{R}(\mu_\Sigma)$ via, for example, Chebyshev least squares approximation. Suppose each supplier receives a revenue of $r_i(\mu_i, \mu_{-i})$ after the distribution. As it is desired to maintain (approximate) budget balance for the revenue distribution, $\sum_i r_i(\mu_i, \mu_{-i}) \approx R(\mu_\Sigma)$. The formal definition is as follows:

Definition 15. *A rule of distributing revenue,* $[r_i(\cdot)]$*, is said to* converge to budget balance in n *if*

$$\lim_{n\to\infty} \sup_{\boldsymbol{\mu}} \left| \sum_{i=1}^{n} r_i(\mu_i, \mu_{-i}) - R(\mu_\Sigma) \right| = 0 \tag{5.1}$$

To maintain a certain level of performance μ_i, the supplier i needs to invest on raw materials, energy, device maintenance, and so on, which will incur a cost denoted by $C_i(\mu_i)$. The cost functions are assumed to be convex. Now for supplier i, since the revenue is $r_i(\mu_i, \mu_{-i})$ and the cost is $C_i(\mu_i)$, the *net utility* is therefore $r_i(\mu_i, \mu_{-i}) - C_i(\mu_i)$. Each supplier i aims to maximize its own net utility. As such, the system reaches a Nash Equilibrium if no supplier in the system can improve its own net utility unilaterally.

Definition 16. *A vector $\widetilde{\boldsymbol{\mu}} := [\widetilde{\mu}_i]$ is said to be a Nash Equilibrium if*

$$\widetilde{\mu}_i = \underset{\mu_i}{\operatorname{argmax}}\, r_i(\mu_i, \widetilde{\mu}_{-i}) - C_i(\mu_i), \forall i \tag{5.2}$$

The platform aims to maximize the total net utility in the system, which can be written as $\sum_i r_i(\mu_i, \mu_{-i}) - C_i(\mu_i)$. Due to approximate budget balance, It can be stated that the platform aims to maximize $R\left(\mu_\Sigma\right) - \sum_i C_i\left(\mu_i\right)$. Let $\boldsymbol{\mu}^* := [\mu_i^*]$ be the vector that maximizes the total net utility, then the platform's problem is to find the rule that distributes revenue, $[r_i(\cdot)]$, to induce optimal choices of $[\mu_i]$.

Definition 17. *A rule of distributing revenue, $[r_i(\cdot)]$, is said to be* efficient *if $\boldsymbol{\mu}^*$ is the only Nash Equilibrium.*

Intuition of an efficient revenue distribution rule can be revealed by some basic properties of the optimal vector $\boldsymbol{\mu}^* = [\mu_i^*]$. Since the algorithm maximizes the total net utility $R\left(\mu_\Sigma\right) - \sum_i C_i\left(\mu_i\right)$, it holds that

$$\frac{\partial}{\partial \mu_i} R(\mu_\Sigma^*) = C_i'(\mu_i^*). \tag{5.3}$$

On the other hand, if $\boldsymbol{\mu}^*$ is also the Nash Equilibrium under some revenue distribution rule $[r_i(\cdot)]$, then μ_i^* maximizes $r_i(\mu_i, \mu_{-i}^*) - C_i(\mu_i)$, and thus

$$\frac{\partial}{\partial \mu_i} r_i(\mu_i^*, \mu_{-i}^*) = C_i'(\mu_i^*). \tag{5.4}$$

Combining the above equations yields

$$\frac{\partial}{\partial \mu_i} [R\left(\mu_\Sigma^*\right) - r_i(\mu_i^*, \mu_{-i}^*)] = 0 \tag{5.5}$$

The above equation suggests that an efficient rule of revenue distribution should ensure that $R\left(\mu_\Sigma\right) - r_i(\mu_i, \mu_{-i})$ is only determined by μ_{-i}, and is not influenced by μ_i. This implication has indeed been formally stated and proved by Ray & Goldamis [60].

Proposition 4. *$[r_i(\cdot)]$ is efficient if and only if there exists functions $E_i : \mathbb{R}^{n-1} \to \mathbb{R}$ such that for all i,*

$$r_i(\mu_i, \mu_{-i}) = R(\mu_\Sigma) - E_i(\mu_{-i}) \tag{5.6}$$

Recall that $R(\mu_\Sigma)$ can be fitted by an $(n-1)$-order polynomial, $\widehat{R}(\mu_\Sigma) = a_1\mu_\Sigma + a_2\mu_\Sigma^2 + \ldots + a_{n-1}\mu_\Sigma^{n-1}$. To better introduce the revenue distribution rule, some helpful terminology is defined:

$$P^j := \left\{ \boldsymbol{p} = [p_i] \middle| p_i \text{ is a nonnegative integer}, \sum_{i=1}^{n} p_i = j \right\} \tag{5.7}$$

$$P_i^j := \left\{ \boldsymbol{p} \in P^j \middle| p_i = 0 \right\} \tag{5.8}$$

for $j = 1, \ldots, n-1$ and $i = 1, \ldots, n$. Next, for $\boldsymbol{p} \in P^j$, let $G(\boldsymbol{p})$ be the number of nonzero coordinates of $\boldsymbol{p}$, i.e. $G(\boldsymbol{p}) := |\{l | p_l \neq 0\}|$. Note that $G(\boldsymbol{p})$ is at most j, for all $G(\boldsymbol{p})$. Finally, define $\binom{j}{\boldsymbol{p}} := \frac{j!}{p_1! \cdots p_n!}$.

The designed revenue distribution rule is given as follows. Let

$$\beta_i^j = a_j \sum_{\boldsymbol{p} \in P_i^j} \frac{n-1}{N - G(\boldsymbol{p})} \binom{j}{\boldsymbol{p}} \mu_1^{p_1} \cdots \mu_n^{p_n} \tag{5.9}$$

for $j = 1, \ldots, n-1$. Let

$$E_i(\mu_{-i}) = \sum_{j=1}^{n-1} \beta_i^j \tag{5.10}$$

Then

$$r_i(\mu_i, \mu_{-i}) = R(\mu_\Sigma) - E_i(\mu_{-i}) \tag{5.11}$$

where $r_i(\mu_i, \mu_{-i})$ is the revenue distributed to each supplier i.

Theorem 15. *The rule of revenue distribution $[r_i(\cdot)]$ as defined above is efficient and converges to budget balance in n.*

Proof. The proof that the revenue distribution rule $[r_i(\cdot)]$ as defined above is efficient is virtually

the same as that of Theorem 12 in Chapter 4. Specifically, we know $\sum E_i(\mu_{-i}) = (n-1)\hat{R}(\mu_\Sigma)$. As for the convergence of budget balance, the proof is similar to that of Proposition 3 in [60]. Specifically, for Chebyshev approximation, there exists a $B < \infty$ such that $|R(\mu_\Sigma) - \hat{R}(\mu_\Sigma)| < B\frac{\ln n}{n^2}$ for $\mu_\Sigma \in [0, \bar{\mu}_\Sigma]$. Hence,

$$\begin{aligned}\lim_{n\to\infty} \sup_{\boldsymbol{\mu}} \left| \sum_{i=1}^{n} r_i\left(\mu_i, \mu_{-i}\right) - R\left(\mu_\Sigma\right) \right| &= \lim_{n\to\infty} \sup_{\boldsymbol{\mu}} \left| (n-1)(R(\mu_\Sigma) - \hat{R}(\mu_\Sigma)) \right| \\ &= \lim_{n\to\infty} \sup_{\boldsymbol{\mu}} \frac{B(n-1)\ln n}{n^2} \\ &= 0. \end{aligned}$$

□

Note that if the revenue function $R\left(\mu_\Sigma\right)$ is indeed an $(n-1)$-order polynomial, then exact budget balance is achieved. In summary, the amount for each manufacturer participant can be calculated using the following Revenue Distribution Procedure:

Efficient Supplier Revenue Distribution Procedure (ESRD):

1. Determine coefficients $a_1, a_2, \ldots, a_{n-1}$ from an $(n-1)$-order polynomial approximation of the given $R\left(\mu_\Sigma\right)$.
2. Calculate β_i^j and $E_i(\mu_{-i})$ using (5.9) and (5.10), respectively.
3. Determine the revenue distribution for each supplier using (5.11).

5.3 Results and Discussions

Simulation studies were conducted in two example scenarios with high and low performance variations between manufacturing suppliers respectively. Each scenario has $n = 10$ suppliers. The performance μ_i for all suppliers for both scenarios are listed in Tables 5.1 and 5.3 respectively. For the high variation scenario, they are generated from the uniform distribution in $[40,\ 160]$. For the low variation scenario, they are generated from the uniform distribution in $[80,\ 120]$. Both scenarios have a revenue function $(\mu_\Sigma) = \alpha_0\sqrt{\mu_\Sigma}$, where α_0 = 1. The individual cost functions are $C_i\left(\mu_i\right) = \alpha_i\mu_i^2$, where the values of α_i, are also listed in Tables 5.1 and 5.3 respectively.

Table 5.1: Parameters for high variation scenario. Reprinted with permission from [79].

i	μ_i	α_i
1	64	7.33E-04
2	157	1.23E-04
3	136	1.64E-04
4	137	1.58E-04
5	157	1.23E-04
6	45	1.39E-03
7	81	4.53E-04
8	103	2.68E-04
9	82	4.19E-04
10	58	8.84E-04

Table 5.3: Parameters for low variation scenario. Reprinted with permission from [79].

i	μ_i	α_i
1	100	2.78E-04
2	82	4.56E-04
3	115	2.30E-04
4	117	2.26E-04
5	86	3.93E-04
6	95	3.40E-04
7	103	2.65E-04
8	113	2.25E-04
9	85	4.16E-04
10	115	2.15E-04

The efficient revenue distribution rule was compared against two practical baseline rules. One is a *proportional distribution rule (PDR)*, where each supplier i receives its revenue proportional to its performance, i.e. $r_i\left(\mu_i, \mu_{-i}\right) = \frac{\mu_i}{\mu_\Sigma} R\left(\mu_\Sigma\right)$. The other is a simple *equal distribution rule (EDR)*, where each supplier i receives an equal share of distributed revenue, i.e. $r_i\left(\mu_i,\ \mu_{-i}\right) = \frac{R(\mu_\Sigma)}{n}$.

Fig. 5.2 summarizes the simulation results for revenue, cost, and total net utility at Nash Equilibrium for the high variation scenario under the three proposed revenue distribution rules.

