Wide-Area Traffic Management for Cloud Services

Joe Wenjie Jiang

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
By the Department of
Computer Science
Adviser: Jennifer Rexford & Mung Chiang

April 2012

Abstract

Cloud service providers (CSPs) need effective ways to distribute content across wide area networks. Providing large-scale, geographically-replicated online services presents new opportunities for coordination between server selection (to match subscribers with servers), traffic engineering (to select efficient paths for the traffic), and content placement (to store content on specific servers). Traditional designs isolate these problems, which degrades performance, scalability, reliability and responsiveness. We leverage the theory of distributed optimization, cooperative game theory and approximation algorithms to provide solutions that jointly optimize these design decisions that are usually controlled by different institutions of a CSP.

This dissertation proposes a set of wide-area traffic management solutions, which consists of the following three thrusts:

(i) *Sharing information*: We develop three cooperation models with an increasing amount of information exchange between the ISP's (Internet Service Provider) traffic engineering and the CDN's (Content Distribution Network) server selection. We show that straightforward ways of sharing information can be quite sub-optimal, and propose a Nash bargaining solution to reduce the efficiency loss. This work sheds light on ways that different groups of a CSP can communicate to improve their performance.

(ii) *Joint control*: We propose a content distribution architecture by federating geographically or administratively separate groups of last-mile CDN servers (*e.g.*, nano data centers) located near end users. We design a set of mechanisms to solve a joint content placement and request routing problem under this architecture, achieving both scalability and cost optimality. This work demonstrates how to jointly control multiple traffic management decisions that may have different resolutions (*e.g.*, inter vs. intra ISP), and may happen at different timescales (*e.g.*, minutes vs. several times a day).

(iii) *Distributed implementation*: Today's cloud services are offered to a large number of geographically distributed clients, leading to the need for a decentralized traffic control. We present DONAR, a distributed mapping service that outsources replica selection, while providing a sufficiently expressive service interface for specifying mapping policies based on performance, load, and cost. Our solution runs on a set of distributed mapping nodes for directing local client requests, which only requires a lightweight exchange of summary statistics for coordination between mapping nodes. This work exemplifies a decentralized design that is simultaneously scalable, reliable, and accurate.

Collectively, these solutions are combined to provide a synergistic traffic management system for CSPs who wish to offer better performance to their clients at a lower cost. The main contribution of this dissertation is to develop new design techniques to make this process more systematic, automated and effective.

Acknowledgments

I owe tremendous thanks to a great many people that I am fortunate to meet, who help build my research and share their wisdom of life.

I am deeply grateful to my advisors, Jennifer Rexford and Mung Chiang, for influencing me by their pursuit of top quality research. I am fortunate to have both of them advise me — an unparalleled opportunity to learn applying theory in solving practical problems. Thanks to Jen for her selfless offerings throughout my entire graduate study: the freedom needed to pursue exciting ideas, the knowledge needed to solve complex problems, the optimism needed to challenge the unknowns, and the skills needed for perfection on all aspects of research: writing, presentation, communication and teamwork. Thanks to Mung for his dedication that helps shape this thesis, his insightful thoughts on both research and life, his guidance on being a professional researcher, and his passion that always encourages me to explore new areas fearlessly. I could not ask for more from them.

I would like to thank Mike Freedman, Andrea LaPaugh, and Augustin Chaintreau for serving on my thesis committee. Mike is also the co-author and mentor of the DONAR project presented in Chapter 4. His taste of research and sharp comments have always been a reliable source for improving my work. Thanks to Andrea for enlightening my presentation. Thanks to Augustin for being hands-on, which has made this thesis more solid and complete.

I would also like to thank the contributors of this thesis. Rui Zhang-Shen co-authored the work in Chapter 2. I am also grateful for her mentorship during the early stage of my Ph.D. Thanks to Stratis Ioannidis, Laurent Massoulie and Fabio Picconi for helping with the proofs and measurement data in Chapter 3. Thanks to Laurent and Christophe Diot for giving me the opportunity to intern at Technicolor Research Lab in Paris. Thanks to Stratis for demonstrating the quality of a true theorist, and other researchers for making the internship an unforgettable experience. Thanks to Patrick Wendell, the gifted young undergrad, for all the system implementation work in Chapter 4. I enjoyed our brainstorming, deadline fights, and lunch sandwiches.

Special thanks to my other collaborators, including S.-H. Gary Chan, Minghua Chen, Sangtae Ha, Tian Lan, Shao Liu, Srinivas Narayana, D. Tony Ren, Bin Wei, and Shaoquan Zhang, for their inputs to the work that I am not able to present in this thesis. Thanks to Gary for hosting my summer visit in HKUST, and providing generous resource and support to conduct my research.

Thanks to Mike Schlansker, Yoshio Turner, and Jean Tourrilhes for mentoring my internship at HP Labs, making it a productive and pleasant winter in Palo Alto.

I would like to thank a long list of members (past and present) from Cabernet group: Ioannis Avramopoulos, Matthew Caesar, Alex Fabrikant, Sharon Goldberg, Robert Harrison, Elliott Karpilovsky, Eric Keller, Changhoon Kim, Haakon Ringberg, Michael Schapira, Srinivas Narayana, Martin Suchara, Peng Sun, Yi Wang, Minlan Yu, Rui Zhang-Shen, and Yaping Zhu; and from EDGE Lab: Ehsan Aryafar, Jiasi Chen, Amitabha Ghosh, Sangtae Ha, Prashanth Hande, Jiayue He, Jianwei Huang, Hazer Inaltekin, Ioannis Kamitsos, Hongseok Kim, Haris Kremo, Tian Lan, Ying Li, Jiaping Liu, Shao Liu, Chris Leberknight, Soumya Sen, Chee Wei Tan, Felix Wong, Dahai Xu, and Yung Yi. Working in a mixed culture of computer science and electrical engineering, and with folks of versatile expertise, has given me a unique opportunity to broaden my scope of knowledge and ways of thinking. It was great to have worked with and learned from them. Special thanks to Melissa Lawson for her years' dedication in serving as the graduate coordinator.

Friends have immensely enriched my life at graduate school. I thank: Dalin Shi, for being like a brother; Yunzhou Wei, for beers and basketball, Yiyue Wu, for being the best roommate; Wei Yuan, for being a good old-school friend; Yinyin Yuan, for comparing Princeton and Cambridge, and introducing the art of wine; Pei Zhang, for offering fun and selfless help; and many others from computer science department, for their support and making me feel home.

I would like to thank John C.S. Lui for sharing his wisdom, and continually encouraging me to follow the heart.

I'd like to acknowledge Princeton University, National Science Foundation, Air Force Office of Scientific Research, and Technicolor Labs for their financial support.

Thanks to Yuanyuan for bringing me happiness and being part of my life.

Thanks to my parents for their enduring love and support. This dissertation is dedicated to you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet is increasingly a platform for online services—such as Web search, social networks, multiplayer games and video streaming— distributed across multiple locations for better reliability and performance. In recent years, large investments have been made in massive data centers supporting computing services, by Cloud Service Providers (CSPs) such as Facebook, Google, Microsoft, and Yahoo!. The significant investment in capital outlay by these companies represents an ongoing trend of moving applications, *e.g.*, for desktops or resource-constrained devices like smartphones, into the cloud. The trend toward geographically-replicated online services will only continue and increasingly include small enterprises, with the success of cloud-computing platforms like Amazon Web Services (AWS) [1]. CSPs usually host a wide range of applications and online services, and lease the computational power, storage and bandwidth to their customers, for instance, the video subscription service Netflix.

Each online service has specific performance requirements. For example, Web search and multiplayer games need low end-to-end latency; video streaming needs high throughput, whereas social networks require a scalable way to store and cache user data. Clients access these services from a wide variety of geographic locations over access networks with wildly different performance. CSPs undoubtedly care about the end-to-end performance their customers experience. For instance, even small increases in round-trip times have significant impact on their revenue [2]. On the other hand, CSPs must consider operational costs, such as the price they pay their upstream network providers for bandwidth, and the electricity costs in their data centers. Large CSPs can easily send and receive petabytes of traffic a day [3], and spend tens of millions of dollars per year on electricity [4]. As such, CSPs increasingly need effective ways to manage their traffic in order to optimize client performance and operational costs.

1.1 An Overview of Today's Cloud Service Providers

Traditionally, traffic management over wide-area networks has been performed *independently* by administratively separated entities, which together make up today's cloud ecosystem. As illustrated in Figure 1.1, there are four parties in the ecosystem:

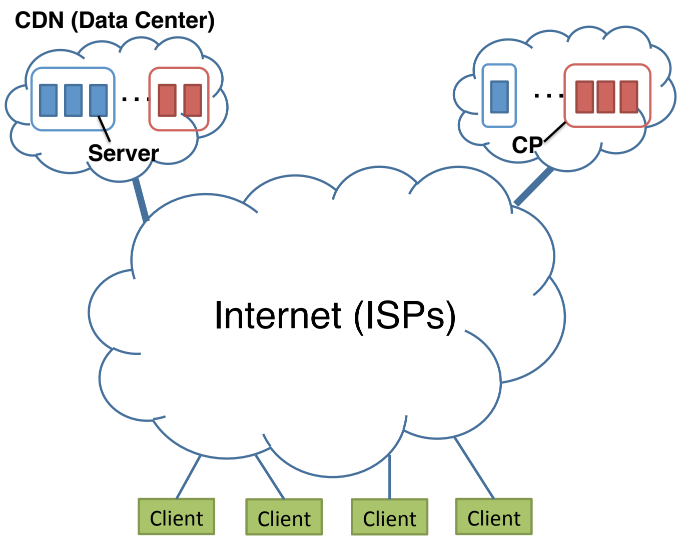


Figure 1.1: Four parties in the cloud ecosystem: Internet Service Provider (ISP), Content Distribution Network (CDN), content provider, and client.

- **Internet Service Provider:** Internet Service Providers (ISPs, *a.k.a.* network providers such as AT&T [5]) provide connectivity, or the bandwidth "pipes" to transport content—simply treating them as packets—and thus are oblivious of content sources. A traditional ISP's primary role is to deploy infrastructure, manage connectivity, and balance traffic load inside its network by computing the *network routing* decisions. In particular, an ISP solves the traffic engineering (TE) problem, *i.e.*, adjusting the routing configuration to the prevailing traffic. The goal of TE is to ensure efficient routing to minimize congestion, so that users experience low packet loss, high throughput, and low latency, and that the network can gracefully absorb bursty traffic.

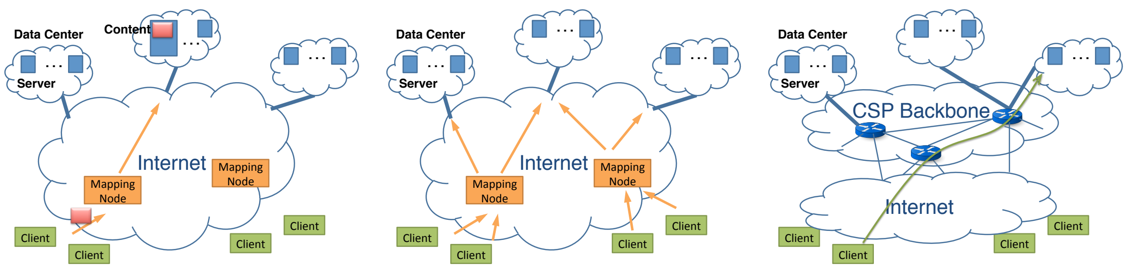
- **Content Distribution Network:** Content Distribution Networks (CDNs, *e.g.*, Akamai [6]) provide the infrastructure to replicate content across geographically-diverse data centers (or servers). Today's CDNs strategically place servers across geographically distributed locations, and replicate content over a number of designated servers. CDNs solve a *server selection* (SS) problem, *i.e.*, determining which servers should deliver content to each end user. The goal of server selection is to meet user demand, minimize network latency to reduce user waiting time, and balance server load to increase throughput. Sometimes, the job of server selection can be outsourced to a third party, in addition to the CDN, to allow customers to specify their own high-level policies, based on performance, server and network load, and cost.

- **Content Provider:** Content Providers (CPs, *e.g.*, Netflix [7]), *a.k.a.* tenants of the cloud service, produce, organize and deliver content to their clients by renting a set of CDN servers for better availability, performance and reliability. Content providers generate revenue by delivering content to clients. Increasingly, cloud computing offers an attractive approach where the cloud provider offers elastic server and network resources, while allowing customers to design and implement their own services. While network routing and server selection are handled by individual CDNs and ISPs, such customers are left largely to handle *content placement* on their own, *i.e.*, placing and caching content on appropriate servers to meet client demands and reduce end-to-end latency.

- **Client:** Clients, who consume the content, choose their ISPs for Internet connectivity, and subscribe to various content providers for their services.

Therefore, the user-perceived performance is affected by many factors that are influenced by different ISPs, CDNs and CPs. As the Internet increasingly becomes a platform for online services, the boundaries between these parties have been more blurred than ever. We formally define a CSP as a service provider that plays two or multiple roles in the cloud ecosystem, for instance:

- ISP + CDN, *e.g.*, AT&T, who deploys CDNs inside its own network.



(a) Content placement (b) Server selection (c) Network routing

Figure 1.2: CSP traffic management decisions

- CDN + CP, *e.g.*, Youtube, who deploys many geographically-distributed servers to stream videos.
- ISP + CDN + CP, *e.g.*, Google, who has its own data centers and the network backbone, provides online services to its customers.