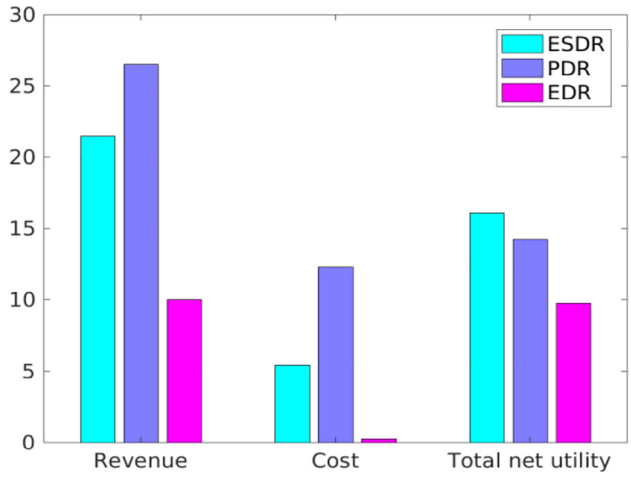


Figure 5.2: Results of the simulation. Reprinted with permission from [79].

As expected, the proposed ESDR is optimal yielding the highest net utility when compared with the PDR and EDR. PDR yields the highest revenue but at the expense of higher costs; this is a proportional revenue distribution rule that the resulting distribution is biased by revenue. On the other hand EDR yields the minimum cost but at the expense of suboptimal revenue. The results for the low performance variation data were similar, and the results are omitted for brevity.

The calculated distributed revenues for each manufacturer for high and low variation scenarios are summarized in Fig. 5.3 and 5.4. It can be observed that the proposed ESDR adjusts the appropriate revenue for each manufacturer according to their performance. Interestingly, from a manufacturer's point of view, the distributed revenue depends not only on its own performance, but also on the performance of other participants. Thus, the ESDR promotes the cooperative nature of

the cyber-enabled manufacturing environment.

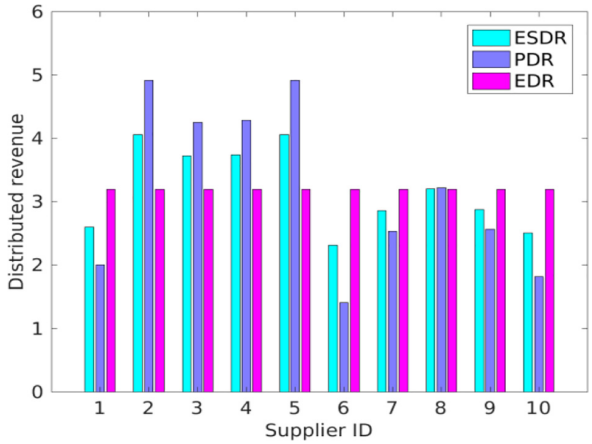


Figure 5.3: Distributed revenue for high variation scenario. Reprinted with permission from [79].

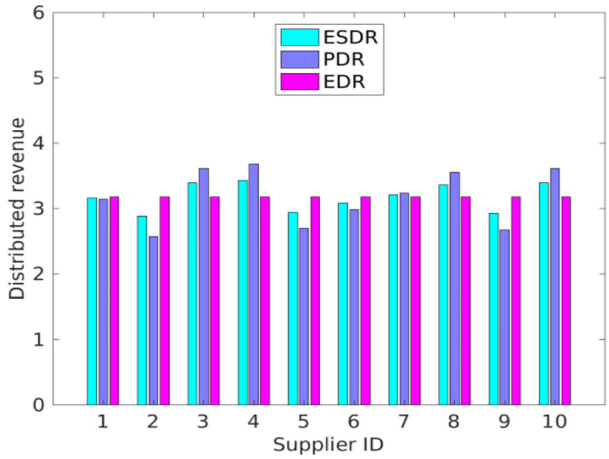


Figure 5.4: Distributed revenue for low variation scenario. Reprinted with permission from [79].

5.4 Distributed Edge Servers

We extend our work on efficient revenue distribution to consider multiple distributed edge servers. More specifically, suppose we have M servers, numbered as $j = 1, 2, \cdots, M$. On the

other hand, we have N suppliers, numbered as $i = 1, 2, \cdots, N$. Each supplier i can dynamically adjust its performance denoted by μ_i. Each supplier can connect to a nonempty subset of servers, and each server can be reached by a nonempty subset of suppliers. Each supplier i can further decide its performance distribution across the servers, μ_{ij} so that $\mu_i = \sum_j \mu_{ij}$. To simplify the notation, let $\boldsymbol{\mu}_i := (\mu_{i1}, \ldots, \mu_{iM})$. We say $\mu_{ij} \equiv 0$ if supplier i cannot reach server j.

On the other hand, each server j has a revenue of $R_j(\sum_i \mu_{ij})$ to be distributed to each supplier i. We assume the function $R_j(\cdot)$ is smooth, strictly increasing, strictly concave, and can be well fitted by a low-order polynomial function via, for example, Chebyshev least squares approximation. To simplify the notation, we let $\Psi_j := \sum_i \mu_{ij}$, $\boldsymbol{\psi}_j := (\mu_{1j}, \ldots, \mu_{Nj})$, and $\mu_{-i,j}$ denote the vector of average request rates of all suppliers other than i at server j.

Suppose each supplier i obtains some revenue and incurs some cost based on its performance $\boldsymbol{\mu}_i$. Specifically, the cost of supplier i is $C_i(\boldsymbol{\mu}_i) = C_i(\mu_{i1}, \ldots, \mu_{iM})$, where $C_i(\cdot)$ is a smooth, strictly increasing, and strictly convex function. Let $r_{ij}(\boldsymbol{\psi}_j) = r_{ij}(\mu_{1j}, \ldots, \mu_{Nj})$ be the revenue that supplier i obtains from server j. Note that $r_{ij}(\cdot)$ only depends on performance available at the server j. The utility of supplier i is $\sum_j r_{ij}(\mu_{1j}, \ldots, \mu_{Nj})$. Supplier i aims to maximize its *net utility*, $\sum_j r_{ij}(\boldsymbol{\psi}_j) - C_i(\boldsymbol{\mu}_i)$ by choosing its performance distribution μ_{ij}.

We aim to maximize the total net utility in the system, which can be written as $\sum_i \sum_j r_{ij}(\boldsymbol{\psi}_j) - C_i(\boldsymbol{\mu}_i)$. Since the total revenue to be distributed is $\sum_j R_j(\Psi_j)$, we say that the system aims to maximize $\sum_j R_j(\Psi_j) - \sum_i C_i(\boldsymbol{\mu}_i)$.

Let $\mathcal{S}_{\boldsymbol{\mu}} := \{\boldsymbol{\mu} \mid \mu_{ij} \geq 0\}$ be the feasible region of $\boldsymbol{\mu}$. The system's optimization problem is thus formally:

$$\max_{\boldsymbol{\mu} \in \mathcal{S}_{\boldsymbol{\mu}}} \sum_j R_j(\Psi_j) - \sum_i C_i(\boldsymbol{\mu}_i) \tag{5.12}$$

Since $R_i(\cdot)$ is concave, $C_j(\cdot)$ is convex, and $\mathcal{S}_{\boldsymbol{\mu}}$ is a convex set, the problem of maximizing the total net utility can be easily solved when one has complete information of all these functions. In practice, however, the function $C_i(\cdot)$ is the private information of supplier i, and a strategic supplier may not reveal its true $C_i(\cdot)$. Besides, the servers are geographically distributed areas and it incurs

additional costs to share $R_j(\cdot)$.

Now consider a game where, given $\mu_{1j}, \ldots, \mu_{Nj}$, the server j determines the revenue distributed to each supplier i, $r_{ij}(\mu_{1j}, \ldots, \mu_{Nj})$, with the constraint that $\sum_i r_{ij}(\mu_{1j}, \ldots, \mu_{Nj}) = R_j(\Psi_j)$. On the other hand, given $C_i(\cdot)$, r_{ij}, and $\mu_{i'j}$ for all $i' \neq i$ and j, each supplier i aims to maximize its own net utility by solving

$$\tilde{\boldsymbol{\mu}}_i = \underset{\boldsymbol{\mu}_i}{\operatorname{argmax}} \sum_j r_{ij}(\boldsymbol{\psi}_j) - C_i(\boldsymbol{\mu}_i) \tag{5.13}$$

We say that the system reaches a Nash Equilibrium if no supplier in the system can improve its own net utility unilaterally.

Definition 18. *A matrix $\tilde{\boldsymbol{\mu}} := [\tilde{\mu}_{ij}]$ is said to be a Nash Equilibrium if $\tilde{\boldsymbol{\mu}}_i = \operatorname{argmax}_{\boldsymbol{\mu}_i} \sum_j r_{ij}(\boldsymbol{\psi}_j) - C_i(\boldsymbol{\mu}_i), \forall i$.*

Let $\boldsymbol{\mu}^* := [\mu^*_{ij}]$ be the matrix of request rates that maximizes the total net utility of the system. The system's problem is to find the rule that distributes revenue at each server j, $[r_{ij}(\cdot)]$, to induce optimal choices of $[\mu_{ij}]$.

Definition 19. *A rule of distributing revenue, $[r_{ij}(\cdot)]$, is said to be* efficient *if $\boldsymbol{\mu}^*$ is the only Nash Equilibrium.*

Proposition 5. *$[r_{ij}(\cdot)]$ is efficient if and only if there exists functions $E_i : \mathbb{R}^{n-1} \to \mathbb{R}$ such that for all i, j,*

$$r_{ij}\left(\mu_{ij}, \mu_{-i,j}\right) = R_j(\Psi_j) - E_i(\mu_{-i,j}) \tag{5.14}$$

Proof. The proof is virtually the same as that of Proposition 1 in [60]. □

Theorem 16. *The revenue distribution rule generated by each edge server executing ESRD distributedly is efficient to the whole system.*

Proof. The proof is virtually the same as that of Theorem 12. □

Remark 4. *Distributed execution of ESRD at each server and individual optimization at each supplier leads to the global optimal solution of the whole system.*

5.5 Conclusions

With the advances in manufacturing technology and cyber infrastructure, it is conceivable that a cloud platform can be developed to connect customers and suppliers of manufacturing services in a way similar to a ride-share platform connecting riders and drivers. For such a pervasive cyber-manufacturing system to succeed, a massive number of customers and suppliers is required. This chapter presents a nonlinear revenue distribution method to promote broad participation of suppliers. Based on the game theory approach, the method optimizes the net utility of the platform and individual suppliers to achieve the Nash equilibrium of the system. The simulation results show that, with the proposed method, the revenue distribution depends not only on individual supplier's performance, but also on the performance of other participants in the system. One future enhancement of the present work is to develop fast approximations that will address scalability issues and allow online implementation with a large number of suppliers.

6. INFORMATION CENTRIC EDGE TESTBED

This chapter introduces our Information Centric Edge testbed with a programmable network stack. More specifically, we have built a testbed with programmable application, network, and MAC layers so that intelligent caching, routing, and scheduling algorithms can be implemented upon. Practical algorithms can be implemented in high level programming languages, which are easy to add, modify, and update. Both distributed and centralized algorithms are supported, considering that in some cases distributed algorithms lead to suboptimal solution. Centralized algorithms are enabled by software defined networking (SDN).

6.1 Related Work

There are quite a few existing testbeds related to Information Centric Edge. ParaDrop [5] is a testbed which envision home Wi-Fi routers as edge servers that host popular mobile Internet services. It provides application programming interface (API) for new mobile Internet services to deploy to home Wi-Fi routers, and uses virtualization for isolation between services. Besides, Furion [80] implements an edge system where a Unity-based virtual reality (VR) application acts as a game client and prefetches the background scenes of neighboring in-game locations from a nearby edge server. It has experimental multi-player support with Photon cloud providing location information of other players in real time. On the other hand, PULS [81], a testbed based on the LabVIEW software and USRP hardware, focuses on PHY and MAC layers. It can achieve low-latency (within $1\,\mathrm{ms}$) packet delivery while providing programmable MAC scheduling. In addition, there is a Docker based testbed [82] which focuses on mobility and handover support.

As mentioned before, the NDN project has an open source codebase, where NFD, the NDN forwarding daemon, has built-in caching and packet forwarding API. Currently in the NDN codebase, caching is implemented as "content store", an in-memory database, in NDN forwarding daemon (NFD). One can inherit the base class to add new caching policies. Besides, ndn-cxx, the base library, makes it easy to develop NDN applications. There are a great number of Named Data

Networking (NDN) based testbeds. However, to our knowledge, none of the NDN based testbeds support programmable MAC scheduling.

SDN has been proposed to replace traditional Ethernet switches or routers with vendor-specific control interface by a more open and vendor-neutral approach. In SDN, there is a centralized controller node that commands all switches and routers in the network via an open protocol OpenFlow. The controller also opens up a general programmable platform for network control applications or middleware, including monitoring, firewall and so on. There has been a great number of implementations of SDN controller, such as ONOS and Ryu. Besides, there are also software switches that can communicate SDN controllers via OpenFlow, including Open vSwitch[1], BOFUSS [83], etc.

6.2 Integration of LabVIEW, NDN, and VR Application

It is natural to see that by integrating LabVIEW, NDN, and the VR application, we would have a programmable network stack, including the MAC layer, network layer, and application layer. Although it appears straightforward, the integration is in fact challenging because these components run on different operating systems (OSs), written in different programmable languages, speaking different protocols, and having interfacing mismatch. Beside, the original PULS testbed was implemented on an outdated platform as of today (LabVIEW NXG 1.0 on Windows 7).

The first challenge of integrating NDN and LabVIEW is that NDN code is Unix-only while LabVIEW NXG 2 only runs on Windows 10. To make the two components run on the same host PC, we make use of the Windows Subsystem for Linux, and install the Ubuntu 16.04 app in Microsoft Store to Windows 10. NDN code, including both the base library ndn-cxx and the forwarding daemon NFD, can then be installed successfully inside the Ubuntu app.

The next challenge of integrating NDN and LabVIEW is their interfacing mismatch. We utilize local UDP sockets to connect NFD, which implements the NDN network layer, to the LabVIEW MAC layer, since UDP is the only interface available in LabVIEW to connect to other applications. LabVIEW code uses two separate UDP sockets, one for incoming traffic and the other for outgoing,

[1] `https://www.openvswitch.org/`

to write to and read from other applications respectively, Therefore, it is natural to use two UDP sockets for NFD to talk to LabVIEW, one for reception and the other for transmission. However, this conflicts with the NFD philosophy that data packets follow the reverse path of interest packets. Specifically, when LabVIEW forwards an interest packet from another host to NFD, NFD will reply with the corresponding data packet back to the same UDP socket. LabVIEW, however, does not listen on the socket and ignores the data packet. The mismatch is also depicted in Fig. 6.1.

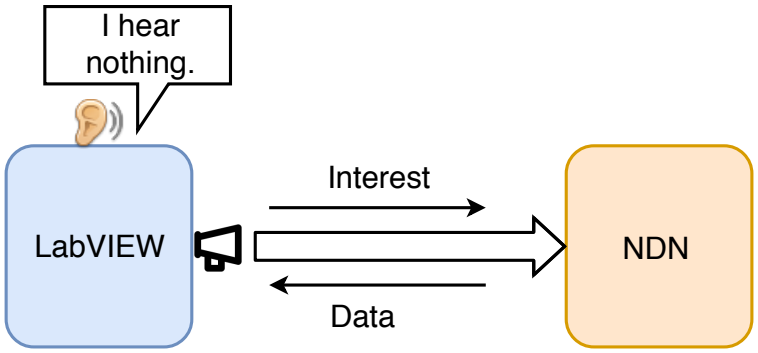


Figure 6.1: Interfacing mismatch between LabVIEW and NDN.

To address the interfacing mismatch, we hacked NFD to redirect the data packets to a separate UDP socket that LabVIEW listens. The details are as follows. First, NFD sets up a special UDP socket upon startup to the port LabVIEW listens, and we let LabVIEW forward interest packets to a special UDP port (e.g. 60001) NFD listens. Then, whenever NFD is going to send out a data packet on a particular connection, it checks whether the connection is on a regular UDP socket, and the NFD endpoint is that special UDP port (e.g. 60001). If so, NFD will move the data packet to the special UDP socket LabVIEW listens. LabVIEW will then send the data packet out via wireless.

We also enable multi-client support for our LabVIEW + NDN integration, meaning that one NDN producer node serves multiple NDN consumer nodes on top of LabVIEW. We note that this is not available in LabVIEW base implementation, nor original PULS. Our approach to enable that is to assign one set of UDP sockets per client. One set means two one-way UDP sockets in our

implementation. To facilitate that, we need a mechanism that maps from UDP port numbers to client IDs and further to MAC addresses. Fig. 6.2 illustrates our assignment of UDP ports to two client nodes. For simplicity, we let the last digit of UDP port numbers match the client ID.

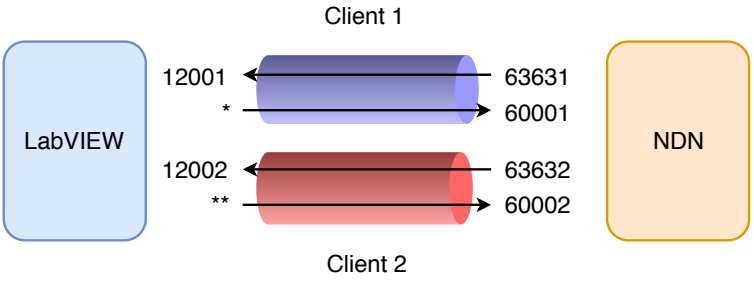


Figure 6.2: Multi-client support for LabVIEW + NDN integration. * and ** denote random port numbers used by LabVIEW to send packets to NDN.

Next, we integrate NDN with the VR application developed by Furion. In Furion, the VR app talks to an edge server via a TCP socket over commercial 802.11ac Wi-Fi. For integration, we let the VR app talk to an NDN consumer via a local TCP socket. The VR app sends out the NDN name of the video frame files it needs to prefetch to the NDN consumer, and the NDN consumer then put it in an interest packet and sends that out. For example, `/vikingvillage/3_2048.mp4,163_512.mp4` is a name for two video frame files of the Viking Village VR game, where the first video frame has an ID of 3 and a width of $2048\,\mathrm{px}$. The interest packet will be forwarded to an NDN producer on another host. The NDN producer incorporates the main logic of Furion edge server application, including preloading video frames, parsing incoming packet, and assembling data packet with requested video frames. Since video frames typically contain many bits, the resulting data packet is usually fairly large. It will be fragmented into $4\,\mathrm{KB}$ chunks by NFD before sending to LabVIEW, since the MAC layer packet has a size limit of $4061\,\mathrm{B}$.

We can demonstrate the integration of LabVIEW, NDN, and the VR app live in our testbed. The interface at the client node is shown in Fig. 6.3, and the communication of components of a client node and an access point (AP) node is shown in Fig. 6.4. The message flow initialized by

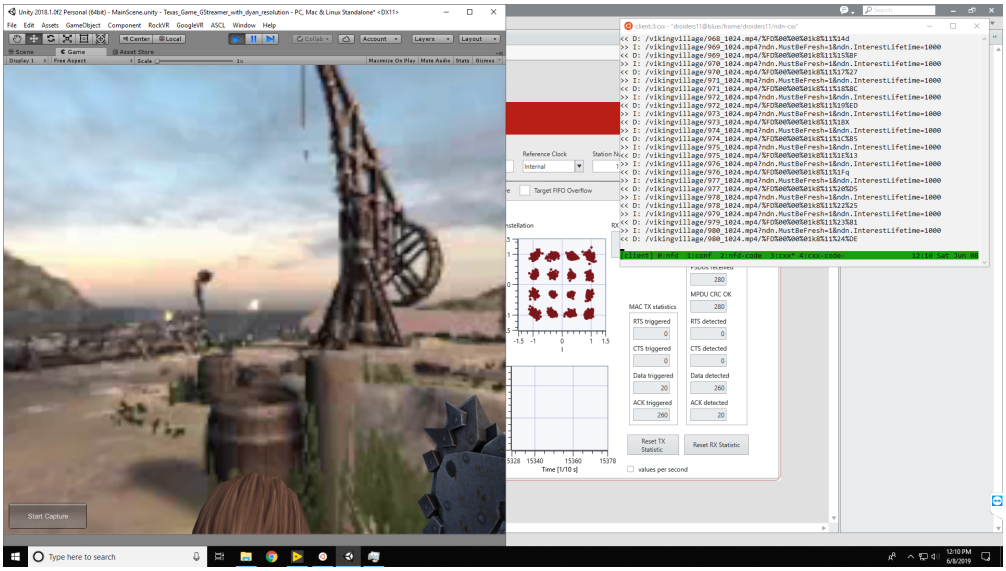


Figure 6.3: Client node interface of our testbed.

the VR app prefetching video frames of neighboring in-game locations is as follows:

1. The VR app requests new video frames files from the NDN consumer via a local TCP socket.
2. The NDN consumer listens for requests and sends out interest packets.
3. NFD forwards interest packets to LabVIEW via a local UDP socket.
4. LabVIEW sends out interest packets to the AP node via USRP wirelessly.
5. AP LabVIEW pushes interest packets to NFD.
6. NFD forwards interest packets to the NDN producer on the same host.
7. The NDN producer prepares data packets with video frames.
8. NFD fragments data packets to $4\,\mathrm{KB}$ chunks for LabVIEW.
9. LabVIEW sends data packets to the client node via USRP wirelessly.
10. Client LabVIEW pushes data packets to NFD and then the NDN consumer.
11. The NDN consumer sends data packet payload to the VR app.