In all of the above scenarios, CSPs have a unique opportunity to coordinate between traffic-management tasks that are previously controlled by different institutions. In particular, today's CSPs optimize client performance and operational costs by controlling (i) *content placement*, *i.e.*, placing and caching content in servers (or data centers) that are close to clients, (ii) *server selection*, *i.e.*, directing clients across the wide area to an appropriate service location (or "replica"), and (iii) *network routing*, *i.e.*, selecting wide-area paths to clients, or intra-domain paths within a CSP's own backbone, as shown in Figure 1.2. To handle wide-area server-selection, CSPs usually run DNS servers or HTTP proxies (front-end proxies) at multiple locations and have these nodes coordinate to distribute client requests across the data centers. To have control over wide-area path performance, large CSPs often build their own backbone network to inter-connect their data centers, or connect each data center to multiple upstream ISPs when they do not have a backbone. Further, some CSPs deploy a large number of "nano" data centers [8] and need to place specific content in each server, or may additionally cache popular content in the front-end proxies when they direct client requests.

In the rest of this chapter, we outline the detailed design requirements (§1.2), introduce our design methodologies (§1.3), and summarize our main contributions (§1.4).

1.2 Requirements of CSP Traffic Management

Today's CSPs usually do not make optimal traffic management decisions and achieve the high performance and low cost that they could, due to (i) *limited visibility*, (ii) *independent control*, and (iii) *poor scalability.* To address these challenges, we propose a complete set of networking solutions that promote *information sharing*, *joint control*, and *distributed implementation.*

1.2.1 The Need for Sharing Information

Traffic management decisions are made by administratively separated groups of the CSP, or even affected by other institutions such as intermediate ISPs that transit traffic between clients and the CSP. They often have relatively *poor visibility* into each other, leading to sub-optimal decisions.

Misaligned objectives lead to conflicting decisions. Conventionally, ISPs and CDNs (or content providers) optimize different objectives. Typically, ISPs adjust the routing configurations for *all* traffic inside their networks, in addition to the CDN traffic, in the hope of minimizing network congestion, achieving high throughput and low latency, and reducing operational costs such as traffic transit costs. On the other hand, CDNs direct clients to the closest data center to reduce round-trip times, without regard to the resulting costs and network congestion. As a consequence, the network routing and server selection decisions are usually at odds. For example, an ISP may prefer to route the CDN traffic on a longer path but with lower congestion, and CDN may direct a client to a closer data center that is reached through an expensive provider path.

Incomplete visibility leads to sub-optimal decisions. In making server-selection decisions, a CDN needs to predict the client round trip latency to a server, which depends on the wide-area path performance. In practice, the CDN has limited visibility into the underlying network topology and routing, and therefore has limited ability to predict client performance in a timely and accurate manner. A CDN rates server performance by IP geolocation database [9, 10], or Internet path performance prediction tools [11], which are usually load oblivious. Therefore, without information about link loads and capacities, a CDN may direct excessive traffic to a geographically closer server, leading to overloaded links.

1.2.2 The Need for a Joint Design

Traffic management decisions are made *independently* by different institutions or different groups in the same company, yet they clearly influence each other. As such, today's practice often achieves much lower performance or higher costs than a coordinated solution.

Separate optimization does not achieve globally optimal performance. We observe that separate decision makings do not enable a globally optimal performance, even given the complete visibility into all participating systems. For example, separating server selection and traffic engineering, *e.g.*, carefully optimizing one decision on top of the other, leads to sub-optimal equilibria, even when the CDN is given accurate and timely information about the network. In general, such separate optimizations do not enable a mutually-optimal performance, motivating a joint design for a coordinated solution.

Decision making happens at different timescales. Since traffic management decisions are made by different institutions, they usually are optimized at different timescales. For instance, the ISP runs traffic engineering at the timescale of hours, although it could run on a much smaller timescale. Server-selection is usually optimized at a smaller timescale such as minutes to achieve an accurate load-balancing across servers. Depending on content providers' choices, how often the content placement decision is updated can vary quite differently, ranging from a few times a day to on-the-fly caching. These heterogeneities raise the need for a joint control that is both accurate and practical.

1.2.3 The Need for a Decentralized Solution

Traffic management decisions are often made in a *centralized* manner, leading to a high complexity and poor scalability. The functional separation implied by today's architecture, and the large number of network elements (*e.g.*, data centers, servers, wide-area network paths, and clients), raise the need for a decentralized solution in our design.

Functional separation is practical and efficient. A joint design naturally leads to a central coordinator for controlling all traffic management decisions inside a CSP. However, we want a modularized design by functionally separating these decisions, *e.g.*, between the ISP and the CDN, yet achieving a jointly optimal solution. Today's server selection, network routing, and content placement are themselves performed by large distributed systems managed by separate groups in the same company, or even outsourced to third parties (*e.g.*, Akamai running Bing's front-end servers, or DONAR [10]). Tightly coupling these systems would lead to a complex design that is difficult to administer. Instead, these systems should continue to operate separately and run on existing infrastructures, with lightweight coordination to arrive at good collective decisions.

Distributed implementation is scalable and reliable. The need for scalability and reliability should drive the design of our system, leading to a distributed solution that consists of a set of spatially-distributed network nodes, such as mapping nodes (for server selection), backbone or edge routers (for networking routing), and nano data centers and proxy servers (for content placement), to handle traffic management in an autonomous and collaborative manner. While a simple approach by having a central coordinator is straightforward, it introduces a single point of failure, as well as an attractive target for attackers trying to bring down the service. Further, it incurs significant overhead for the infrastructure nodes to interact with the controller, leading to excessive communication overhead. Finally, a centralized solution adds additional delay, making the system less responsive to sudden changes in client demands (*i.e.*, flash crowds). To overcome these limitations, we need a scalable and reliable distributed solution that runs on individual nodes while still attains a globally optimal performance.

Meeting all the above requirements poses several significant challenges. First, each online service runs at multiple data centers at different locations, which vary in their capacity (*e.g.*, number of servers), their connectivity to the Internet (*e.g.*, upstream ISPs for multihoming, or bandwidth provisioning for individual nano data centers), and the proximity to their clients. These heterogeneities present many practical constraints in our design. Second, some problems, *e.g.*, enabling the joint control as an optimization problem, do not accept a simple formulation that is computationally tractable. We need advanced optimization and approximation techniques to make the solution computationally efficient and easy to implement, yet with provable guarantee of optimality. Further, as client demands and network conditions are varying from time to time, our solution should be well adaptive to these changes. We address these challenges in this dissertation.

1.3 Design Approaches

To address the wide-area networking needs of CSPs, our research follows methodologies that are tailored to our specific design goals, as summarized in the following three thrusts.

1.3.1 A Top-Down Design

Today's CSPs deploy online services across multiple ISP networks and data centers. Traffic management decisions, such as server selection, can have different control granularities. For example, a CDN deploys its servers inside many ISP networks. When directing client requests, a CDN's decision consists of two levels: inter-domain (*e.g.*, which ISP to choose) and intra-domain (*e.g.*, which server to choose within an ISP network). Further, traffic management decisions are made at different timescales. Content placement and network routing are updated relatively less frequently, while server selection are re-optimized at a smaller timescale in order to adapt to changing traffic and network conditions. To address these issues, our design follows a top-down principle: make decisions from large (*e.g.*, ISP- and data center-wide) to small (*e.g.*, server-wide) resolutions, and from large (*e.g.*, hourly) to small (*e.g.*, of several minutes) timescales. Such a design choice has several merits. First, it simplifies the problem and allows us to divide-and-conquer the problem of a prohibitively large size. Second, the top-down design is scalable as the number of data centers and clients grows. Third, it reduces the implementation complexity as it allows a separation of control at different institutions.

1.3.2 Optimization as a Design Language

Convex optimization [12] is a special class of mathematical optimization problems that can be solved numerically very efficiently. We utilize convex optimization to formulate traffic management problems faced by today's CSPs. We carefully define a set of performance metrics, *e.g.*, latency and throughput, as the objectives in the optimization problem. We are also able to capture various operational costs, *e.g.*, network congestion, bandwidth and electricity costs, in the objective function. Optimization also allows us to express real-world constraints, such as link capacity, and to freely customize CSP policies, for instance, each data center specifies a traffic split weight or bandwidth cap. The solid foundation in convex programming allows us to solve these problems in a computationally efficient way, and with the optimality guarantee that many heuristics used in practice today could not achieve. We are faced with the challenge, however, that many practical problems—such as joint control of traffic engineering and server selection—do not have a straightforward convex formulation, and hence we cannot directly apply efficient solution techniques. To overcome this limitation, we develop methods to convert or approximate the non-convex problem into a convex form, with a provable bound on the optimality gap.

1.3.3 Optimization Decomposition for a Distributed Solution

To enable decentralized traffic management rather than a centralized approach, we leverage the optimization decomposition theory to derive distributed solutions that are provably optimal. Distributed algorithms are notoriously prone to oscillations (*e.g.*, distributed mapping nodes overreact based on their own local information) and inaccuracy (*e.g.*, the system does not optimize the designated objectives). Our design must avoid falling into these traps. We utilize primal-based and dual-based decomposition methods to decouple local decision variables, *e.g.*, server selection at each mapping node, and content placement at each server. We analytically prove that our decentralized algorithms converge to the optimal solutions of the optimization problems, and validate their effectiveness in practice with real-life traffic traces.

1.4 Contributions

This dissertation is about design, analysis, and evaluation of a set of wide-area traffic management solutions for cloud service providers, including new ways to (i) share information among cooperating parties, (ii) jointly optimize over multiple design variables, and (iii) implement a decentralized solution for wide-area traffic control. We present these solutions on three stages—*sharing information*, *joint control*, and *decentralized implementation*—that today's CSPs can perform to maximize client performance and minimize operational costs:

Sharing Information (Chapter 2). We study how a CSP overcomes the limited visibility by incentivizing information sharing among different parties. In particular, we examine the cooperation between network routing and server selection. With the strong motivation for ISPs to provide content services, they are faced with the question of whether to stay with the current design or to start sharing information. We develop three cooperation models with an increasing amount of information exchange between the ISP and the CDN. We show that straightforward ways to share information can still be quite sub-optimal, and propose a solution that is mutually-beneficial for both the ISP and the CDN. This work sheds light on ways the ISP and the CDN can share information, starting from the current practice, to move towards a full cooperation that is unilaterally-actionable, backward-compatible, and incrementally-deployable.

Joint Control (Chapter 3). Today, CSP traffic management decisions are made independently by administratively separate groups. We study how to enable joint control of multiple traffic management decisions. In particular, we consider a content delivery architecture based on geographically or administratively separate groups of "last-mile" servers (nano data centers) located within users homes. We propose a set of mechanisms to manage joint content replication and request routing within this architecture, achieving both scalability and cost optimality. This work demonstrates an example of joint optimization over content placement and server selection decisions that may have varied resolutions (*e.g.*, inter v.s. intra ISP), and may happen at different timescales (*e.g.*, minutes v.s. several times a day).

Decentralized Implementation (Chapter 4). Today's cloud services are often offered to geographically distributed clients, leading to the need for a decentralized traffic management solution inside a CSP's network. We present DONAR, a distributed system that can offload the burden of server selection, while providing these services with a sufficiently expressive interface for specifying mapping policies. Our solution runs on a set of distributed mapping nodes for directing local client requests, which is simple, efficient, and only requires a lightweight exchange of summary statistics for coordination between mapping nodes. This work exemplifies a decentralized design that is simultaneously scalable, accurate, and effective.

Through the examination of the three systems, we believe our solutions together shed light on the fundamental network support that CSPs should build for their online service. We summarize our key contributions as follows:

- **A timely study of prevalent cloud services and applications:** We use three application examples to provide a set of wide-area networking solutions for CSPs, including: (i) cooperative server selection and traffic engineering for *information sharing*, (ii) content distribution among federated CDNs for *joint control*, and (iii) DONAR: a distributed server selection system for *decentralized implementation*. While we do not enumerate every possible traffic-management task that today's CSP has to handle, many of our solution techniques will be useful to CSPs who wish to deploy efficient, reliable and scalable distributed services.

- **A mathematical framework for CSP traffic management:** We provide a mathematical framework to formulate CSP traffic management, including *server selection*, *network routing*, and *content placement*. Such a framework also allows us to accurately define key performance and cost metrics that are common to a wide range of online services and applications.

- **Practical optimization formulation:** We provide optimization formulations for CSPs to maximize performance or minimize cost. Our practical problem formulation also allows CSPs to freely express many policy and operational constraints that are often considered today.

- **Scalable solution architecture:** We propose scalable traffic management solutions through separation of control and distributed algorithms. We design a set of simple system architectures for CSPs to implement these solutions through local measurements and message passing that involve a judicious amount of communication overhead.

- **Evaluation with real traffic traces:** We evaluate the benefits of our proposed solutions through trace-driven simulations that employ real networks and traffic traces, including: (i) realistic backbone topologies of tier-1 ISPs, (ii) content download traces from the biggest BitTorrent network Vuze, and (iii) client request traces from an operational CDN CoralCDN. We demonstrate that our solutions are in practice effective, scalable, and adaptive.

- **Design, implementation, and deployment of system prototypes:** Together with our collaborators, we apply the distributed algorithm design DONAR, a decentralized server-selection service that is implemented, prototyped, and deployed on CoralCDN and the Measurement Lab. Through live experiments with DONAR, we demonstrate that our solution performs well "in the wild".

The advent of cloud computing presents new opportunities for traffic management across wide-area networks. While this problem has long existed since the birth of the Internet, today's cloud service provider are faced with the dramatically increasing scale of content-centric, geographically-distributed services that run across data centers, making it more significant. Today's CSPs usually rely on *ad hoc* techniques for controlling their traffic across data centers and to and from clients. This dissertation proposes a set of networking solutions for CSPs who wish to offer better performance to users at a lower cost. Our research develops new methods to make this process more systematic, and offer a new design paradigm for effective, automated techniques for scalable wide-area traffic control.