12. The VR app decodes and plays the new video frames when the player moves.

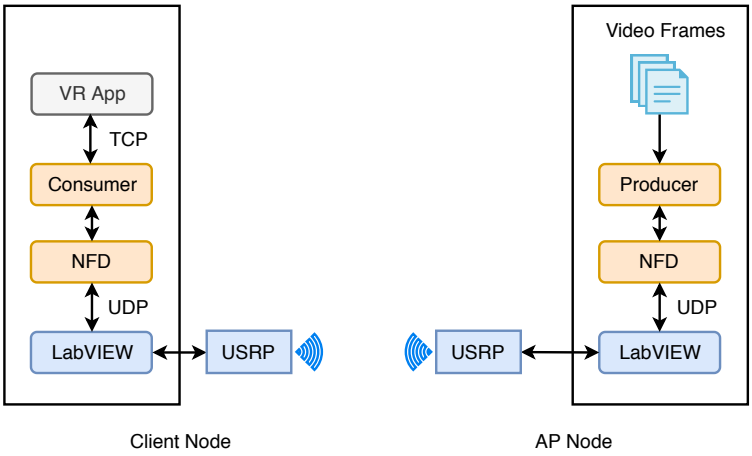


Figure 6.4: Communication of components in the integrated testbed.

6.3 Integration with Software Defined Networking

Next, we bring centralized control to the testbed via SDN. Our integration with SDN is illustrated in Fig. 6.5. In our testbed, a network administrator can use any device with a web browser for controlling the testbed. The web browser sends HTTP requests to a standard SDN controller. The SDN controller runs on its own host PC, and communicates with a software switch on an AP node via OpenFlow. The software switch acts as a proxy, and is further connected to LabVIEW and NDN on the same host. The communication between the software switch and LabVIEW/NDN is via UDP. With this design, LabVIEW or NDN does not need to know OpenFlow. Instead, the software switch forwards commands to LabVIEW or NDN based on the command type. Besides, the SDN controller can manage external standard SDN switches or routers to form a larger testbed. In our testbed, we choose Ryu as the SDN controller software because of its modular design and BOFUSS as the software switch because of its simplicity. Ryu is a Python application and thus can run on Windows directly. BOFUSS, on the other hand, is Linux only and cannot even run on the Ubuntu app on Windows. Hence, we run it in a Ubuntu virtual machine created in Oracle VM

VirtualBox[2] on Windows 10.

We extend the OpenFlow protocol[3] to support control of wireless functionalities in LabVIEW and the network layer management in NDN in our testbed. We use the Experimenter Message of the OpenFlow protocol to carry customized commands between the SDN controller and the software switch.

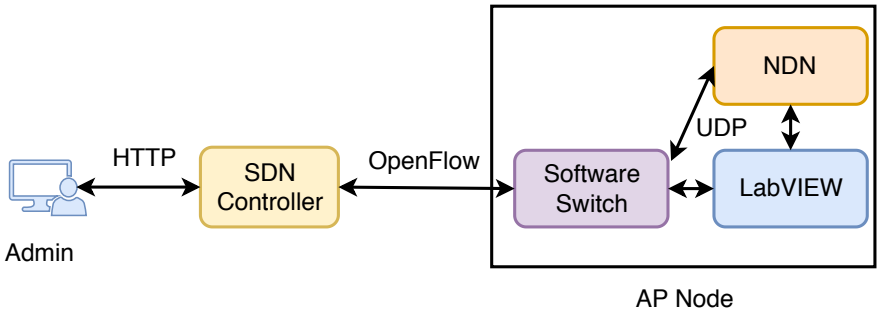


Figure 6.5: Integration with SDN.

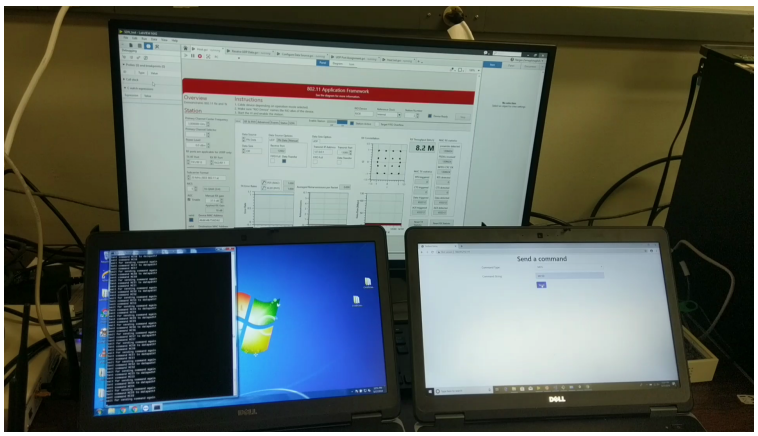


Figure 6.6: Integration of SDN and LabVIEW.

We can demonstrate the integration of SDN and LabVIEW by changing the modulation coding

[2] https://www.virtualbox.org/
[3] The OpenFlow version we are based on is 1.3.

scheme (MCS) in LabVIEW for wireless transmission on the fly from a web page. Fig. 6.6 shows the system setup, where an AP node running LabVIEW is in the upper half of the figure, to the bottom left is another host running the SDN controller, and to the bottom right is a laptop acting as the administrator device. We can set the target MCS to be e.g. 0 in the web page. After clicking the "Send" button, the MCS will be updated in the LabVIEW front panel. Since we enable RF loopback in LabVIEW for this demo, we can also observe the changes in the constellation plot and the RX throughput value. What happen behind the scene are as follows. The command in the web page is sent to the SDN controller via an HTTP REST API. The controller then sends a corresponding command to the software switch via OpenFlow. The software switch parses the command and forwards it to LabVIEW via UDP. Finally, LabVIEW parses the command and acts accordingly.

6.4 Future Work

Our Information Centric Edge testbed is work in progress.[4] In this section, we outline several directions for future work.

In terms of the integration of LabVIEW and NDN, for now we use two separate UDP sockets for each client. However, one UDP socket can be used for bidirectional communication. Hence, it is desirable to use just one socket per client for NDN communicating with LabVIEW. This simplifies the interface and is consistent with the NDN philosophy that data packets traverse the reverse path of interest packets. A potential design is as follows:

1. NDN listens on a special UDP port e.g. 60001 on startup.
2. LabVIEW creates a new UDP socket with local port e.g. 12001 and remote port 60001 on startup.
3. LabVIEW then sends a hello packet to indicate the LabVIEW link layer is available.
4. NDN, upon hearing hello, creates a special UDP socket to LabVIEW.

[4]Currently we have 14 Git repositories and over 22 wiki pages on `https://github.tamu.edu/NDN-WEN`.

5. Both NDN and LabVIEW read and write on this socket.

Another potential improvement to the integration of LabVIEW and NDN is to let NFD see the remote LabVIEW endpoint. In our current implementation, NFD treats the local LabVIEW as the remote endpoint for a connection, and is not aware of the real remote endpoint on another host. Although it is not essential for an edge testbed, it will become necessary if we would like to route packets to different upstream routers. To achieve that, a new abstract connection type for NDN, the LabVIEW face, can be introduced. The local (resp. remote) endpoint of such face is identified by the MAC address used by LabVIEW on the local (resp. remote) host. Such abstraction is seen by packet forwarding and upper layers, while local UDP sockets and wireless communication are still used for the underlying link. Particularly, we need to establish the local UDP socket when creating a new LabVIEW face.

Regarding the integration of NDN and the VR app, we are working on supporting the multi-player version of the VR game. Prior to the integration, the multi-player game relies on a cloud service called Photon to update players' locations. One key challenge of the integration is how to replace Photon with an edge service, preferably over NDN.

As for the integration with SDN we are working on enabling changing MAC scheduling policy and NDN caching policy from the SDN controller, as well as designing an SDN control application in the web page as the front end of our testbed. We would like to extend the OpenFlow protocol to handle capabilities, configuration, events, and statistics for changing various policies in our wireless testbed, while preserving the compatibility of standard wired SDN networks. To allow SDN to control NDN network management, we will create custom commands based on OpenFlow between the SDN controller and the software switch. The software switch then forwards such commands to a special NDN consumer that translates the custom commands to NDN interest packets for network management. Existing NFD management functions include changing forwarding strategies and forwarding/routing table entries, gathering events and statistics from given connections and so on. We are extending the API to support reporting available caching policies and changing caching policies from the SDN controller.

Besides, it is preferable that the testbed is easy to use from the tester's perspective. It should be made easy to configure all nodes in the testbed globally given a particular setup. It would be helpful to collect important state information and performance metric and display them in a central place for real-time monitoring and evaluation. Below is the test workflow we are working on realizing:

1. Start a test with a given setup in a control panel in a web page.
2. All test nodes configure themselves according to control commands and start running software applications.
3. All test nodes report their state information (e.g. queue length) and metrics (e.g. cost) periodically to the control panel.
4. Stop the test in the control panel.
5. All test nodes report final metrics and terminate applications.
6. Repeat the above with a different setup.

7. SUMMARY AND CONCLUSIONS

In this dissertation, we have presented our research on networking at the Information Centric Edge. We have worked on the dynamic service caching problem and proposed an online algorithm with provable order optimality and good performance against real-world traces. We have studied the joint optimization of content placement, cache selection, and video version selection, and proposed practical distributed algorithms that achieve close to optimal performance in evaluations. To handle strategic users, we have designed a non-monetary mechanism that drives distributed optimization of individual clients to converge to the global optimum via efficient cost allocation. The same method can be applied to cyber-enabled manufacturing to solve the efficient revenue distribution problem. In addition, we are building an Information Centric Edge testbed with a programmable network stack which enables virtual reality applications in real wireless environment.

REFERENCES

[1] Statista. (2019) Number of smart homes forecast worldwide from 2017 to 2023 (in millions). [Online]. Available: https://www.statista.com/statistics/887613/number-of-smart-homes-in-the-smart-home-market-worldwide/

[2] R. Ravindran, X. Liu, A. Chakraborti, X. Zhang, and G. Wang, "Towards software defined ICN based edge-cloud services," in *2013 IEEE 2nd Int. Conf. Cloud Networking (CloudNet)*. IEEE, 2013, pp. 227–235.

[3] M. Satyanarayanan, G. Lewis, E. Morris, S. Simanta, J. Boleng, and K. Ha, "The role of cloudlets in hostile environments," *IEEE Pervasive Comput.*, vol. 12, no. 4, pp. 40–49, Oct. 2013.

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st ACM Mobile Cloud Computing Workshop*, ACM. Helsinki, Finland: ACM, 2012, pp. 13–16.

[5] D. Willis, A. Dasgupta, and S. Banerjee, "ParaDrop: A multi-tenant platform to dynamically install third party services on wireless gateways," in *Proc. 9th ACM Workshop Mobility in the Evolving Internet Architecture (MobiArch '14)*, ser. MobiArch '14. ACM, 2014, pp. 43–48.

[6] T. Zhao, S. Zhou, X. Guo, and Z. Niu, "Tasks scheduling and resource allocation in heterogeneous cloud for delay-bounded mobile edge computing," in *2017 IEEE Int. Conf. Communications (ICC)*, May 2017.

[7] Q. Li, W. Shi, X. Ge, and Z. Niu, "Cooperative edge caching in software-defined hyper-cellular networks," *IEEE J. Sel. Areas Commun.*, vol. PP, no. 99, 2017.

[8] J. Tadrous, A. Eryilmaz, and H. El Gamal, "Proactive content distribution for dynamic content," in *2013 IEEE Int. Symp. Information Theory (ISIT)*. IEEE, 2013, pp. 1232–1236.

[9] J. Llorca, A. M. Tulino, K. Guan, J. Esteban, M. Varvello, N. Choi, and D. Kilper, "Dynamic in-network caching for energy efficient content delivery," in *2013 Proc. IEEE INFOCOM*. IEEE, 2013, pp. 245–249.

[10] M. M. Amble, P. Parag, S. Shakkottai, and L. Ying, "Content-aware caching and traffic management in content distribution networks," in *2011 Proc. IEEE INFOCOM*, 2011.

[11] X. Qiu, H. Li, C. Wu, Z. Li, and F. Lau, "Cost-minimizing dynamic migration of content distribution services into hybrid clouds," in *2012 Proc. IEEE INFOCOM*. IEEE, 2012, pp. 2571–2575.

[12] S. Borst, V. Gupt, and A. Walid, "Distributed caching algorithms for content distribution networks," in *2010 Proc. IEEE INFOCOM*. IEEE, 2010, pp. 1–9.

[13] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *Proc. IFIP Networking*, 2015.

[14] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, vol. 91, pp. 205–228, 2015.

[15] J. L. Hellerstein, "Google cluster data," Google research blog, Jan. 2010, posted at http://googleresearch.blogspot.com/2010/01/google-cluster-data.html.

[16] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017.

[17] A. Machen, S. Wang, K. K. Leung, B. Ko, and T. Salonidis, "Live service migration in mobile edge clouds," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 140–147, Feb. 2018.

[18] K. Ha, Y. Abe, T. Eiszler, Z. Chen, W. Hu, B. Amos, R. Upadhyaya, P. Pillai, and M. Satyanarayanan, "You can teach elephants to dance: Agile VM handoff for edge computing," in *SEC '17*, 2017.

[19] L. Ma, S. Yi, and Q. Li, "Efficient service handoff across edge servers via Docker container migration," in *SEC '17*, 2017.

[20] G. Lewis, S. Echeverría, S. Simanta, B. Bradshaw, and J. Root, "Tactical cloudlets: Moving cloud computing to the edge," in *2014 IEEE Military Communications Conf. (MILCOM)*. IEEE, 2014, pp. 1440–1446.

[21] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1002–1016, Apr. 2017.

[22] J. Liu, T. Zhao, S. Zhou, Y. Cheng, and Z. Niu, "CONCERT: A cloud-based architecture for next-generation cellular systems," *IEEE Wireless Commun.*, vol. 21, no. 6, pp. 14–22, Dec. 2014.

[23] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos, "Edge analytics in the Internet of Things," *IEEE Pervasive Computing*, no. 2, pp. 24–31, 2015.

[24] M. Satyanarayanan, Z. Chen, K. Ha, W. Hu, W. Richter, and P. Pillai, "Cloudlets: at the leading edge of mobile-cloud convergence," in *2014 6th Int. Conf. Mobile Computing, Applications and Services (MobiCASE)*. IEEE, 2014, pp. 1–9.

[25] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 2017.

[26] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[27] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *IEEE INFOCOM*, Paris, France, 2019, in press.

[28] L. Yang, J. Cao, G. Liang, and X. Han, "Cost aware service placement and load dispatching in mobile cloud systems," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1440–1452, May 2016.

[29] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966.

[30] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Commun. ACM*, vol. 28, no. 2, pp. 202–208, Feb. 1985.

[31] D. Achlioptas, M. Chrobak, and J. Noga, "Competitive analysis of randomized paging algorithms," *Theoretical Computer Science*, vol. 234, no. 1, pp. 203–218, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397598001169

[32] N. Bansal, N. Buchbinder, and J. S. Naor, "A primal-dual randomized algorithm for weighted paging," *J. ACM*, vol. 59, no. 4, pp. 19:1–19:24, Aug. 2012.

[33] T. Zhao, I.-H. Hou, S. Wang, and K. S. Chan, "RED/LED: An asymptotically optimal and scalable online algorithm for service caching at the edge," *IEEE J. Sel. Areas Commun.*, 2018.

[34] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, kc claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, Jul. 2014.

[35] S. Mastorakis, A. Afanasyev, and L. Zhang, "On the evolution of ndnSIM: an open-source simulator for NDN experimentation," *ACM Computer Communication Review*, Jul. 2017.

[36] E. Ramadan, A. Narayanan, Z.-L. Zhang, R. Li, and G. Zhang, "BIG cache abstraction for cache networks," in *2017 IEEE 37th Int. Conf. Distributed Computing Systems (ICDCS)*. IEEE, Jun. 2017.

[37] A. Sasikumar, T. Zhao, I.-H. Hou, and S. Shakkottai, "Cache-version selection and content placement for adaptive video streaming in wireless edge networks," in *2019 17th Int. Symp. Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt '19)*, Avignon, France, Jun. 2019.

[38] A. Araldo, F. Martignon, and D. Rossi, "Representation selection problem: Optimizing video delivery through caching," in *2016 IFIP Networking Conf. and Workshops*. IEEE, May 2016.

[39] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear programming: theory and algorithms*. John Wiley & Sons, 2006.

[40] M. McFly. (2018, Jan.) Test video quality 720p 1080p 1440p 2160p 4320p max bitrate which compresses YouTube. [Online]. Available: https://www.tutorialguidacomefare.com/test-video-quality-720p-1080p-1440p-2160p-max-bitrate-which-compresses-youtube/

[41] T. Ahmed, A. Mehaoua, R. Boutaba, and Y. Iraqi, "Adaptive packet video streaming over IP networks: a cross-layer approach," *IEEE J. Sel. Areas Commun.*, vol. 23, no. 2, pp. 385–401, Feb. 2005.

[42] V. K. Adhikari, S. Jain, Y. Chen, and Z.-L. Zhang, "Vivisecting YouTube: An active measurement study," in *2012 Proc. IEEE INFOCOM*, Mar. 2012.

[43] L. De Cicco and S. Mascolo, "An experimental investigation of the Akamai adaptive video streaming," in *HCI in Work and Learning, Life and Leisure*, G. Leitner, M. Hitz, and A. Holzinger, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 447–464.

[44] V. K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z.-L. Zhang, "Unreeling Netflix: Understanding and improving multi-CDN movie delivery," in *Proc. IEEE INFOCOM*. IEEE, Mar. 2012.

[45] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang, "A case for a coordinated internet video control plane," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, p. 359, sep 2012.

[46] B. Rainer, D. Posch, and H. Hellwagner, "Investigating the performance of pull-based dynamic adaptive streaming in NDN," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 8, pp. 2130–2140, Aug. 2016.

[47] T. X. Tran, P. Pandey, A. Hajisami, and D. Pompili, "Collaborative multi-bitrate video caching and processing in mobile-edge computing networks," in *2017 13th Annu. Conf. Wireless On-demand Network Systems and Services (WONS)*. IEEE, Feb. 2017.

[48] D. Bethanabhotla, G. Caire, and M. Neely, "Adaptive video streaming for wireless networks with multiple users and helpers," *IEEE Trans. Commun.*, pp. 268–285, 2014.

[49] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan, "Optimal content placement for a large-scale VoD system," *IEEE/ACM Trans. Netw.*, vol. 24, no. 4, pp. 2114–2127, aug 2016.

[50] E. Yeh, T. Ho, Y. Cui, M. Burd, R. Liu, and D. Leong, "VIP: A framework for joint dynamic forwarding and caching in named data networks," in *Proc. 1st Int. Conf. Information-Centric Networking (ICN '14)*. ACM Press, 2014.

[51] Y. Wang, W. Wang, Y. Cui, K. G. Shin, and Z. Zhang, "Distributed packet forwarding and caching based on stochastic network utility maximization," *IEEE/ACM Trans. Netw.*, vol. 26, no. 3, pp. 1264–1277, Jun. 2018.

[52] S. Ioannidis and E. Yeh, "Jointly optimal routing and caching for arbitrary network topologies," in *Proc. 4th ACM Conf. Information-Centric Networking (ICN '17)*. ACM Press, 2017.

[53] D. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 8, pp. 1439–1451, aug 2006.

[54] I.-H. Hou and P. R. Kumar, "Utility-optimal scheduling in time-varying wireless networks with delay constraints," in *Proc. 11th ACM Int. Symp. Mobile Ad Hoc Networking and Computing*. Chicago, Illinois, USA: ACM, 2010, pp. 31–40.

[55] A. Eryilmaz and I. Koprulu, "Discounted-rate utility maximization (DRUM): A framework for delay-sensitive fair resource allocation," in *2017 15th Int. Symp. Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, May 2017.

[56] R. Gupta, L. Vandenberghe, and M. Gerla, "Centralized network utility maximization over aggregate flows," in *2016 14th Int. Symp. Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, May 2016.

[57] V. Ramaswamy, D. Choudhury, and S. Shakkottai, "Which protocol? mutual interaction of heterogeneous congestion controllers," *IEEE/ACM Trans. Netw.*, vol. 22, no. 2, pp. 457–469, Apr. 2014.

[58] Cisco and/or its affiliates, "Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021," Cisco, White Paper, Mar. 2017. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html#_=_

[59] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, no. 3, pp. 237–252, Mar. 1998.

[60] K. Ray and M. Goldmanis, "Efficient cost allocation," *Management Science*, vol. 58, no. 7, pp. 1341–1356, 2012.

[61] E. Altman, T. Boulogne, R. El-Azouzi, T. Jiménez, and L. Wynter, "A survey on networking games in telecommunications," *Computers & Operations Research*, vol. 33, no. 2, pp. 286–311, 2006, special issue on Game Theory: Numerical Methods and Applications. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0305054804001248

[62] T. Alpcan and T. Başar, "A utility-based congestion control scheme for internet-style networks with delay," in *IEEE INFOCOM 2003*, vol. 3, Mar. 2003, pp. 2039–2048.

[63] T. Groves, "Incentives in teams," *Econometrica*, vol. 41, no. 4, pp. 617–631, 1973.

[64] T. Baldenius, S. Dutta, and S. Reichelstein, "Cost allocation for capital budgeting decisions," *The Accounting Review*, vol. 82, no. 4, p. 837, 2007.

[65] H. Moulin and S. Shenker, "Serial cost sharing," *Econometrica*, vol. 60, no. 5, pp. 1009–1037, 1992.

[66] M. V. Rajan, "Cost allocation in multiagent settings," *The Accounting Review*, vol. 67, no. 3, pp. 527–545, 1992.

[67] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global internet," *IEEE J. Sel. Areas Commun.*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[68] L. A. Grieco and S. Mascolo, *TCP Westwood and Easy RED to Improve Fairness in High-Speed Networks*. Springer, Berlin, Heidelberg, 2002, vol. 2334, pp. 130–146. [Online]. Available: http://dx.doi.org/10.1007/3-540-47828-0_9

[69] ——, "Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 25–38, Apr. 2004.