Chapter 2

Cooperative Server Selection and Traffic Engineering in an ISP Network

This chapter focuses on the challenge of *sharing information* among administratively separated groups that are in charge of different traffic management decisions [13]. Traditionally, ISPs make profit by providing Internet connectivity, while CPs play the more lucrative role of delivering content to users. As network connectivity is increasingly a commodity, ISPs have a strong incentive to offer content to their subscribers by deploying their own content distribution infrastructure. Providing content services in an ISP network presents new opportunities for coordination between traffic engineering (to select efficient routes for the traffic) and server selection (to match servers with subscribers). In this work, we develop a mathematical framework that considers three models with an increasing amount of cooperation between the ISP and the CP. We show that separating server selection and traffic engineering leads to sub-optimal equilibria, even when the CP is given accurate and timely information about the ISP's network in a partial cooperation. More surprisingly, extra visibility may result in a less efficient outcome and such performance degradation can be unbounded. Leveraging ideas from cooperative game theory, we propose an architecture based on the concept of Nash bargaining solution. Simulations on realistic backbone topologies are performed to quantify the performance differences among the three models. Our results apply both when a network provider attempts to provide content, and when separate ISP and CP entities wish to cooperate. This study is a step toward a systematic understanding of the interactions between those who provide and operate networks and those who generate and distribute content.

2.1 Introduction

ISPs and CPs are traditionally independent entities. ISPs only provide connectivity, or the "pipes" to transport content. As in most transportation businesses, connectivity and bandwidth are becoming commodities and ISPs find their profit margin shrinking [14]. At the same time, content providers generate revenue by utilizing existing connectivity to deliver content to ISPs' customers. This motivates ISPs to host and distribute content to their customers. Content can be enterprise-oriented, like web-based services, or residential-based, like triple play as in AT&T's U-Verse [15] and Verizon FiOS [16] deployments. When ISPs and CPs operate independently, they optimize their performance without much cooperation, even though they influence each other indirectly. When ISPs deploy content services or seek cooperation with CP, they face the question of how much can be gained from such cooperation and what kind of cooperation should be pursued.

A traditional service provider's primary role is to deploy infrastructure, manage connectivity, and balance traffic load inside its network. In particular, an ISP solves the *traffic engineering* (TE) problem, *i.e.*, adjusting the routing configuration to the prevailing traffic. The goal of TE is to ensure efficient routing to minimize congestion, so that users experience low packet loss, high throughput, and low latency, and that the network can gracefully absorb flash crowds.

To offer its own content service, an ISP replicates content over a number of strategically-placed servers and directs requests to different servers. The CP, whether as a separate business entity or as a new part of an ISP, solves the *server selection* (SS) problem, *i.e.*, determining which servers should deliver content to each end user. The goal of SS is to meet user demand, minimize network latency to reduce user waiting time, and balance server load to increase throughput.

To offer both network connectivity and content delivery, an ISP is faced with coupled TE and SS problems, as shown in Figure 2.1. TE and SS interact because TE affects the routes that carry the CP's traffic, and SS affects the offered load seen by the network. Actually, the degrees of

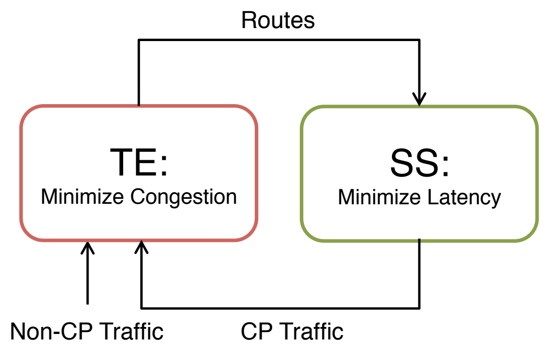


Figure 2.1: The interaction between traffic engineering (TE) and server selection (SS).

freedom are also the "mirror-image" of each other: the ISP controls routing matrix, which is the constant parameter in the SS problem, while the CP controls traffic matrix, which is the constant parameter in the TE problem.

In this chapter, we study several approaches an ISP could take in managing traffic engineering and server selection, ranging from running the two systems independently to designing a joint system. We refer to CP as the part of the system that manages server selection, whether it is performed directly by the ISP or by a separate company that cooperates with the ISP. This study allows us to explore a migration path from the status-quo to different models of synergistic traffic management. In particular, we consider three scenarios with increasing amounts of cooperation between traffic engineering and server selection:

- **Model I:** no cooperation (current practice).
- **Model II:** improved visibility (sharing information).
- **Model III:** a joint design (sharing control).

Model I. Content services could be provided by a CDN that runs independently on the ISP network. However, the CP has limited visibility into the underlying network topology and routing, and therefore has limited ability to predict user performance in a timely and accurate manner. We model a scenario where the CP measures the end-to-end latency of the network and greedily assigns each user to the servers with the lowest latency to the user, a strategy some CPs employ today [17]. We call this *SS with end-to-end info*. In addition, TE assumes the offered traffic is unaffected by its routing decisions, despite the fact that routing changes can affect path latencies and therefore the CP's traffic. When the TE problem and the SS problem are solved separately, their interaction can be modeled as a game in which they take turns to optimize their own networks and settle in a Nash equilibrium, which may not be *Pareto optimal*.

Not surprisingly, performing TE and SS independently is often sub-optimal because (i) server selection is based on incomplete (and perhaps inaccurate) information about network conditions and (ii) the two systems, acting alone, may miss opportunities for a joint selection of servers and routes. Models II and III capture these two issues, allowing us to understand which factor is more important in practice.

Model II. Greater visibility into network conditions should enable the CP to make better decisions. There are, in general, four types of information that could be shared: (i) physical topology information [18, 19, **?**], (ii) logical connectivity information, *e.g.*, routing in the ISP network, (iii) dynamic properties of links, *e.g.*, OSPF link weights, background traffic, and congestion level, and (iv) dynamic properties of nodes, *e.g.*, bandwidth and processing power that can be shared. Our work focuses on a combination of these types of information, *i.e.*, (i)-(iii), so that the CP is able to solve the SS problem more efficiently, *i.e.*, to find the *optimal server selection*.

Sharing information requires minimal extensions to existing solutions for TE and SS, making it amenable to incremental deployment. Similar to the results in the parallel work [20], we observe and prove that TE and SS separately optimizing over their own variables is able to converge to a global optimal solution, when the two systems share the same objectives with the absence of background traffic. However, when the two systems have different or even conflicting performance objectives (*e.g.*, SS minimizes end-to-end latency and TE minimizes congestion), the equilibrium is *not* optimal. In addition, we find that model II sometimes performs *worse* than model I—that is, extra visibility into network conditions sometimes leads to a *less efficient* outcome—and the CP's latency degradation can be unbounded. The facts that both Model I and Model II in general do not achieve optimality, and that extra information (Model II) sometimes hurts the performance, motivate us to consider a clean-slate joint design for selecting servers and routes next.

Model III. A joint design should achieve *Pareto optimality* for TE and SS. In particular, our joint design's objective function gives rise to *Nash Bargaining Solution* [21]. The solution not only

Model	Optimality	Information	Fairness	Arch. Change
I	Large gap Not Pareto-optimal	No exchange Measurement only	No	Current practice
II	Not Pareto-optimal Special case global-optimal More info. may hurt	Topology, capacity Routing Background traffic	No	Minor CP changes Better SS algorithm
III	Pareto-optimal 5-30% improvement	Topology Link prices	Yes	Clean-slate design Incrementally deployable CP given more control

Table 2.1: Summary of results and engineering implications.

guarantees *efficiency*, but also *fairness* between synergistic or even conflicting objectives of two players. It is a point on the Pareto optimal curve where both TE and SS have better performance compared to the Nash equilibrium. We then apply the optimization decomposition technique [22] so that the joint design can be implemented in a distributed fashion with a limited amount of information exchange.

The analytical and numerical evaluation of these three models allows us to gain insights for designing a cooperative TE and SS system, summarized in Table 2.1. The conventional approach of Model I requires minimum information passing, but suffers from sub-optimality and unfairness. Model II requires only minor changes to the CP's server selection algorithm, but the result is still not Pareto optimal and performance is not guaranteed to improve, even possibly degrading in some cases. Model III ensures optimality and fairness through a distributed protocol, requires a moderate increase in information exchange, and is incrementally deployable. Our results show that letting CP have some control over network routing is the key to effective TE and SS cooperation.

We perform numerical simulations on realistic ISP topologies, which allow us to observe the performance gains and losses over a wide range of traffic conditions. The joint design shows significant improvement for both the ISP and the CP. The simulation results further reveal the impact of topologies on the efficiencies and fairness of the three system models.

Our results apply both when a network provider attempts to provide content, and when separate ISP and CP entities wish to cooperate. For instance, an ISP playing both roles would find the optimality analysis useful such that a low efficiency operating region can be avoided. And cooperative ISP and CP would appreciate the distributed implementation of Nash bargaining solution that allows for an incremental deployment.

The rest of this chapter is organized as follows. Section 2.2 presents a standard model for traffic engineering. Section 2.3 presents our two models for server selection, when given minimal information (*i.e.*, Model I) and more information (*i.e.*, Model II) about the underlying network. Section 2.4 studies the interaction between TE and SS as a game and shows that they reach a Nash equilibrium. Section 2.5 analyzes the efficiency loss of Model I and Model II in general. We show that the Nash equilibria achieved in both models are not Pareto optimal. In particular, we show that more information is not always helpful. Section 2.6 discusses how to jointly optimize TE and SS by implementing a Nash bargaining solution. We propose an algorithm that allows practical and incremental implementation. We perform large-scale numerical simulations on realistic ISP topologies in Section 2.7. Finally, Section 2.8 presents related work, and Section 2.9 concludes the chapter and discusses our future work.

2.2 Traffic Engineering (TE) Model

In this section, we describe the network model and formulate the optimization problem that the standard TE model solves. We also start introducing the notation used in this work, which is summarized in Table 2.2.

Consider a network represented by graph $G = (V, E)$, where V denotes the set of nodes and E denotes the set of directed physical links. A node can be a router, a host, or a server. Let x_{ij} denote the rate of flow (i, j), from node i to node j, where $i, j \in V$. Flows are carried on end-to-end paths consisting of some links. One way of modeling routing is $W = \{w_{pl}\}$, *i.e.*, $w_{pl} = 1$ if link l is on path p, and 0 otherwise. We do not limit the number of paths so W can include *all* possible paths, but in practice it is often pruned to include only paths that actually carry traffic. The capacity of a link $l \in E$ is $C_l > 0$.

Given the traffic demand, traffic engineering changes routing to minimize network congestion. In practice, network operators control routing either by changing OSPF (Open Shortest Path First) link weights [23] or by establishing MPLS (Multiprotocol Label Switch) label-switched paths [24]. In this paper we use the multi-commodity flow solution to route traffic, because a) it is optimal, *i.e.*, it gives the routing with minimum congestion cost, and b) it can be realized by routing protocols that use MPLS tunneling, or as recently shown, in a distributed fashion by a

Notation	**Interpretation**
G	Network graph $G = (V, E)$. V set of nodes, E set of links
S	$S \subset V$, the set of CP servers
T	$T \subset V$, the set of users
C_l	Capacity of link l
r_l^{ij}	Proportion of flow $i \to j$ traversing link l
R	The routing matrix $R : \{r_l^{ij}\}$, TE variable
R_{bg}	Background routing matrix $R : \{r_l^{ij}\}_{(i,j) \notin S \times T}$
X	Traffic matrix of all communication pairs $X = \{x_{ij}\}_{(i,j) \in V \times V}$
x_{st}	Traffic rate from server s to user t
X_{cp}	$X_{cp} = \{x_{st}\}_{(s,t) \in S \times T}$, SS variable
M_t	User t's demand rate for content
B_s	Service capacity of server s
x_l^{st}	The amount of traffic for (s, t) pair on link l
$\hat{X}_{cp}$	$\hat{X}_{cp} = \{x_l^{st}\}_{(s,t) \in S \times T}$, the generalized SS variable
f_l^{cp}	CP's traffic on link l
f_l^{bg}	Background traffic on link l
f_l	$f_l = f_l^{cp} + f_l^{bg}$, total traffic on link l. $\vec{f} = \{f_l\}_{l \in E}$
D_p	Delay of path p
D_l	Delay of link l
$g(\cdot)$	Cost function used in ISP traffic engineering
$h(\cdot)$	Cost function used in CP server selection

Table 2.2: Summary of key notation.

new link-state routing protocol PEFT [25]. Let $r_l^{ij} \in [0, 1]$ denote the proportion of traffic of flow (i, j) that traverses link l. To realize the multi-commodity flow solution, the network splits each flow over a number of paths. Let $R = \{r_l^{ij}\}$ be the routing matrix.

Let f_l denote the total traffic traversing link l, and we have $f_l = \sum_{(i,j)} x_{ij} \cdot r_l^{ij}$. Now traffic engineering can be formulated as the following optimization problem:

TE(X)

$$\text{minimize} \quad TE = \sum_l g_l(f_l) \tag{2.1a}$$

$$\text{subject to} \quad f_l = \sum_{(i,j)} x_{ij} \cdot r_l^{ij} \le C_l, \ \forall l \tag{2.1b}$$

$$\sum_{l:l \in \text{In}(v)} r_l^{ij} - \sum_{l:l \in \text{Out}(v)} r_l^{ij} = I_{v=j}, \ \forall (i,j), \ \forall v \in V \backslash \{i\} \tag{2.1c}$$

$$\text{variables} \quad 0 \le r_l^{ij} \le 1, \ \forall (i,j), \ \forall l \tag{2.1d}$$

where $g_l(\cdot)$ represents a link's congestion cost as a function of the load, $I_{v=j}$ is an indicator function which equals 1 if $v = j$ and 0 otherwise, $\mathrm{In}(v)$ denotes the set of incoming links to node v, and $\mathrm{Out}(v)$ denotes the set of outgoing links from node v.

In this model, TE does not differentiate between the CP's traffic and background traffic. In fact, TE assumes a constant traffic matrix X, *i.e.*, the offered load between each pair of nodes, which can either be a point-to-point background traffic flow, or a flow from a CP's server to a user. As we will see later, this common assumption is undermined when the CP performs dynamic server selection.

For computational tractability, ISPs usually consider cost functions $g_l(\cdot)$ that are convex, continuous, and non-decreasing. By using such an objective, TE penalizes high link utilization and balances load inside the network. We follow this approach and discuss the analytical form of $g_l(\cdot)$ that ISPs use in practice in a later section.

2.3 Server Selection (SS) Models

While traffic engineering usually assumes that traffic matrix is point-to-point and constant, both assumptions are violated when some or all of the traffic is generated by the CP. A CP usually has many servers that offer the same content, and the servers selected for each user depend on the network conditions. In this section, we present two novel CP models which correspond to models I and II introduced in Section 2.1. The first one models the current CP operation, where the CP relies on end-to-end *measurement* of the network condition in order to make server selection decisions; the second one models the situation when the CP obtains enough information from the ISP to *calculate* the effect of its actions.

2.3.1 Server Selection Problem

The CP solves the server selection problem to optimize the perceived performance of all of its users. We first introduce the notation used in modeling server selection. In the ISP's network, let $S \subset V$ denote the set of CP's servers, which are strategically placed at different locations in the network. For simplicity we assume that all content is duplicated at all servers, and our results can be extended to the general case. Let $T \subset V$ denote the set of users who request content from the servers. A user $t \in T$ has a demand for content at rate M_t, which we assume to be constant during the time a CP optimizes its server section. We allow a user to simultaneously download content from multiple servers, because node t can be viewed as an edge router in the ISP's network that aggregates the traffic of many endhosts, which may be served by different servers.

To differentiate the CP's traffic from background traffic, we denote x_{st} as the traffic rate from server s to user t. To satisfy the traffic demand, we need

$$\sum_{s \in S} x_{st} = M_t.$$

In addition, the total amount of traffic aggregated at a server s is limited by its service capacity B_s, *i.e.*,

$$\sum_{t \in T} x_{st} \leq B_s.$$

We denote $X_{cp} = \{x_{st}\}_{s \in S, t \in T}$ as the CP's decision variable.

One of the goals in server selection is to optimize the overall performance of the CP's customers. We use an additive link cost for the CP based on latency models, *i.e.*, each link has a cost, and the end-to-end path cost is the sum of the link costs along the way. As an example, suppose the content is delay-sensitive (*e.g.*, IPTV), and the CP would like to minimize the average or total end-to-end delay of all its users. Let D_p denote the end-to-end latency of a path p, and $D_l(f_l)$ denote the latency of link l, modeled as a convex, non-decreasing, and continuous function of the amount of flow f_l on the link. By definition, $D_p = \sum_{l \in p} D_l(f_l)$. By making the x_{st} decisions, the CP implicitly decides the network flow, and as a consequence the overall latency experienced by CP users, which can be rewritten as:

$$\begin{aligned} SS &= \sum_{(s,t)} \sum_{p \in P(s,t)} x_p^{st} \cdot D_p(f) \\ &= \sum_{(s,t)} \sum_{p \in P(s,t)} x_p^{st} \cdot \sum_{l \in p} D_l(f_l) \\ &= \sum_{l} D_l(f_l) \cdot \sum_{(s,t)} \sum_{p \in P(s,t): l \in p} x_p^{st} \\ &= \sum_{l} f_l^{cp} \cdot D_l(f_l) \qquad (2.2) \end{aligned}$$

where $P(s,t)$ is the set of paths serving flow (s,t) and x_p^{st} is the amount of flow (s,t) traversing path $p \in P(s,t)$.

Let $h_l(\cdot)$ represent the cost of link l, which we assume is convex, non-decreasing, and continuous. In this example, $h_l(f_l^{cp}, f_l) = f_l^{cp} \cdot D_l(f_l)$. Thus, the link cost $h_l(\cdot)$ is a function of the CP's total traffic f_l^{cp} on the link, as well as the link's total traffic f_l, which also includes background traffic.

Expression (2.2) provides a simple way to calculate the total user-experienced end-to-end delay—simply sum over all the *links*, but it requires the knowledge of the load on each link, which is possible only in Model II. Without such knowledge (Model I), the CP can rely only on *end-to-end* measurement of delay.

2.3.2 Server Selection with End-to-end Info: Model I

In today's Internet architecture, a CP does not have access to an ISP's network information, such as topology, routing, link capacity, or background traffic. Therefore a CP relies on measured or inferred information to optimize its performance. To minimize its users' latencies, for instance, a CP can assign each user to servers with the lowest (measured) end-to-end latency to the user. In practice, content distribution networks like Akamai's server selection algorithm is based on this principle [17]. We call it *SS with end-to-end info* and use it as our first model.

CP monitors the latencies from all servers to all users, and makes server selection decisions to minimize users' total delay. Since the demand of a user can be arbitrarily divided among the servers, we can think of the CP as greedily assigning each infinitesimal demand to the best server. The placement of this traffic may change the path latency, which is monitored by the CP. Thus, at the equilibrium, the servers which send (non-zero) traffic to a user should have the same end-to-end latency to the user, because otherwise the server with lower latency will be assigned more demand, causing its latency to increase, and the servers not sending traffic to a user should have higher latency than those that serve the user. This is sometimes called the *Wardrop equilibrium* [26]. The SS model with end-to-end info is very similar to selfish routing [27, 28], where each flow tries to minimize its average latency over multiple paths without coordinating with other flows. It is known that the equilibrium point in selfish routing can be viewed as the solution to a global convex optimization problem [27]. Therefore, SS with end-to-end info has a unique equilibrium point under mild assumptions.

Although the equilibrium point is well-defined and is the solution to a convex optimization problem, in general it is hard to compute the solution analytically. Thus we leverage the idea of Q-learning [29] to implement a distributed iterative algorithm to find the equilibrium of SS with end-to-end info. The algorithm is guaranteed to converge even under dynamic network environments with cross traffic and link failures, and hence can be used in practice by the CPs. The detailed description and implementation can be found in [30]. We show in Section 2.5 that, SS with end-to-end info is sub-optimal. We use it as a baseline for how well a CP can do with only the end-to-end latency measurements.

2.3.3 Server Selection with Improved Visibility: Model II

We now describe how a CP can optimize server selection given *complete* visibility into the underlying network, but not into the ISP objective. That is, this is the best the CP can do without *changing* the routing in the network. We also present an optimization formulation that allows us to analytically study its performance.

Suppose that content providers are able to either obtain information on network conditions directly from the ISP, or infer it by its measurement infrastructure. In the best case, the CP is able to obtain the complete information about the network, *i.e.*, routing decision and link latency. This situation is characterized by problem (2.3). To optimize the overall user experience, the CP solves the following cost minimization problem:

$$\begin{aligned}
&\mathbf{SS}(R) && \\
&\text{minimize} && SS = \sum_l h_l(f_l^{cp}, f_l) && (2.3a)\\
&\text{subject to} && f_l^{cp} = \sum_{(s,t)} x_{st} \cdot r_l^{st},\ \forall l && (2.3b)\\
& && f_l = f_l^{cp} + f_l^{bg} \leq C_l,\ \forall l && (2.3c)\\
& && \sum_{s\in S} x_{st} = M_t,\ \forall t && (2.3d)\\
& && \sum_{t\in T} x_{st} \leq B_s,\ \forall s && (2.3e)\\
&\text{variables} && x_{st} \geq 0,\ \forall (s,t) && (2.3f)
\end{aligned}$$

where we denote $f_l^{bg} = \sum_{(i,j)\neq(s,t)} x_{ij} \cdot r_l^{ij}$ as the non-CP traffic on link l, which is a parameter to the optimization problem. If the cost function $h_l(\cdot)$ is increasing and convex on the variable f_l^{cp}, one can verify that (2.3) is a convex optimization problem, hence has a unique global optimal value. To ease our presentation, we relax the server capacity constraint by assuming very large server bandwidth caps in the remainder of this chapter. Our results hold in general cases when the server constraints exist.

SS with improved visibility (2.3) is amenable to an efficient implementation. The problem can either be solved centrally, *e.g.*, at the CP's central coordinator, or via a distributed algorithm similar to that used for Model I. We solve (2.3) centrally in our simulations, since we are more interested in the performance improvement brought by complete information than any particular algorithm for implementing it.

2.4 Analyzing TE-SS Interaction

In this section, we study the interaction between the ISP and the CP when they operate independently without coordination in both Model I and Model II, using a game-theoretic model. The game formulation allows us to analyze the stability condition, *i.e.*, we show that alternating TE and SS optimizations will reach an equilibrium point. In addition, we find that when the ISP and the CP optimize the same system objective, their interaction achieves *global optimality* under Model II. Results in this section are also found in a parallel work [20].

2.4.1 TE-SS Game and Nash Equilibrium

We start with the formulation of a two-player non-cooperative Nash game that characterizes the TE-SS interaction.

Definition 1. *The* TE-SS game *consists of a tuple* $< N, A, U >$. *The player set* $N = \{isp, cp\}$. *The action set* $A_{isp} = \{R\}$ *and* $A_{cp} = \{X_{cp}\}$, *where the feasible set of* R *and* X_{cp} *are defined by the constraints in* (2.1) *and* (2.3) *respectively. The utility functions are* $U_{isp} = -TE$ *and* $U_{cp} = -SS$.

Figure 2.1 shows the interaction between SS and TE. In both Model I and Model II, the ISP chooses the best response strategy, *i.e.*, the ISP always optimizes (2.1) given the CP's strategy X_{cp}.

Similarly, the CP chooses the best response strategy in Model II by solving (2.3). However, the CP's strategy in Model I is not the best response, since it is not able to optimize the objective (2.3) due to poor network visibility. Indeed, the utility the CP implicitly optimizes in SS with end-to-end info is [27]

$$U_{cp} = -\sum_{l \in E} \int_0^{f_l} D_l(u) du$$

This later helps us understand the stability conditions of the game.

Consider a particular game procedure in which the ISP and the CP take turns to optimize their own objectives by varying their own decision variables, treating that of the other player as constant. Specifically, in the $(k+1)$-th iteration, we have

$$R^{(k+1)} = \underset{R}{\text{argmin}}\ TE(X_{cp}^{(k)}) \tag{2.4a}$$

$$X_{cp}^{(k+1)} = \underset{X_{cp}}{\text{argmin}}\ SS(R^{(k+1)}) \tag{2.4b}$$

Note that the two optimization problems may be solved at different timescales. The ISP runs traffic engineering at the timescale of hours, although it could run on a much smaller timescale. Depending on the CP's design choices, server selection is optimized a few times a day, or at a smaller timescale like seconds or minutes of a typical content transfer duration. We assume that each player has fully solved its optimization problem before the other one starts.

Next we prove the existence of Nash equilibrium of the TE-SS game. We establish the stability condition when two players use general cost functions $g_l(\cdot)$ and $h_l(\cdot)$ that are continuous, non-decreasing, and convex. While TE's formulation is the same in Model I and Model II, we consider the two SS models, *i.e.*, SS with end-to-end info and SS with improved visibility.

Theorem 2. *The TE-SS game has a Nash equilibrium for both Model I and Model II.*

Proof. It suffices to show that (i) each player's strategy space is a nonempty compact convex subset, and (ii) each player's utility function is continuous and quasi-concave on its strategy space, and follow the standard proof in [31]. The ISP's strategy space is defined by the constraint set of (2.1), which are affine equalities and inequalities, hence a convex compact set. Since $g_l(\cdot)$ is continuous and convex, we can easily verify that the objective function (2.1a) is quasi-convex on $R = \{r_l^{ij}\}$. CP's strategy space is defined by the constraint set of (2.3), which is also convex and compact.

Similarly, if $h_l(f_l^{cp})$ is continuous and convex, the objective function (2.3a) is quasi-convex on X_{cp}. In particular, consider the special case in which CP minimizes latency (2.2). When CP solves SS with end-to-end info, $h_l(f_l) = \int_0^{f_l} D_l(u)du$. When CP solves SS with improved visibility, $h_l(f_l^{cp}) = f_l^{cp} D_l(f_l)$. In both cases, if $D_l(\cdot)$ is continuous, non-decreasing, and convex, so is $h_l(\cdot)$. $\square$

While there exists a Nash equilibrium, it does not guarantee that alternating optimizations (2.4) lead to one. In Section 2.7 we demonstrate the convergence of alternating optimizations through simulation. In general, the Nash equilibrium may not be unique, in terms of both decision variables and objective values. Next, we discuss a special case where the Nash equilibrium is unique and can be attained by alternating optimizations (2.4).

2.4.2 Global Optimality under Same Objective and Absence of Background Traffic

In the following, we consider a special case of the TE-SS game, in which the ISP and the CP optimize the *same* objective function, *i.e.*, $g_l(\cdot) = h_l(\cdot)$, so

$$TE = SS = \sum_l \Phi_l(f_l), \tag{2.5}$$

when there is *no background traffic*. One example is when the network carries *only* the CP traffic, and both the ISP and the CP aim to minimize the average traffic latency, *i.e.*, $\Phi_l(f_l) = f_l \cdot D_l(f_l)$. An interesting question that naturally arises is whether the two players' alternating best response to each other's decision can lead to a socially optimal point.