[70] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: Motivation, architecture, algorithms, performance," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1246–1259, Dec. 2006.

[71] T. Zhao, K. Ray, and I.-H. Hou, "A non-monetary mechanism for optimal rate control through efficient cost allocation," *IEEE/ACM Trans. Netw.*, vol. 26, no. 3, pp. 1418–1431, Jun. 2018.

[72] A. Eryilmaz and R. Srikant, "Asymptotically tight steady-state queue length bounds implied by drift conditions," *Queueing Systems*, vol. 72, no. 3, pp. 311–359, 2012.

[73] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, MA: Athena Scientific, 1999.

[74] S. K. Bose. Analysis of a M/G/1/K queue without vacations. [Online]. Available: http://home.iitk.ac.in/~skb/qbook/MG1K_Queue.PDF

[75] D. Wu, D. W. Rosen, L. Wang, and D. Schaefer, "Cloud-based design and manufacturing: A new paradigm in digital manufacturing and design innovation," *Computer-Aided Design*, vol. 59, pp. 1–14, Feb. 2015.

[76] D. Wu, M. J. Greer, D. W. Rosen, and D. Schaefer, "Cloud manufacturing: Strategic vision and state-of-the-art," *Journal of Manufacturing Systems*, vol. 32, no. 4, pp. 564–579, Oct. 2013.

[77] G. P. Cachon and M. A. Lariviere, "Supply chain coordination with revenue-sharing contracts: Strengths and limitations," *Management Science*, vol. 51, no. 1, pp. 30–44, Jan. 2005.

[78] M. Kunter, "Coordination via cost and revenue sharing in manufacturer–retailer channels," *European Journal of Operational Research*, vol. 216, no. 2, pp. 477–486, Jan. 2012.

[79] T. Zhao, I.-H. Hou, V. J. Leon, K. Ray, and J. Wang, "An efficient revenue distribution method in a cyber-enabled manufacturing system," *Manufacturing Letters*, 2018.

[80] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, N. Dai, and H.-S. Lee, "Furion: Engineering high-quality immersive virtual reality on today's mobile devices," *IEEE Trans. Mobile Comput.*, 2019.

[81] S. Yau, P.-C. Hsieh, R. Bhattacharyya, K. R. K. Bhargav, S. Shakkottai, I.-H. Hou, and P. R. Kumar, "PULS: Processor-supported ultra-low latency scheduling," in *Proc. 18th ACM Int. Symp. Mobile Ad Hoc Networking and Computing (MobiHoc '18)*. ACM Press, 2018.

[82] M. F. Tufail, "Resource management in container-based mobile edge computing," mathesis, Aalto University, Aug. 2018.

[83] E. L. Fernandes, E. Rojas, J. Alvarez-Horcajo, Z. L. Kis, D. Sanvito, N. Bonelli, C. Cascone, and C. E. Rothenberg, "The road to BOFUSS: The basic OpenFlow user-space software switch," submitted to Telecommunications Systems journal. [Online]. Available: https://arxiv.org/abs/1901.06699v1

APPENDIX A

APPENDIX FOR SERVICE CACHING

A.1 Proof of Lemma 1

This section provides a proof for Lemma 1. By assumptions of the lemma, OPT$(l) \neq$ RL(l), for all $n \leq l \leq m$. Therefore, there must exist some service that is in RL(l) but not in OPT(l). OPT needs to forward requests for these services to the back-end cloud. We will count the number of requests for these services.

Since the edge server can cache K services, we say that the edge server has K slots, numbered as L_k for $k = 1, 2, \ldots, K$. We use $L_k[r]$ to denote the r-th service cached at slot L_k during $[n, m]$. We use $d_k[r]$ to denote the time after which $L_k[r]$ is deleted and $L_k[r+1]$ is downloaded. We set $d_k[0] = n - 1$, for all k. If $L_k[r]$ is the last service cached at L_k before the m-th arrival, we set $d_k[r] = m$. To simplify the notation, let $\delta_k[r] := d_k[r-1] + 1$. Therefore, the service $L_k[r]$ is cached at L_k during $[\delta_k[r], d_k[r]]$ for all r.

Lemma 8. *Suppose* $L_k[r] = S_j$*, then* $\sum_{l=\delta_k[r]}^{d_k[r]} x_j(l) \geq \sum_{l=\delta_k[r]}^{d_k[r]} x_i(l) - 2M$.

Proof. We prove this by contradiction. By the assumption of Lemma 1, $S_i \notin$ RL(l), for all $\delta_k[r] \leq l \leq d_k[r]$. If $\sum_{l=\delta_k[r]}^{d_k[r]} x_j < \sum_{l=\delta_k[r]}^{d_k[r]} x_i - 2M$, then RED/LED would have downloaded S_i before the $d_k[r]$-th arrival. □

We classify all services that are in RL(l) but not in OPT(l), for all $l \in [n, m]$, into two types:

Definition 20. *Suppose* $L_k[r] = S_j \notin$ *OPT*(n)*, then we say that* S_j *is of* ***type I*** *if* $\sum_{l=n}^{d_k[r]} x_j(l) \geq \sum_{l=n}^{d_k[r]} x_i(l) - 4M$*, and say that* S_j *is of* ***type II*** *if there exists some* $\tau \geq n$ *such that* $\sum_{l=\tau}^{d_k[r]} x_j(l) \geq \sum_{l=\delta_k[r]}^{d_k[r]} x_i(l)$.

By this definition, OPT needs to forward at least $\sum_{l=n}^{d_k[r]} x_i(l) - 4M$ requests for $L_k[r]$ if it is of type I, and at least $\sum_{l=\delta_k[r]}^{d_k[r]} x_i(l)$ requests for $L_k[r]$ if it is of type II. It is possible for a service

to be of both type I and type II. We first prove that a service $L_k[r] \notin \text{OPT}(n)$ is of either type I or type II.

Lemma 9. *Suppose $L_k[r] = S_j \notin OPT(n)$, then S_j is of either type I or type II.*

Proof. If $r = 1$, then $\delta_k[r] = n$, and we have $\sum_{l=n}^{d_k[r]} x_j(l) \geq \sum_{l=n}^{d_k[r]} x_i(l) - 2M > \sum_{l=n}^{d_k[r]} x_i(l) - 4M$ by Lemma 8. In this case, S_j is of type I.

Next, we consider the case $r > 1$. S_j is downloaded by RED/LED after the $(d_k[r-1])$-th arrival. By the design of RED/LED, there exists a service S_{j^*} and τ such that $S_{j^*} \in \text{RL}(l)$, for all $\tau \leq l \leq d_k[r-1]$, and

$$\sum_{l=\tau}^{d_k[r-1]} x_j(l) \geq \sum_{l=\tau}^{d_k[r-1]} x_{j^*}(l) + 2M. \tag{A.1}$$

If $\tau \geq n$, then we have

$$\begin{aligned}
&\sum_{l=\tau}^{d_k[r]} x_j(l) = \sum_{l=\tau}^{d_k[r-1]} x_j(l) + \sum_{l=\delta_k[r]}^{d_k[r]} x_j(l) \\
&\geq \sum_{l=\tau}^{d_k[r-1]} x_{j^*}(l) + 2M + \sum_{l=\delta_k[r]}^{d_k[r]} x_i(l) - 2M \\
&\geq \sum_{l=\delta_k[r]}^{d_k[r]} x_i(l),
\end{aligned}$$

and S_j is of type II. On the other hand, if $\tau < n$, then we have

$$\sum_{l=\tau}^{n-1} x_j(l) < \sum_{l=\tau}^{n-1} x_{j^*}(l) + 2M, \tag{A.2}$$

or S_j would have been downloaded earlier. Combining (A.1) and (A.2) yields $\sum_{l=n}^{d_k[r-1]} x_j(l) \geq$

$\sum_{l=n}^{d_k[r-1]} x_{j^*}(l)$. Therefore,

$$\begin{aligned}\sum_{l=n}^{d_k[r]} x_j(l) &= \sum_{l=n}^{d_k[r-1]} x_j(l) + \sum_{l=\delta_k[r]}^{d_k[r]} x_j(l) \\ &\geq \sum_{l=n}^{d_k[r-1]} x_{j^*}(l) + \sum_{l=\delta_k[r]}^{d_k[r]} x_i(l) - 2M \\ &\geq \sum_{l=\delta_k[r]}^{d_k[r]} x_i(l) - 4M,\end{aligned}$$

and S_j is of type I. □

We are now ready to prove Lemma 1.

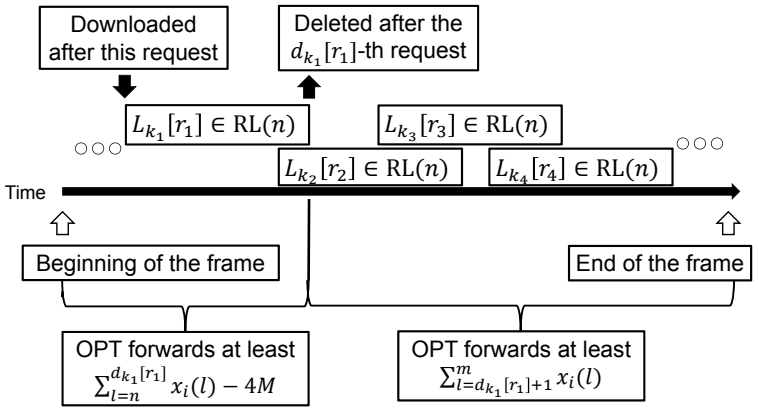


Figure A.1: An example for the proof of Lemma 1. Reprinted with permission from [33].

Proof of Lemma 1. Let $L_{k_1}[r_1]$ be the type I service with the largest $d_k[r]$. Since $\mathrm{OPT}(l) \neq \mathrm{RL}(l), \forall l \in [d_{k_1}[r_1]+1, m]$, we can find a set of type II services $\{L_{k_2}[r_2], L_{k_3}[r_3] \dots\}$ such that the union of $[d_{k_j}[r_j - 1] + 1, d_{k_j}[r_j]]$ covers $[d_{k_1}[r_1] + 1, m]$. Fig. A.1 illustrates an example of finding $L_{k_1}[r_1], L_{k_2}[r_2], \dots$. By the definition of type II service, the total number of requests that OPT needs to forward for these services is at least $\sum_{l=d_{k_1}[r_1]+1}^{m} x_i(l)$. Also, OPT needs to forward

at least $\sum_{l=n}^{d_{k_1}[r_1]} x_i(l) - 4M$ requests for $L_{k_1}[r_1]$. Therefore, in total, OPT needs to forward at least $\sum_{l=n}^{m} x_i(l) - 4M$ requests. □

A.2 Proof of Lemma 2

Proof of Lemma 2. By the first condition of RED, there are at least $2M$ requests for S_i during $[n, m]$. Otherwise, S_i cannot be downloaded during $[n, m]$, and therefore cannot be deleted at the m-th arrival.