Define a notion of *global optimum*, which is the optimal point to the following optimization problem.

TESS – Special

$$\begin{aligned}
\text{minimize} \quad & \sum_l \Phi_l(f_l) & \text{(2.6a)}\\
\text{subject to} \quad & f_l = \sum_{(s,t)} x_l^{st} \le C_l,\ \forall l & \text{(2.6b)}\\
& \sum_{s\in S}\left(\sum_{l:l\in \mathrm{In}(v)} x_l^{st} - \sum_{l:l\in \mathrm{Out}(v)} x_l^{st}\right) = M_t \cdot I_{v=t},\ \forall v \notin S,\ \forall t\in T & \text{(2.6c)}\\
\text{variables} \quad & x_l^{st} \ge 0,\ \forall (s,t), \forall l & \text{(2.6d)}
\end{aligned}$$

where x_l^{st} denotes the traffic rate for flow (s,t) delivered on link l. The variable x_l^{st} allows a global coordinator to route a user's demand from any server in any way it wants, thus problem (2.6) establishes an upper-bound on how well one can do to minimize the traffic latency. Note that x_l^{st} captures both R and X_{cp}, which offers more degrees of freedom for a joint routing and server-selection problem. Its mathematical properties will be further discussed in Section 2.5.2.

The special case TE-SS game (2.5) has a Nash equilibrium, as shown in Theorem 2. Nash equilibrium may not be unique in general. This is because when there is no traffic between a server-user pair, the TE routing decision for this pair can be arbitrary without affecting its utility. In the worst case, a Nash equilibrium can be arbitrarily suboptimal to the global optimum. Suppose there exists non-zero traffic demand between any server-user pair as considered in [20], *e.g.*, by appending an infinitesimally small traffic load to every server-user pair. We show that alternating optimizations (2.4) reach a *unique* Nash equilibrium, which is also an optimal solution to (2.6). TE-SS interaction does not sacrifice any efficiency in this special case, and the optimal operating point can be achieved by iterative best response unilaterally, without the need for a global coordination. This result is shown in [20] in which the idea is to prove the equivalence of Nash equilibrium and the global optimum. We show an alternative proof by considering alternating projections of variables onto a convex set, which is presented as follows.

Consider the following optimization problem:

$$\begin{aligned}
\text{minimize} \quad & \sum_l \Phi_l(f_l) && \text{(2.7a)}\\
\text{subject to} \quad & f_l = \sum_{(s,t)} x_{st} \cdot r_l^{st} \le C_l,\ \forall l && \text{(2.7b)}\\
& \sum_{l:l\in \mathrm{In}(v)} r_l^{st} - \sum_{l:l\in \mathrm{Out}(v)} r_l^{st} = I_{v=t},\ \forall (s,t),\ \forall v \in V\backslash\{s\} && \text{(2.7c)}\\
& \sum_{s\in S} x_{st} = M_t,\ \forall t && \text{(2.7d)}\\
\text{variables} \quad & 0 \le r_l^{st} \le 1,\ x_{st} \ge 0 && \text{(2.7e)}
\end{aligned}$$

The alternating optimizations (2.4) in the special case TE-SS game is solving (2.7) by applying the non-linear Gauss-Seidel algorithm [32], which consists of alternating projections of one variable onto the steepest descending direction while keeping the other fixed. Note that (2.7) is a non-convex problem, since it involves the product of two variables r_l^{st} and x_{st}. However, we next show that it is equivalent to the convex problem (2.6).

Lemma 3. *The non-convex problem (2.7) that the TE-SS game solves is equivalent to the convex problem (2.6).*

Proof. We show that there is a one-to-one mapping between the feasible solutions of (2.6) and (2.7). Consider a feasible solution $\{x_l^{st}\}$ of (2.6). Let $x_{st} = \sum_{l:l\in \mathrm{In}(t)} x_l^{st} - \sum_{l:l\in \mathrm{Out}(t)} x_l^{st}$, $r_l^{st} = x_l^{st}/x_{st}$ if $x_{st} \neq 0$. To avoid the case of $x_{st} = 0$, suppose there is infinitesimally small background traffic for every (s,t) pair, so the one-to-one mapping holds. On the other hand, for each feasible solution $\{x_{st}, r_l^{st}\}$ of (2.7), let $x_l^{st} = x_{st} \cdot r_l^{st}$. It is easy to see that for every feasible solution $\{x_l^{st}\}$, the derived $\{x_{st}, r_l^{st}\}$ is also feasible, and vice versa. Since the two problems have the same objective function, they are equivalent. $\square$

Lemma 4. *The Nash equilibrium of TE-SS game is an optimal solution to* (2.6)*.*

Proof. The key idea of the proof is to check the KKT conditions [12] of TE and SS optimization problems at Nash equilibrium (step I), and show that they also satisfy the KKT condition of the global problem (2.6) (step II).

Step I: Consider a feasible solution $\{x_{st}, r_l^{st}\}$ at Nash equilibrium, *i.e.*, each one is the best response of the other. To assist our proof, we define $\phi_l(f_l) = \Phi'(f_l)$ as the marginal cost of link l.

We first show the optimality condition of SS. Let $\phi_{st} = \sum_l \phi_l(f_l) \cdot r_l^{st}$, which denotes the marginal cost of (s, t) pair. By the definition of Nash equilibrium, for any s such that $x_{st} > 0$, we have $\phi_{st} \leq \phi_{s't}$ for any $s' \in S$, by inspecting the KKT condition of the SS optimization. This implies that servers with positive rate have the same marginal latency, which is less than those of servers with zero rate. Let $\phi_t = \phi_{st}$ for all $x_{st} > 0$.

We next show the optimality condition of TE. Consider an (s, t) server-user pair. Let δ_{sv} denote the average marginal cost from node s to v, which can be recursively defined as

$$\delta_{sv} = \begin{cases} \sum_{l:(u,v)\in \mathrm{In}(v)} (\delta_{su} + \phi_l) \cdot r_l^{st} / \sum_{l \in \mathrm{In}(v)} r_l^{st} & \text{if } v \neq s \\ 0 & \text{if } v = s \end{cases}$$

The KKT condition of the TE optimization is for $\forall v \in V$, $\forall l = (u, v), l' = (u', v) \in \mathrm{In}(v)$, $r_l^{st} > 0$ implies $\delta_{su} + \phi_l \leq \delta_{su'} + \phi_{l'}$. In other words, for any node v, the marginal cost accumulated from any incoming link with positive flow is equal, and less than those of incoming links with zero flow. So we can define $\delta_{sv} = \delta_{su} + \phi_l$, $\forall l = (u, v) \in \mathrm{In}(v)$ with $r_l^{st} \geq 0$. In fact, $\delta_{st} = \sum_{l:(u,t)} (\delta_{su} + \phi_l) \cdot r_l^{st} = \sum_l \phi_l \cdot r_l^{st} = \phi_{st}$, by inspecting flow conservation at each node and the fact that any (s, t) path has the same marginal latency as observed above. Combining the two KKT conditions together gives us the necessary and sufficient condition for Nash equilibrium:

$$\begin{cases} \delta_{su} + \phi_l \leq \delta_{su'} + \phi_{l'} \text{ if } r_l^{st} > 0 & \forall s \in S,\ \forall v \in V,\ \forall l, l' \in \mathrm{In}(v) \\ \delta_{st} \leq \delta_{s't},\ \text{if } x_{st} > 0 & \forall s, s' \in S \end{cases} \tag{2.8}$$

An intuitive explanation is to consider the marginal latency of any path p that is realized by the routing decision. Let $P(t)$ be the set of paths that connect all possible servers and the user t. Let $\phi_p = \sum_{l:l\in p} \phi_l$. A path p is *active* if $r_l^{st} > 0$ for all $l \in p$, which means there is a positive flow between (s, t). Then the above condition can be translated into the following argument: for any path $p, p' \in P(t)$, $\phi_p \leq \phi_{p'}$ if p is active. In other words, any active path has the same marginal latency, which is less than those of non-active paths.

Step II: We show the KKT condition of (2.6). Let $\{x_l^{st}\}$ be an optimal solution to (2.6). Similarly, we define the marginal latency from node s to v as

$$\Delta_{sv} = \begin{cases} \sum_{l:(u,v)\in \mathrm{In}(v)} (\Delta_{su} + \phi_l) \cdot x_l^{st} / \sum_{l\in \mathrm{In}(v)} x_l^{st} & \text{if } v \neq s \\ 0 & \text{if } v = s \end{cases}$$

The KKT condition of (2.6) is the following:

$$\begin{cases} \Delta_{su} + \phi_l \leq \Delta_{su'} + \phi_{l'} \text{ if } x_l^{st} > 0 \quad \forall s \in S,\ \forall v \in V,\ \forall l, l' \in \mathrm{In}(v) \\ \Delta_{st} \leq \Delta_{s't},\ \text{if } x_l^{st} > 0 \text{ for some } l \quad \forall s, s' \in S \end{cases} \tag{2.9}$$

One can readily check the equivalence of conditions (2.8) and (2.9). To be more specific, suppose $\{x_{st}, r_l^{st}\}$ is a Nash equilibrium that satisfies (2.8), we can construct $\{x_l^{st}\}$ as discussed in the proof of Lemma 3, which also satisfies (2.9), and vice versa. □

Lemma 5. *The alternating TE and SS optimizations (2.4) converge to a Nash equilibrium of the TE-SS game.*

Proof. Consider the objective of (2.7), which is also a Lyapunov function. Since two players have the same utility function, and each step of (2.4) is one's best response by fixing the other's decision, the trajectory of the objective value is a decreasing sequence. In addition, the objective of (2.7) is lower-bounded by the optimal value of (2.6). Therefore, there exists a limit point of the sequence. It remains to show that this limit point is indeed an equilibrium.

Consider the sequence $(X_{cp}^{(k)}, R^{(k)})$, where $k = 1, 2, \ldots, \infty$ is the step index. Denote the limit point of the objective (2.7a) by Φ^*. Since the feasible space of $\{X_{cp}, R\}$ is compact and continuous, there exists a limit point (X_{cp}^*, R^*) as $k \to \infty$. By the definition of (2.7a), $X^* = \mathrm{argmin}_{X_{cp}} SS(R^*)$. To show that R^* is also a best response to X^*, suppose by contradiction that there exists $\tilde{R}$ such that with X^* there is a lower objective value $\tilde{\Phi} < \Phi^*$. By the continuity of the objective function, there exists large k such that $(\tilde{R}, X^{(k)})$ will result in an objective value $\bar{\Phi}$ within a $\epsilon-$ball of $\tilde{\Phi}$ for arbitrarily small ϵ, namely, $\bar{\Phi} \leq \tilde{\Phi} + \epsilon \leq \Phi^*$. Therefore, the best response to $X^{(k)}$ will result in an objective value no greater than $\tilde{\Phi} + \epsilon$, which is less than Φ^*. This contradicts the fact that the sequence is lower-bounded by Φ^*. Thus we complete the proof that (X_{cp}^*, R^*) is a Nash equilibrium. □

Theorem 6. *The alternating optimizations of TE and SS* (2.4) *in the special TE-SS game (2.5) achieves the global optimum of* (2.6).

Proof. Combining Lemmas 3, 4, and 5 leads to the statement. □

The special case analysis establishes a lower bound on the efficiency loss of TE-SS interaction. In general, there are two sources of mis-alignment between the optimization problems of TE and SS: (i) different shapes of the cost functions, *e.g.*, the delay function, and (ii) the existence of background traffic in the TE problem. The above special case illustrates what might happen if these differences are avoided. However, in general, such mis-alignment will lead to a significant efficiency loss, as we show in the next section. The evaluation results shown in Section 2.7 further highlight the difference (i).

2.5 Efficiency Loss

We next study the efficiency loss in the general case of the TE-SS game, which may be caused by incomplete information, or unilateral actions that miss the opportunity to achieve a jointly attainable optimal point. We present two case studies that illustrate these two sources of suboptimal performance. We first present a toy network and show that under certain conditions the CP performs even worse in Model II than Model I, despite having more information about underlying network conditions. We next propose the notion of Pareto-optimality as the performance benchmark, and quantify the efficiency loss in both Model I and Model II.

2.5.1 The Paradox of Extra Information

Consider an ISP network illustrated in Figure 2.2. We designate an end user node, $T = \{F\}$, and two CP servers, $S = \{B, C\}$. The end user has a content demand of $M_F = 2$. We also allow two background traffic flows, $A \to D$ and $A \to E$, each of which has one unit of traffic demand. Edge directions are noted on the figure, so one can figure out the possible routes, *i.e.*, there are two paths for each traffic flow (clockwise and counter-clockwise). To simplify the analysis and deliver the most essential message from this example, suppose that both TE and SS costs on the four thin links are negligible so the four bold links constitute the *bottleneck* of the network. In Table 2.3,

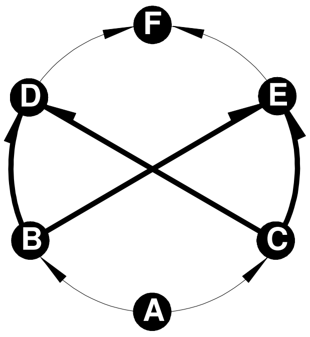


Figure 2.2: An Example of the Paradox of Extra Information

we list the link capacities, ISP's cost function $g_l(\cdot)$, and link latency function $D_l(\cdot)$. Suppose the CP aims to minimize the average latency of its traffic. We compare the Nash equilibrium of two situations when the CP optimizes its network by SS with end-to-end info and SS with improved visibility.

link	$l_1 : BD$	$l_2 : BE$	$l_3 : CD$	$l_4 : CE$
C_l	$1+\epsilon$	$1+\epsilon$	$1+\epsilon$	$1+\epsilon$
$D_l(f_l)$	f_1	$\frac{1}{1+\epsilon-f_2}$	$\frac{1}{1+\epsilon-f_3}$	f_4
$g_l(x)$	$g_1(\cdot) = g_2(\cdot) = g_3(\cdot) = g_4(\cdot)$			

Table 2.3: Link capacities, ISP's and CP's link cost functions in the example of Paradox of Extra Information.

The stability condition for the ISP at Nash equilibrium is $g'_1(f_1) = g'_2(f_2) = g'_3(f_3) = g'_4(f_4)$. Since the ISP's link cost functions are identical, the total traffic on each link must be identical. On the other hand, the stability condition for the CP at Nash equilibrium is that (B, F) and (C, F) have the same marginal latency. Based on the observations, we can derive two Nash equilibrium points.

When the CP takes the strategy of SS with end-to-end info, let

$$\text{Model I:} \begin{cases} X_{CP} : \Big\{ x_{BF} = 1,\ x_{CF} = 1 \Big\} \\ R : \Big\{ r_1^{BF} = 1-\alpha,\ r_2^{BF} = \alpha,\ r_3^{CF} = \alpha,\ r_4^{CF} = 1-\alpha, \\ \quad r_1^{AD} = \alpha,\ r_3^{AD} = 1-\alpha,\ r_2^{AE} = 1-\alpha,\ r_4^{AE} = \alpha \Big\} \end{cases}$$

One can check that this is indeed a Nash equilibrium solution, where $f_1 = f_2 = f_3 = f_4 = 1$, and $D_{BF} = D_{CF} = 1 - \alpha + \alpha/\epsilon$. The CP's objective $SS_{\mathrm{I}} = 2(1 - \alpha + \alpha/\epsilon)$.

When the CP takes the strategy of SS with improved visibility, let

$$\text{Model II:} \begin{cases} X_{CP} : \Big\{ x_{BF} = 1,\ x_{CF} = 1 \Big\} \\ \quad R : \Big\{ r_1^{BF} = \alpha,\ r_2^{BF} = 1 - \alpha,\ r_3^{CF} = 1 - \alpha,\ r_4^{CF} = \alpha, \\ \qquad r_1^{AD} = 1 - \alpha,\ r_3^{AD} = \alpha,\ r_2^{AE} = \alpha,\ r_4^{AE} = 1 - \alpha \Big\} \end{cases}$$

This is a Nash equilibrium point, where $f_1 = f_2 = f_3 = f_4 = 1$, and $d_{BF} = d_{CF} = \alpha(1+\alpha) + (1 - \alpha)(1/\epsilon + (1 - \alpha)/\epsilon^2)$. The CP's objective $SS_{\mathrm{II}} = 2(\alpha + (1 - \alpha)/\epsilon)$.

When $0 < \epsilon < 1, 0 \leq \alpha < 1/2$, we have the counter-intuitive $SS_{\mathrm{I}} < SS_{\mathrm{II}}$: more information may hurt the CP's performance. In the worst case,

$$\lim_{\alpha \to 0, \epsilon \to 0} \frac{SS_{\mathrm{II}}}{SS_{\mathrm{I}}} = \infty$$

i.e., the performance degradation can be unbounded.

This is not surprising, since the Nash equilibrium is generally non-unique, both in terms of equilibrium solutions and equilibrium objectives. When ISP and CP's objectives are mis-aligned, the ISP's decision may route CP's traffic on bad paths from the CP's perspective. In this example, the paradox happens when the ISP route the CP traffic on good paths in Model I (though SS makes decision based on incomplete information), and the ISP mis-routes the CP traffic to bad paths in Model II (though SS gains better visibility). In practice, such a scenario is likely to happen, since the ISP cares about link congestion (link utilization), while the CP cares about latency, which depends not only on link load, but also on propagation delay. Thus ISP and CP's partial collaboration by only passing information is not sufficient to achieve global optimality.

2.5.2 Pareto Optimality and Illustration of Sub-Optimality

As in the above example, one of the causes of sub-optimality is that TE and SS's objectives are not necessarily aligned. To measure efficiency in a system with multiple objectives, a common approach is to explore the *Pareto curve*. For points on the Pareto curve, we cannot improve one objective further without hurting the other. The Pareto curve characterizes the tradeoff of potentially conflicting goals of different parties. One way to trace the tradeoff curve is to optimize a weighted sum of the objectives:

$$\begin{aligned} \text{minimize} \quad & TE + \gamma \cdot SS && (2.10\text{a}) \\ \text{variables} \quad & R \in \mathcal{R},\ X_{cp} \in \mathcal{X}_{cp} && (2.10\text{b}) \end{aligned}$$

where $\gamma \geq 0$ is a scalar representing the relative weight of the two objectives. $\mathcal{R}$ and $\mathcal{X}_{cp}$ are the feasible regions defined by the constraints in (2.1) and (2.3):

$$\mathcal{R} \times \mathcal{X}_{cp} = \Bigg\{ \left(r_l^{ij},\, x_{st}\right) \,\Big|\, 0 \leq r_l^{ij} \leq 1;\ x_{st} \geq 0;\ \sum_{l: l \in \mathrm{In}(v)} r_l^{ij} - \sum_{l: l \in \mathrm{Out}(v)} r_l^{ij} = I_{v=j},\ \forall v \in V \backslash \{i\};$$
$$f_l = \sum_{(i,j)} x_{ij} \cdot r_l^{ij} \leq C_l,\ \forall l \in E;\ \sum_{s \in S} x_{st} = M_t,\ \forall t \in T \Bigg\}$$

The problem (2.10) is not easy to solve. In fact, the objective of (2.10) is no longer convex in variables $\{r_l^{st}, x_{st}\}$, and the feasible region defined by constraints of (2.10) is not convex. One way to overcome this problem is to consider a relaxed decision space that is a superset of the original solution space. Instead of restricting each player to its own operating domain, *i.e.*, ISP controls routing and CP controls server selection, we introduce a joint routing and content delivery problem. Let x_l^{st} denote the *rate* of traffic carried on link l that belongs to flow (s,t). Such a convexification of the original problem (2.10) gives more freedom to joint TE and SS problem. Denote the generalized CP decision variable as $\hat{X}_{cp} = \{x_l^{st}\}_{s \in S, t \in T}$, and $R_{bg} = \{r_l^{ij}\}_{(i,j) \notin S \times T}$ as background routing matrix. Consider the following optimization problem:

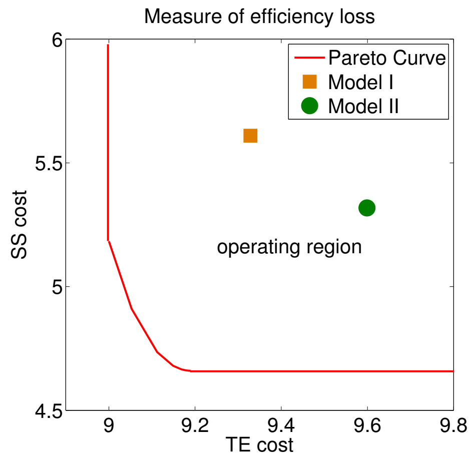


Figure 2.3: A numerical example illustrating sub-optimality.

TESS – weighted

$$\text{minimize} \quad TE + \gamma \cdot SS \tag{2.11a}$$

$$\text{subject to} \quad f_l^{cp} = \sum_{(s,t)} x_l^{st}, \ \forall l \tag{2.11b}$$

$$f_l = f_l^{cp} + \sum_{(i,j)\notin S\times T} x_{ij} \cdot r_l^{ij} \le C_l, \ \forall l \tag{2.11c}$$

$$\sum_{l:l\in \text{In}(v)} r_l^{ij} - \sum_{l:l\in \text{Out}(v)} r_l^{ij} = I_{v=j}, \ \forall (i,j) \notin S \times T, \forall v \in V\backslash\{i\} \tag{2.11d}$$

$$\sum_{s\in S} \left(\sum_{l:l\in \text{In}(v)} x_l^{st} - \sum_{l:l\in \text{Out}(v)} x_l^{st} \right) = M_t \cdot I_{v=t}, \ \forall v \notin S, \ \forall t \in T \tag{2.11e}$$

$$\text{variables} \quad x_l^{st} \ge 0, \ 0 \le r_l^{ij} \le 1 \tag{2.11f}$$

Denote the feasible space of the joint variable as $\mathcal{A} = \{\hat{X}_{cp}, R_{bg}\}$. If we vary γ and plot the achieved TE objectives versus SS objectives, we obtain the Pareto curve.

To illustrate the Pareto curve and efficiency loss in Model I and Model II, we plot in Figure 2.3 the Pareto curve and the Nash equilibria in the two-dimensional objective space (TE,SS) for the network shown in Figure 2.2. The simulation shows that when the CP leverages the complete information to optimize (2.3a), it is able to achieve lower delay, but the TE cost suffers. Though it is not clear which operating point is better, both equilibria are away from the Pareto curve, which shows that there is room for performance improvement in both dimensions.

2.6 A Joint Design: Model III

Motivated by the need for a joint TE and SS design, we propose the Nash bargaining solution to reduce the efficiency loss observed above. Using the theory of optimization decomposition, we derive a distributed algorithm by which the ISP and the CP can act separately and communicate with a limited amount of information exchange.

2.6.1 Motivation

An ISP providing content distribution service in its own network has control over both routing and server selection. So the ISP can consider the characteristics of both types of traffic (background and CP) and jointly optimize a carefully chosen objective. The jointly optimized system should meet at least two goals: (i) optimality, *i.e.*, it should achieve Pareto optimality so the network resources are efficiently utilized, and (ii) fairness, *i.e.*, the tradeoff between two non-synergistic objectives should be balanced so both parties benefit from the cooperation.

One natural design choice is to optimize the weighted sum of the traffic engineering goal and server selection goal as shown in (2.11). However, solving (2.11) for each γ and adaptively tuning γ in a *trial-and-error* fashion is impractical and inefficient. First, it is not straightforward to weigh the tradeoff between the two objectives. Second, one needs to compute an appropriate weight parameter γ for every combination of background load and CP traffic demand. In addition, the offline computation does not adapt to dynamic changes of network conditions, such as cross traffic or link failures. Last, tuning γ to explore a broad region of system operating points is computationally expensive.

Besides the system considerations above, the economic perspective requires a *fair* solution. Namely, the joint design should benefit both TE and SS. In addition, such a model also applies to a more general case when the ISP and the CP are different business entities. They cooperate only when the cooperation leads to a win-win situation, and the "division" of the benefits should be fair, *i.e.*, one who makes greater contribution to the collaboration should be able to receive more reward, even when their goals are conflicting.

While the joint system is designed from a clean state, it should accept an incremental deployment from the existing infrastructure. In particular, we prefer that the functionalities of routing and server selection be separated, with minor changes to each component. The modularized design allows us to manage each optimization independently, with a judicious amount of information exchange. Designing for scalability and modularity is beneficial to both the ISP and the CP, and allows their cooperation either as a single entity or as different ones.

Based on all the above considerations, we apply the concept of *Nash bargaining solution* [21, 33] from cooperative game theory. It ensures that the joint system achieves an *efficient* and *fair* operating point. The solution structure also allows a modular implementation.

2.6.2 Nash Bargaining Solution

Consider a Nash bargaining solution which solves the following optimization problem:

NBS

$$\begin{aligned} \text{maximize} \quad & (TE_0 - TE)(SS_0 - SS) && \text{(2.12a)} \\ \text{variables} \quad & \{\hat{X}_{cp}, R_{bg}\} \in \mathcal{A} && \text{(2.12b)} \end{aligned}$$

where (TE_0, SS_0) is a constant called the *disagreement point*, which represents the starting point of their negotiation. Namely, (TE_0, SS_0) is the status-quo we observe before any cooperation. For instance, one can view the Nash equilibrium in Model I as a disagreement point, since it is the operating point the system would reach without any further optimization. By optimizing the product of performance improvements of TE and SS, the Nash bargaining solution guarantees the joint system is optimal and fair. A Nash bargaining solution is defined by the following axioms, and is the only solution that satisfies all of four axioms [21, 33]:

- *Pareto optimality.* A Pareto optimal solution ensures efficiency.

- *Symmetry.* The two players should get equal share of the gains through cooperation, if the two players' problems are symmetric, *i.e.*, they have the same cost functions, and have the same objective value at the disagreement point.

- *Expected utility axiom.* The Nash bargaining solution is invariant under affine transformations. Intuitively, this axiom suggests that the Nash bargaining solution is insensitive to different units used in the objective and can be efficiently computed by affine projection.

- *Independence of irrelevant alternatives.* This means that adding extra constraints in the feasible operating region does not change the solution, as long as the solution itself is feasible.

The choice of the disagreement point is subject to different economic considerations. For a single network provider who wishes to provide both services, it can optimize the product of improvement ratio by setting the disagreement point to be the origin, *i.e.*, equivalent to $TE{\cdot}SS/(TE_0{\cdot}SS_0)$. For two separate ISP and CP entities who wish to cooperate, the Nash equilibrium of Model I may be a natural choice, since it represents the benchmark performance of current practice, which is the baseline for any future cooperation. It can be obtained from the empirical observations of their average performance. Alternatively, they can choose their preferred performance level as the disagreement point, written into the contract. In this work, we use the Nash equilibrium of Model I as the disagreement point to compare the performances of our three models.