When S_i is to be deleted at the m-th arrival, it must have the largest τ_i among all services currently cached by RED/LED, where τ_i is defined in Def. 3. Since there are at least $2M$ requests for S_i during $[n, m]$, all services in RL(m) have at least $2M$ requests during $[n, m]$.

First, consider the case RL$(m-1) \neq$ OPT$(m-1)$. There must exist a service S_j such that $S_j \in$ RL$(m-1)$, but $S_j \notin$ OPT$(m-1)$. OPT needs to forward all requests for S_j during $[n, m]$, and there are at least $2M$ of them.

Next, consider the case RL$(m-1) =$ OPT$(m-1)$. At the m-th arrival, RED/LED deletes S_i in order to download another service $S_j \notin$ RL$(m-1) =$ OPT$(m-1)$. By the design of RED/LED, there exists τ and a service $S_{i^*} \in$ RL$(m-1)$ such that the conditions in Def. 2 are satisfied. In particular, $\sum_{l=m-\tau}^{m} x_j(l) \geq \sum_{l=m-\tau}^{m} x_{i^*}(l) + 2M$. If $m - \tau \geq n$, then there are at least $2M$ requests for S_j during $[n, m]$, and OPT needs to forward all of them. On the other hand, consider the case $m - \tau < n$. Since RED/LED does not download S_j until the m-th arrival, we have $\sum_{l=m-\tau}^{n-1} x_j(l) < \sum_{l=m-\tau}^{n-1} x_{i^*}(l) + 2M$, and therefore $\sum_{l=n}^{m} x_j(l) \geq \sum_{l=n}^{m} x_{i^*}(l) \geq 2M$. OPT still needs to forward at least $2M$ requests. □

A.3 Proof of Lemma 3

Note that Lemma 8 still holds. For RED/LED with double capacity, there are at least K services that are in RL(l) but not in OPTb(l) for any $n \leq l \leq m$. These services are either of type I or type II:

Definition 21. *Suppose $L_k[r] = S_j \notin$ OPTb(n), then we say that S_j is of **type I** if $\sum_{l=n}^{d_k[r]} x_j(l) \geq \sum_{l=n}^{d_k[r]} x_i(l) - 4M$, and say that S_j is of **type II** if there exists some $\tau \geq n$ such that $\sum_{l=\tau}^{d_k[r]} x_j(l) \geq$*

$\sum_{l=\delta_k[r]}^{d_k[r]} x_i(l)$.

Lemma 10. *Suppose $L_k[r] = S_j \notin OPTb(n)$, then S_j is of either type I or type II.*

Proof. The proof is virtually the same as the proof of Lemma 9. □

We are now ready to prove Lemma 3.

Proof of Lemma 3. Recall that we can find K services that are not cached by OPTb but by RED/LED after each arrival during $[n, m]$, and they are either of type I or type II. Since the exact location of a service in the $2K$ slots does not affect the policy, we can think of the K services as packed in the first K rows out of the $2K$ slots during $[n, m]$, and assume the unchanged services between consecutive arrivals stay in their row. Because there will be at most one download after each request arrival, the sets of K services differ at most by one element between any consecutive arrivals. If a service in these K rows is deleted, we relabel the service so that it appears to be a new service afterwards. Note that the relabeling does not change the decision or the cost of RED/LED and OPTb.

For each of the K rows, we can count the number of requests that OPTb needs to forward for the services in the row in a similar fashion as in the proof of Lemma 1. Let $L_{k_1}[r_1]$ be the type I service with the largest $d_k[r]$. The rest services in this row are of type II. By the definition of type II service, the total number of requests that OPTb needs to forward for these services is at least $\sum_{l=d_{k_1}[r_1]+1}^{m} x_i(l)$. Besides, OPTb needs to forward at least $\sum_{l=n}^{d_{k_1}[r_1]} x_i(l) - 4M$ requests for $L_{k_1}[r_1]$. Therefore, in total, OPTb needs to forward at least $\sum_{l=n}^{m} x_i(l) - 4M$ requests. Note the result holds for extreme cases with only type I services or only type II services in one row.

With service relabeling, the services in each row will not appear in any other row, and thus there is no duplicate in summing up the cost of all K rows. Therefore, in total, OPTb needs to forward at least $K(\sum_{l=n}^{m} x_i(l) - 4M)$ requests. □

A.4 Proof of Lemma 4

Proof of Lemma 4. By the first condition of RED as in Def. 2, there are at least $2M$ requests for S_i during $[n, m]$. Otherwise, S_i cannot be downloaded during $[n, m]$, and therefore cannot be deleted

at the m-th arrival.

When S_i is to be deleted at the m-th arrival, it must have the largest τ_i among all services currently cached by RED/LED, where τ_i is defined in Def. 3. Since there are at least $2M$ requests for S_i during $[n, m]$, all services remaining in RL(m) have at least $2M$ requests during $[n, m]$.

Now consider RL$(m-1)$. There must exist at least K services $S_{j_1}, S_{j_2}, \ldots, S_{j_K}$ such that $S_{j_k} \in$ RL$(m-1)$, but $S_{j_k} \notin$ OPTb$(m-1)$, for all $k = 1, 2, \ldots, K$. OPTb needs to forward all requests for these K services during $[n, m]$, and there are at least $2KM$ of them. □

APPENDIX B

APPENDIX FOR EFFICIENT COST ALLOCATION

B.1 Proof of Lemma 7

Proof. The proof of Eq. (4.18) is omitted since it is virtually the same as the proof of Lemma 7 in [72].

The proof of Eq. (4.19) is stated below:

$$\begin{aligned}
|\Delta V_\perp(t)| &= |\|\boldsymbol{Q}_\perp(t+\tau)\| - \|\boldsymbol{Q}_\perp(t)\||\\
&\le \|\boldsymbol{Q}_\perp(t+\tau) - \boldsymbol{Q}_\perp(t)\|\\
&= \|\boldsymbol{Q}(t+\tau) - \boldsymbol{Q}(t) - \boldsymbol{Q}_\parallel(t+\tau) + \boldsymbol{Q}_\parallel(t)\|\\
&\le \|\boldsymbol{Q}(t+\tau) - \boldsymbol{Q}(t)\| + \|\boldsymbol{Q}_\parallel(t+\tau) - \boldsymbol{Q}_\parallel(t)\|.
\end{aligned}$$

The vector in the second term is exactly the projection of $\boldsymbol{Q}(t+\tau)-\boldsymbol{Q}(t)$ onto $\boldsymbol{g}$. Due to Pythagoras theorem, $\left\|\boldsymbol{Q}_\parallel(t+\tau) - \boldsymbol{Q}_\parallel(t)\right\| \le \|\boldsymbol{Q}(t+\tau) - \boldsymbol{Q}(t)\|$. Hence,

$$\begin{aligned}
|\Delta V_\perp(t)| &\le 2(\|\boldsymbol{Q}(t+\tau) - \boldsymbol{Q}(t)\|)\\
&= 2\sqrt{\sum_{i=1}^{N} \frac{1}{\hat{g}_i}(A_i(t) - S_i(t))^2}\\
&\le 2\sqrt{\frac{N}{\hat{g}_{\min}}},
\end{aligned}$$

where the last inequality follows because we assume that there is at most one request arrival and one request service in each time slot. □

B.2 Proof of Theorem 14

We will use a descent lemma in [73]:

Lemma 11. *Let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be continously differentiable, and let $\boldsymbol{x}$ and $\boldsymbol{y}$ be two vectors in $\mathbb{R}^n$. Suppose that*

$$\|\nabla f(\boldsymbol{x} + t\boldsymbol{y}) - \nabla f(\boldsymbol{x})\| \leq Lt\|\boldsymbol{y}\|, \forall t \in [0, 1],$$

where L is some scalar. Then

$$f(\boldsymbol{x} + \boldsymbol{y}) \leq f(\boldsymbol{x}) + \boldsymbol{y}^T \nabla f(\boldsymbol{x}) + \frac{L}{2}\|\boldsymbol{y}\|^2.$$

Proof. See Proposition A.24 of [73]. □

Proof of Theorem 14. First, note the distributed protocol is possible to stop at some iteration k, if $\boldsymbol{\lambda}(k) = \boldsymbol{\lambda}^*$. Since $\nabla f(\boldsymbol{\lambda}^*) = \mathbf{0}$, $\boldsymbol{\lambda}^*$ is stationary between successive iterations. In such case, the optimal point is reached in finite iterations. Below we will focus on the case where we have an infinite sequence $\{\boldsymbol{\lambda}(k)\}$.

Let $f(\boldsymbol{\lambda}) := \Lambda C(\Lambda) - \sum_i U_i(\lambda_i)$ be the opposite to the objective function of the server's optimization problem in (4.1). Easy to check f is smooth, strictly convex, and bounded on $\mathcal{S}_{\boldsymbol{\lambda}}$. Therefore, ∇f is Lipschitz-continous, i.e. there exists $L < \infty$ such that $\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|, \forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{S}_{\boldsymbol{\lambda}}$.

By Lemma 11, we have

$$\begin{aligned} f(\boldsymbol{\lambda}(k+1)) - f(\boldsymbol{\lambda}(k)) \leq & \nabla^T f(\boldsymbol{\lambda}(k))(\boldsymbol{\lambda}(k+1) - \boldsymbol{\lambda}(k)) \\ & + \frac{L}{2}\|\boldsymbol{\lambda}(k+1) - \boldsymbol{\lambda}(k)\|^2 \end{aligned} \tag{B.1}$$

We can rewrite the iterative update in the distributed protocol in vector form:

$$\hat{\boldsymbol{\lambda}}(k+1) = \boldsymbol{\lambda}(k) - \kappa(k)\nabla f(\boldsymbol{\lambda}(k)), \tag{B.2}$$

$$\boldsymbol{\lambda}(k+1) = \mathrm{P}^k(\hat{\boldsymbol{\lambda}}(k+1)), \tag{B.3}$$

where P^k is the projection to the convex set $\mathcal{S}_{\boldsymbol{\lambda}}^k := \{\boldsymbol{\lambda} \mid \lambda_\delta \leq \lambda_i \leq \lambda_i(k)\frac{(1-\epsilon)\mu}{\Lambda(k)}, \forall i = 1, 2, \ldots, N\}$. Easy to see $\mathcal{S}_{\boldsymbol{\lambda}}^k \subset \mathcal{S}_{\boldsymbol{\lambda}}, \boldsymbol{\lambda}(k) \in \mathcal{S}_{\boldsymbol{\lambda}}^k$, and $\boldsymbol{\lambda}(k+1) \in \mathcal{S}_{\boldsymbol{\lambda}}^k$.

By the Projection Theorem (See [73, Proposition 2.1.3]),

$$\left(\hat{\boldsymbol{\lambda}}(k+1) - \boldsymbol{\lambda}(k+1)\right)(\boldsymbol{\lambda} - \boldsymbol{\lambda}(k+1)) \leq 0, \forall \boldsymbol{\lambda} \in \mathcal{S}_{\boldsymbol{\lambda}}^k.$$

Let $\boldsymbol{\lambda} = \boldsymbol{\lambda}(k)$, and substitute in (B.2). We then have

$$(\boldsymbol{\lambda}(k) - \kappa(k)\nabla f(\boldsymbol{\lambda}(k)) - \boldsymbol{\lambda}(k+1))(\boldsymbol{\lambda}(k) - \boldsymbol{\lambda}(k+1)) \leq 0.$$

Hence,

$$\nabla^T f(\boldsymbol{\lambda}(k))(\boldsymbol{\lambda}(k+1) - \boldsymbol{\lambda}(k)) \leq -\frac{1}{\kappa(k)}\|\boldsymbol{\lambda}(k+1) - \boldsymbol{\lambda}(k)\|^2 \tag{B.4}$$

Substituting (B.4) to (B.1), we get

$$f(\boldsymbol{\lambda}(k+1)) - f(\boldsymbol{\lambda}(k)) \leq \left(\frac{L}{2} - \frac{1}{\kappa(k)}\right)\|\boldsymbol{\lambda}(k+1) - \boldsymbol{\lambda}(k)\|^2 \tag{B.5}$$

Since $\kappa(k)$ satisfies $\sum_{k=0}^{\infty}\kappa^2(k) < \infty$, there must exist some integer $K_1 > 0$ such that for all $k \geq K_1$, $\kappa(k) < \frac{2}{L}$. Therefore,

$$f(\boldsymbol{\lambda}(k+1)) \leq f(\boldsymbol{\lambda}(k)), \forall k \geq K_1.$$

By assumption, there is a bounded optimal value for the server's optimization problem at $\boldsymbol{\lambda}^*$. Hence, $\{f(\boldsymbol{\lambda}(k))\}$ is monotonically decreasing and lower bounded by $f(\boldsymbol{\lambda}^*)$. Therefore, $\{f(\boldsymbol{\lambda}(k))\}$ converges as $k \to \infty$. Taking the limit of (B.5), the left hand side goes to 0, and the right hand side is nonpositive. Therefore, $\|\boldsymbol{\lambda}(k+1) - \boldsymbol{\lambda}(k)\| \to 0$ as $k \to \infty$. Since $\{\boldsymbol{\lambda}(k)\}$ is bounded in $\mathcal{S}_{\boldsymbol{\lambda}}$, the sequence must converge to some point in $\mathcal{S}_{\boldsymbol{\lambda}}$.

Let $\bar{\boldsymbol{\lambda}} \in \mathcal{S}_{\boldsymbol{\lambda}}$ be the limit point of $\{\boldsymbol{\lambda}(k)\}$ as $k \to \infty$. We shall show $\bar{\boldsymbol{\lambda}} = \boldsymbol{\lambda}^*$ by contradiction. Suppose $\bar{\boldsymbol{\lambda}} \neq \boldsymbol{\lambda}^*$, which implies $\nabla f(\bar{\boldsymbol{\lambda}}) \neq \mathbf{0}$. Hence, $\lim_{k\to\infty}\|\nabla f(\boldsymbol{\lambda}(k))\| = \|\nabla f(\bar{\boldsymbol{\lambda}})\| > 0$.

Since the sequence $\{\boldsymbol{\lambda}(k)\}$ is infinite, $\|\nabla f(\boldsymbol{\lambda}(k))\| > 0$ for all k. Therefore, there exists $\varsigma_1 > 0$ such that $\|\nabla f(\boldsymbol{\lambda}(k))\| > \varsigma_1 > 0$ for all k.

Let $\Gamma(\Lambda) := \Lambda C(\Lambda)$. $\Gamma(\Lambda)$ is strictly convex and thus $\Gamma'(\Lambda)$ is strictly increasing. Besides, since $U_i(\cdot)$ is strictly concave, $U_i'(\cdot)$ is strictly decreasing. Consider $\boldsymbol{\lambda}(k)$ and $\Lambda(k) = \sum_i \lambda_i(k)$ for large k, the following are all the possible cases:

1. $\Lambda(k) = (1-\epsilon)\mu$. We know $\Lambda(k) > \Lambda^*$, and therefore $\Gamma'(\Lambda(k)) \geq \Gamma'(\Lambda^*)$. There must be some client i such that $\lambda_i(k) > \lambda_i^*$, and thus $U_i'(\lambda_i(k)) < U_i'(\lambda_i^*)$. Hence, $U_i'(\lambda_i(k)) - \Gamma'(\Lambda(k)) < U_i'(\lambda_i^*) - \Gamma'(\Lambda^*) = 0$, and the update substep will have $\hat{\lambda}_i(k+1) < \lambda_i(k)$. Since $\lambda_i(k) > \lambda_i^* > \lambda_\delta$, the distributed projection allows λ_i to decrease. Therefore, after one iteration we have $\lambda_i(k+1) < \lambda_i(k) = \lambda_i(k)\frac{(1-\epsilon)\mu}{\Lambda(k)}$. For all $j \neq i$, $\lambda_j(k+1) \leq \lambda_j(k)\frac{(1-\epsilon)\mu}{\Lambda(k)}$. Therefore, $\Lambda(k+1) < (1-\epsilon)\mu$.

2. $\Lambda(k) < (1-\epsilon)\mu$, and there is some i such that $\lambda_i(k) = \lambda_\delta$. Recall that under efficient delay allocation, $\frac{\partial}{\partial \lambda_i}\lambda_i D_i(\lambda_i, \lambda_{-i}) = \frac{\partial}{\partial \lambda_i}\Lambda C(\Lambda) = \Gamma'(\Lambda)$. We have

$$\begin{aligned} -\frac{\partial}{\partial \lambda_i} f(\boldsymbol{\lambda}(k)) &= U_i'(\lambda_\delta) - \Gamma'(\Lambda(k)) \\ &= U_i'(\lambda_\delta) - \frac{\partial \lambda_i D_i}{\partial \lambda_i}(\lambda_\delta, \lambda_{-i}(k)) > 0, \end{aligned}$$

   where the last inequality is due to the assumption that the Nash Equilibrium is in the interior of the feasible set $\mathcal{S}_{\boldsymbol{\lambda}}$. The update substep will then have $\hat{\lambda}_i(k+1) > \lambda_i(k)$. Note that $\sum_k \kappa^2(k) < \infty$ implies $\lim_{k\to\infty} \kappa(k) = 0$. Besides, $\frac{\partial}{\partial \lambda_i} f(\boldsymbol{\lambda}(k))$ is bounded. Since $\lambda_i(k) < \lambda_i(k)\frac{(1-\epsilon)\mu}{\Lambda(k)}$, for sufficiently large k, $\lambda_\delta < \hat{\lambda}_i(k+1) < \lambda_i(k)\frac{(1-\epsilon)\mu}{\Lambda(k)}$. After one iteration, we have $\lambda_\delta < \lambda_i(k+1) < \lambda_i(k)\frac{(1-\epsilon)\mu}{\Lambda(k)}$. Hence, $\Lambda(k+1) < (1-\epsilon)\mu$ and $\lambda_i(k+1) > \lambda_\delta, \forall i$.

3. $\Lambda(k) < (1-\epsilon)\mu$ and $\lambda_i(k) > \lambda_\delta, \forall i$. In this case, $\boldsymbol{\lambda}(k)$ lies in the interior of $\mathcal{S}_{\boldsymbol{\lambda}}^k$. Note that $\lim_{k\to\infty} \kappa(k) = 0$, and $\|\nabla f(\boldsymbol{\lambda}(k))\|$ is bounded. Therefore, for sufficiently large k, $\hat{\boldsymbol{\lambda}}(k+1)$ also lies in the interior of $\mathcal{S}_{\boldsymbol{\lambda}}^k$. In this case, $\boldsymbol{\lambda}(k+1) = P^k(\hat{\boldsymbol{\lambda}}(k+1)) = \hat{\boldsymbol{\lambda}}(k+1)$. Hence, $\Lambda(k+1) < (1-\epsilon)\mu$ and $\lambda_i(k+1) > \lambda_\delta, \forall i$.

Therefore, we can conclude that there exists an integer $K_2 > 0$, such that for all $k \geq K_2$, $\Lambda(k) < (1-\epsilon)\mu$, and $\lambda_i(k) > \lambda_\delta, \forall i$. $\boldsymbol{\lambda}(k+1) = \hat{\boldsymbol{\lambda}}(k+1) = \boldsymbol{\lambda}(k) - \kappa(k)\nabla f(\boldsymbol{\lambda}(k))$. Using Lemma 11 again, we have

$$\begin{aligned} f(\boldsymbol{\lambda}(k+1)) - f(\boldsymbol{\lambda}(k)) \leq & -\kappa(k)\|\nabla f(\boldsymbol{\lambda}(k))\|^2 \\ & + \frac{L}{2}\kappa^2(k)\|\nabla f(\boldsymbol{\lambda}(k))\|^2 \\ = & -\kappa(k)\left(1 - \frac{L}{2}\kappa(k)\right)\|\nabla f(\boldsymbol{\lambda}(k))\|^2 \end{aligned} \tag{B.6}$$

Let $K_3 := \max\{K_1, K_2\}$. For all $k \geq K_3$, $\kappa(k) < \frac{2}{L}$, and there exists some $\varsigma_2 > 0$ such that $1 - \frac{L}{2}\kappa(k) > \varsigma_2$. Recall $\|\nabla f(\boldsymbol{\lambda}(k))\| > \varsigma_1 > 0$. Substituting into (B.6), we have

$$f(\boldsymbol{\lambda}(k+1)) - f(\boldsymbol{\lambda}(k)) < -\varsigma_1^2\varsigma_2\kappa(k), \forall k \geq K_3. \tag{B.7}$$

Let $\varsigma := \varsigma_1^2\varsigma_2$. Taking the telescopic sum of (B.7) from K_3 to some $\bar{k} > K_3$, we get

$$f(\boldsymbol{\lambda}(\bar{k})) - f(\boldsymbol{\lambda}(K_3)) < -\varsigma \sum_{k=K_3}^{\bar{k}} \kappa(k).$$

Let $\bar{k} \to \infty$. We have

$$f(\bar{\boldsymbol{\lambda}}) - f(\boldsymbol{\lambda}(K_3)) < -\varsigma \sum_{k=K_3}^{\infty} \kappa(k).$$

The left hand side is bounded, while the right hand side is $-\infty$ since $\sum_k \kappa(k) = \infty$. This results in a contradiction. Hence, it is impossible that $\bar{\boldsymbol{\lambda}} \neq \boldsymbol{\lambda}^*$. In other words, $\boldsymbol{\lambda}(k) \to \boldsymbol{\lambda}^*$ as $k \to \infty$. $\square$