2.6.3 COTASK Algorithm

In this section, we show how Nash bargaining solution can be implemented in a modularized manner, *i.e.*, keeping SS and TE functionalities separate. This is important because modularized design increases the re-usability of legacy systems with minor changes, like existing CDNs deployment. In terms of cooperation between two independent financial entities, the modularized structure presents the possibility of cooperation without revealing confidential internal information to each other.

We next develop COTASK (COoperative TrAffic engineering and Server selection inside an ISP networK), a protocol that implements NBS by separate TE and SS optimizations and communication between them. We apply the theory of optimization decomposition [22] to decompose problem (2.12) into subproblems. ISP solves a new routing problem, which controls the routing of background traffic only. The CP solves a new server selection problem, given the network topology information. The ISP also passes the routing control of content traffic to the CP, offering more freedom to how content can be delivered on the network. They communicate via underlying *link prices*, which are computed locally using traffic levels on each link.

Consider the objective (2.12a), which can be converted into

$$\text{maximize} \quad \log(TE_0 - TE) + \log(SS_0 - SS)$$

since the log function is monotonic and the feasible solution space is unaffected. The introduction of the log functions help reveal the decomposition structure of the original problem. Two auxiliary variable $\overline{f_l^{cp}}$ and $\overline{f_l^{bg}}$ are introduced to reflect the preferred CP traffic level from the ISP's perspective and the preferred background traffic level from the CP's perspective. (2.12) can be rewritten as

$$\begin{aligned}
\text{maximize} \quad & \log(TE_0 - \sum_l g_l(f_l^{bg} + \overline{f_l^{cp}})) + \log(SS_0 - \sum_l h_l(f_l^{cp} + \overline{f_l^{bg}})) && \text{(2.13a)}\\
\text{subject to} \quad & f_l^{cp} = \sum_{(s,t)} x_l^{st},\ f_l^{bg} = \sum_{(i,j)\notin S\times T} x_{ij}\cdot r_l^{ij},\ \forall l && \text{(2.13b)}\\
& \overline{f_l^{cp}} = f_l^{cp},\ \overline{f_l^{bg}} = f_l^{bg},\ f_l^{cp} + f_l^{bg} \le C_l,\ \forall l && \text{(2.13c)}\\
& \sum_{l:l\in \text{In}(v)} r_l^{ij} - \sum_{l:l\in \text{Out}(v)} r_l^{ij} = I_{v=j},\ \forall (i,j)\notin S\times T, \forall v \in V\backslash\{i\} && \text{(2.13d)}\\
& \sum_{s\in S}\left(\sum_{l:l\in \text{In}(v)} x_l^{st} - \sum_{l:l\in \text{Out}(v)} x_l^{st}\right) = M_t \cdot I_{v=t},\ \forall v\notin S,\ \forall t\in T && \text{(2.13e)}\\
\text{variables} \quad & x_l^{st} \ge 0,\ 0 \le r_l^{ij} \le 1,\ \forall (i,j)\notin S\times T,\ \overline{f_l^{cp}},\ \overline{f_l^{bg}} && \text{(2.13f)}
\end{aligned}$$

The consistency constraint on the auxiliary variable and the original variable ensures that the solution equivalent to problem (2.12). We take the partial Lagrangian of (2.13) as

$$\begin{aligned}
\mathcal{L}(x_l^{st}, r_l^{ij}, \overline{f_l^{cp}}, \overline{f_l^{bg}}, \lambda_l, \mu_l, \nu_l) \quad = \quad & \log(TE_0 - \sum_l g_l(f_l^{bg} + \overline{f_l^{cp}})) + \log(SS_0 - \sum_l h_l(f_l^{cp} + \overline{f_l^{bg}}))\\
& + \sum_l \mu_l(f_l^{bg} - \overline{f_l^{bg}}) + \sum_l \nu_l(f_l^{cp} - \overline{f_l^{cp}}) + \sum_l \lambda_l(C_l - f_l^{bg} - f_l^{cp})
\end{aligned}$$

λ_l is the *link price*, which reflects the cost of overshooting the link capacity, and μ_l, ν_l are the *consistency prices*, which reflect the cost of disagreement between ISP and CP on the preferred link resource allocation. Observe that f_l^{cp} and f_l^{bg} can be separated in the Lagrangian function. We take a dual decomposition approach, and (2.13) is decomposed into two subproblems:

SS – NBS

$$\begin{aligned}
\text{maximize} \quad & \log(SS_0 - \sum_l h_l(f_l^{cp} + \overline{f_l^{bg}})) + \sum_l (\nu_l f_l^{cp} - \mu_l \overline{f_l^{bg}} - \lambda_l f_l^{cp}) && (2.14a)\\
\text{subject to} \quad & f_l^{cp} = \sum_{(s,t)} x_l^{st}, \ \forall l && (2.14b)\\
& \sum_{s \in S} \left(\sum_{l: l \in \text{In}(v)} x_l^{st} - \sum_{l: l \in \text{Out}(v)} x_l^{st} \right) = M_t \cdot I_{v=t}, \ \forall v \notin S, \ \forall t \in T && (2.14c)\\
\text{variables} \quad & x_l^{st} \geq 0, \forall (s,t) \in S \times T, \ \overline{f_l^{bg}} && (2.14d)
\end{aligned}$$

and

TE – NBS

$$\begin{aligned}
\text{maximize} \quad & \log(TE_0 - \sum_l g_l(f_l^{bg} + \overline{f_l^{cp}})) + \sum_l (\mu_l f_l^{bg} - \nu_l \overline{f_l^{cp}} - \lambda_l f_l^{bg}) && (2.15a)\\
\text{subject to} \quad & f_l^{bg} = \sum_{(i,j) \notin S \times T} x_{ij} \cdot r_l^{ij}, \ \forall l && (2.15b)\\
& \sum_{l: l \in \text{In}(v)} r_l^{ij} - \sum_{l: l \in \text{Out}(v)} r_l^{ij} = I_{v=j}, \ \forall (i,j) \notin S \times T, \forall v \in V \backslash \{i\} && (2.15c)\\
\text{variables} \quad & 0 \leq r_l^{ij} \leq 1, \forall (i,j) \notin S \times T, \ \overline{f_l^{cp}} && (2.15d)
\end{aligned}$$

The optimal solutions of (2.14) and (2.15) for a given set of prices μ_l, ν_l, and λ_l define the dual function $\text{Dual}(\mu_l, \nu_l, \lambda_l)$. The dual problem is given as:

$$\begin{aligned}
\text{minimize} \quad & \text{Dual}(\mu_l, \nu_l, \lambda_l) && (2.16a)\\
\text{variables} \quad & \lambda_l \geq 0, \mu_l, \nu_l && (2.16b)
\end{aligned}$$

```
COTASK Algorithm

        ISP: TE algorithm
(i)     Receives link price $\lambda_l$ and consistency price $\mu_l, \nu_l$ from
        physical links $l \in E$
(ii)    ISP solves (2.15a) and computes $R_{bg}$ for background traffic
(iii)   ISP passes $f_l^{bg}$, $\overline{f_l^{cp}}$ information to each link $l$
(iv)    Go back to (i)

        CP: SS algorithm
(i)     Receives link price $\lambda_l$ and consistency price $\mu_l, \nu_l$ from
        physical links $l \in E$
(ii)    CP solves (2.14a) and computes $X_{cp}$ for content traffic.
(iii)   CP passes $f_l^{cp}$, $\overline{f_l^{bg}}$ information to each link $l$
(iv)    Go back to (i)

        Link: price update algorithm
(i)     Initialization step: set $\lambda_l \geq 0$, and $\mu_l, \nu_l$ arbitrarily
(ii)    Updates link price $\lambda_l$ according to (2.17)
(iii)   Updates consistency prices $\mu_l, \nu_l$ according to (2.18)(2.19)
(iv)    Passes $\lambda_l, \mu_l, \nu_l$ information to TE and SS
(v)     Go back to (ii)
```

Table 2.4: Distributed algorithm for solving problem (2.12a).

We can solve the dual problem with the following price updates:

$$\lambda_l(t+1) = \left[\lambda_l(t) - \beta_{\lambda l}\left(C_l - f_l^{bg} - f_l^{cp}\right)\right]^+, \ \forall l \tag{2.17}$$

$$\mu_l(t+1) = \mu_l(t) - \beta_{\mu l}\left(f_l^{bg} - \overline{f_l^{bg}}\right), \ \forall l \tag{2.18}$$

$$\nu_l(t+1) = \nu_l(t) - \beta_{\nu l}\left(f_l^{cp} - \overline{f_l^{cp}}\right), \ \forall l \tag{2.19}$$

where β's are diminishing step sizes or small constant step sizes often used in practice [34]. Table 2.4 presents the COTASK algorithm that implements the Nash bargaining solution distributively.

In the COTASK algorithm, the ISP solves the new version TE, *i.e.*, TE-NBS, and the CP solves the new version SS, *i.e.*, SS-NBS. In terms of information sharing, the CP learns the network topology from the ISP. They do not directly exchange information with each other. Instead, they report $\overline{f_l^{cp}}$ and $\overline{f_l^{bg}}$ information to underlying links, which pass the computed price information back to TE and SS. It is possible to further implement TE or SS in a distributed manner, such as on the user/server levels.

There are two main challenges on practical implementation of COTASK. First, TE needs to adapt quickly to network dynamics. Fast timescale TE has recently been proposed in various works. Second, an extra price update component is required on each link, which involves price computation and message passing between TE and SS. This functionality can be potentially implemented in routers.

Theorem 7. *The distributed algorithm COTASK converges to the optimum of (2.12)*

Proof. The COTASK algorithm is precisely captured by the decomposition method described above. Certain choice of step sizes, such as $\beta(t) = \beta_0/t$, where $\beta_0 > 0$, guarantees that the algorithm converges to a global optimum [35]. □

2.7 Performance Evaluation

In this section, we use simulations to demonstrate the efficiency loss that may occur for real network topologies and traffic models. We also compare the performance of the three models. We solve the Nash bargaining solution centrally, without using the COTASK algorithm, since we are primarily interested in its performance. Complementary to the theoretical analysis, the simulation results allow us to gain a better understanding of the efficiency loss under realistic network environments. These simulation results also provide guidance to network operators who need to decide which approach to take, sharing information or sharing control.

2.7.1 Simulation Setup

We evaluate our models under ISP topologies obtained from Rocketfuel [36]. We use the backbone topology of the research network Abilene [37] and several major tier-1 ISPs in north America. The choice of these topologies also reflects different geometric properties of the graph. For instance, Abilene is the simplest graph with two bottleneck paths horizontally. The backbones of AT&T and Exodus have a hub-and-spoke structure with some shortcuts between nodes pairs. The topology of Level 3 is almost a complete mesh, while Sprint is in between these two kinds. We simulate the traffic demand using a gravity model [38], which reflects the pairwise communication pattern on the Internet. The content demand of a CP user is assumed to be proportional to the node population.

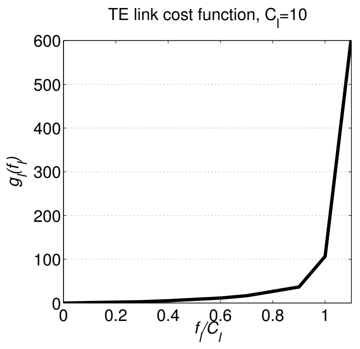
The TE cost function $g(\cdot)$ and the SS cost function $h(\cdot)$ are chosen as follows. ISPs usually model congestion cost with a convex increasing function of the link load. The exact shape of the function $g_l(f_l)$ is not important, and we use the same piecewise linear cost function as in [23], given below:

$$g_l(f_l, C_l) = \begin{cases} f_l & 0 \le f_l/C_l < 1/3 \\ 3f_l - 2/3C_l & 1/3 \le f_l/C_l < 2/3 \\ 10f_l - 16/3C_l & 2/3 \le f_l/C_l < 9/10 \\ 70f_l - 178/3C_l & 9/10 \le f_l/C_l < 1 \\ 500f_l - 1468/3C_l & 1 \le f_l/C_l < 11/10 \\ 5000f_l - 16318/3C_l & 11/10 \le f_l/C_l < \infty \end{cases}$$

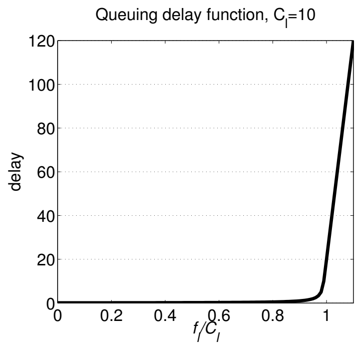
The CP's cost function can be the performance cost like latency, financial cost charged by ISPs. We consider the case where latency is the primary performance metric, *i.e.*, the content traffic is delay sensitive like video conferencing or live streaming. So we let the CP's cost function $h_l(\cdot)$ be of the form given by (2.2), *i.e.*, $h_l(f_l) = f_l^{cp} \cdot D_l(f_l)$. A link's latency $D_l(\cdot)$ consists of queuing delay and propagation delay. The propagation delay is proportional to the geographical distances between nodes. The queuing delay is approximated by the M/M/1 model, *i.e.*,

$$D_{queue} = \frac{1}{C_l - f_l}, \quad f_l < C_l$$

with a linear approximation when the link utilization is over 99%. We relax hard capacity constraints by penalizing traffic overshooting the link with a high cost, for consistency throughput this work. The shapes of the TE link cost function and queuing delay function are illustrated in Figure 2.4. We intensionally choose the cost functions of TE and SS to be similar in shape. This allows us to quantify the efficiency loss of Model I and Model II even when their objectives are relatively well aligned, as well as the improvement brought by Model III.

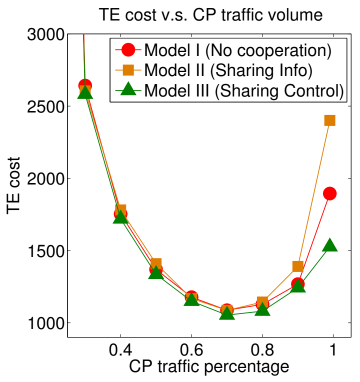


(a) TE link cost

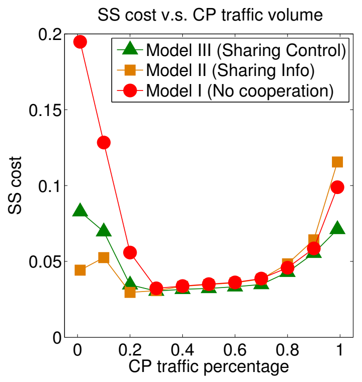


(b) Link queuing delay

Figure 2.4: ISP and CP cost functions.



(a)



(b)

Figure 2.5: The TE-SS tussle v.s. CP's traffic intensity (Abilene topology)

2.7.2 Evaluation Results

Tussle between background and CP's traffic

We first demonstrate how CP's traffic intensity affects the overall network performance. We fix the total amount of traffic and tune the ratio between background traffic and CP's traffic. We evaluate the performance of different models when CP traffic grows from 1% to 100% of the total traffic. Figure 2.5 illustrates the results on Abilene topology.

The general trend of both TE and SS objectives for all three models is that the cost first decreases as CP traffic percentage grows, and later increases as CP's traffic dominates the network.

The decreasing trend is due to the fact that CP's traffic is self-optimized by selecting servers close to a user, thus offloading the network. The increasing trend is more interesting, suggesting that when a higher percentage of total traffic is CP-generated, the negative effect of TE-SS interaction is amplified, even when the ISP and the CP share similar cost functions. Low link congestion usually means low end-to-end latency, and vice versa. However, they differ in the following: (i) TE might penalize high utilization before queueing delay becomes significant in order to leave as much room as possible to accommodate changes in traffic, and (ii) CP considers both propagation delay and queueing delay so it may choose a moderately-congested short path over a lightly-loaded long path. This explains why the optimization efforts of two players are at odds.

Network congestion v.s. performance improvement

We now study the network conditions under which more performance improvement is possible. We evaluate the three models on the Abilene topology. Again, we fix the total amount of traffic and vary the CP's traffic percentage. Now we change link capacities and evaluate two scenarios: when the network is moderately congested and when the network is highly congested. We show the performance improvement of Model II and Model III over Model I (in percentages) and plot the results in Figure 2.6. Figures 2.6(a-b) show the improvement of the ISP and the CP when the network is under low load. Generally, Model II and Model III improve both TE and SS, and Model III outperforms Model II in most cases, with the exception that Model II is biased towards SS sometimes. However, both ISP and CP's improvement are not substantial (note the different scales of y-axes), except when CP traffic is trivial (1%). This is because when the network is under low load, the slopes of TE and SS cost functions are "flat," thus leaving little space for improvement.

Figure 2.6(c-d) show the results when the network is under a high load. The performance improvement becomes more significant, especially at two extremes: when CP traffic is trivial or prevalent. This suggests that when CP traffic is dominant, there is a large room for improvement when two objectives are similar in shape. However, observe that while model III always improves TE and SS, Model II could sometimes perform worse than Model I. This indicates that there are more inferior Nash equilibria, when a larger fraction of CP traffic exists.

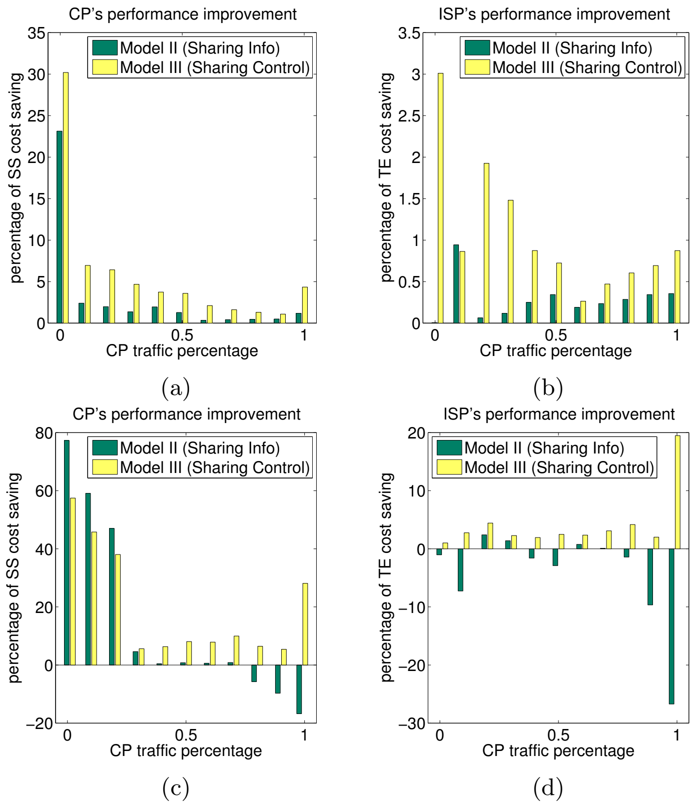
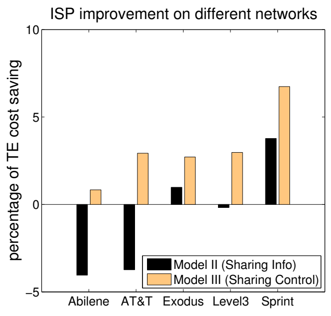


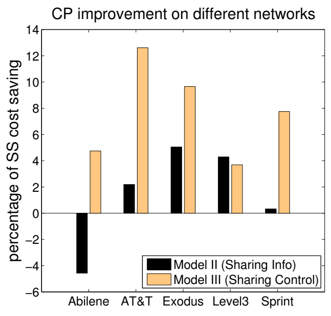
Figure 2.6: TE and SS performance improvement of Model II and III over Model I. (a-b) Abilene network under low traffic load: moderate improvement; (c-d) Abilene network under high traffic load: more significant improvement, but more information (in Model II) does not necessarily benefit the CP and the ISP (the paradox of extra information).

Impact of ISP topologies

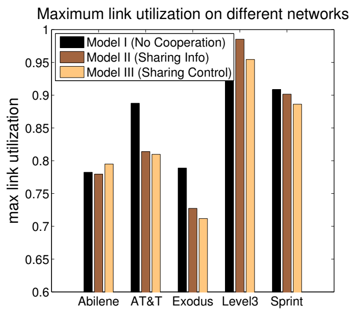
We evaluate our three models on different ISP topologies. The topological properties of different graphs are discussed earlier. The CP's traffic is 80% of the total traffic and link capacities are set such that networks are under high traffic load. Our findings are depicted in Figure 2.7. Note that performance improvement is relatively more significant in more complex graphs. Simple topologies with small min-cut sizes are networks where the apparent paradox of more (incomplete) information is likely to happen. Besides the TE and SS objectives, we also plot the maximum link utilization to illustrate the level of congestion in the network. Higher network load shows more space for potential improvement. Also, model III improves this metric generally, which might be another important consideration for network providers.



(a)



(b)



(c)

Figure 2.7: Performance evaluation over different ISP topologies. Abilene: small cut graph; AT&T, Exodus: hub-and-spoke with shortcuts; Level 3: complete mesh; Sprint: in between.

2.8 Related Work

This work is an extension of our earlier workshop paper [39]. Additions in this paper include the following: a more general CP model, analysis of optimality conditions in three cooperation models, paradox of extra information, implementation of Nash bargaining solution, and large scale evaluation.

The most similar work is a parallel work [20], which studied the interaction between content distribution and traffic engineering. The authors show the optimality conditions for two separate problems to converge to a socially optimal point, as discussed in Section 2.4.2. They provide a 4/3-bound on efficiency loss for linear cost functions, and discuss generalizations to multiple ISPs and overlay networks.

Some earlier work studied the self-interaction within ISPs or CPs themselves. In [28], the authors used simulation to show that selfish routing is close to optimal in Internet-like environments without sacrificing much performance degradation. [40] studied the problem of load balancing by overlay routing, and how to alleviate race conditions among multiple co-existing overlays. [41] studied the resource allocation problem at inter-AS level where ISPs compete to maximize their revenues. [42] applied Nash bargaining solution to solve an inter-domain ISP peering problem.

The need for cooperation between content providers and network providers is raising much discussion in both the research community and industry. [43] used price theory to reconcile the tussle between peer-assisted content distribution and ISP's resource management. [19] proposed

	CP no change	**CP change**
ISP no change	current practice	partial collaboration
ISP change	partial collaboration	joint system design

Table 2.5: To cooperate or not: possible strategies for content provider (CP) and network provider (ISP)

a communication portal between ISPs and P2P applications, which P2P applications can consult for ISP-biased network information to reduce network providers' cost without sacrificing their performances. [18] proposed an oracle service run by the ISP, so P2P users can query for the ranked neighbor list according to certain performance metrics. [44] utilized existing network views collected from content distribution networks to drive biased peer selection in BitTorrent, so cross-ISP traffic can be significantly reduced and download-rate improved.

[45] studied the interaction between underlay routing and overlay routing, which can be thought of as a generalization of server selection. The authors studied the equilibrium behaviors when two problems have conflicting goals. Our work explores when and why sub-optimality appears, and proposes a cooperative solution to address these issues. [46] studied the economic aspects of traditional transit providers and content providers, and applied cooperative game theory to derive an optimal settlement between these entities.

2.9 Summary

We examine the interplay between traffic engineering and content distribution. While the problem has long existed, the dramatically increased amount of content-centric traffic, *e.g.*, CDN and P2P traffic, makes it more significant. With the strong motivation for ISPs to provide content services, they are faced with the question of whether to stay with the current design or to start sharing information or control. This work sheds light on ways ISPs and CPs can cooperate.

This work serves as a starting point to better understand the interaction between those that operate networks and those that distribute content. Traditionally, ISPs provide and operate the pipes, while content providers distribute content over the pipes. In terms of what information can be shared between ISPs and CPs and what control can be jointly performed, there are four general categories as summarized in Table 2.5. The top left corner is the current practice, which may give an undesirable Nash equilibrium. The bottom right corner is the joint design, which achieves optimal operation points. The top right corner is the case where the CP receives extra information and adapts control accordingly, and the bottom left corner is the case of content-aware networking. This work studies three of the four corners in the table. Starting from the current practice, to move towards the bottom right corner of the table, while the two parties remain separate business entities, requires unilaterally-actionable, backward-compatible, and incrementally-deployable migration paths yet to be discovered.

Chapter 3

Federating Content Distribution across Decentralized CDNs

This chapter focuses on the challenge of *joint control* over multiple traffic management decisions that are operated by multiple institutions at different time-scales [47]. We consider a content delivery architecture based on geographically-distributed groups of "last-mile" CDN servers, *e.g.*, set-top boxes located within users' homes. In contrast to Chapter 2, these servers may belong to administratively separate domains, *e.g.*, multiple ISPs or CDNs. We propose a set of mechanisms to jointly manage content replication and request routing within this architecture, achieving both scalability and cost optimality. Specifically, our solution consists of two parts. First, we identify the replication and routing variables for each group of servers, based on distributed message-passing between these groups. Second, we describe algorithms for content placement and request mapping at the server granularity within each group, based on the decisions made in the first step. We formally prove the optimality of these methods, and confirm their efficacy through evaluations based on BitTorrent traces. In particular we observe a reduction of network costs by more than 50% over state-of-the art mechanisms.

3.1 Introduction

The total Internet traffic per month in 2011 is already in excess of 10^{19} Bytes [48]. Video-on-demand traffic alone is predicted to grow to three times this amount by 2015 [48]. This foreseen growth prompts a rethinking of the current content delivery architecture. Today's content delivery networks (CDNs) operate in isolation, hence missing the opportunity to pool resources of individual CDNs. Recognizing the potential of federating CDNs, industry stakeholders have created the IETF CDNi working group [49] to standardize protocols for CDN interoperability. Another promising evolution consists of extending the CDN to the "last mile" of content delivery by incorporating servers at the network periphery. This approach leverages small servers within users' homes, such as set-top boxes or broadband gateways, as advocated by the Nano-Datacenter consortium [50], or dedicated toaster-sized appliances promoted by business initiatives [51]. By inter-connecting these diffuse clouds, user requests directed to one operator may be forwarded to another for the purpose of availability and proximity.

Operating a federation of decentralized CDNs requires efficient traffic management among a collection of distinct service providers. Traffic crossing the provider boundaries may experience degraded performance such as extra latency. Within each provider, traffic exchange between the "last-mile" servers located at different ISPs also implies increased billing costs. Our work aims at developing solutions that minimize cross-traffic costs and accommodate user demands in a scalable manner.

Distributing content among a collection of operators presents an optimization problem over several degrees of freedom: (i) content replication within each operator, (ii) request mapping to different operators, and (iii) service assignment to individual servers. Traditional design addresses these problems separately and yet they clearly impact one another. In this work, we present a set of solutions that collectively achieve all of the following:

- A joint optimization over both content placement and routing with provably-optimal performance.
- A decentralized implementation that facilitates coordination between different administrations.
- A scalable service assignment scheme that is easy to implement.

- An adaptive content caching algorithm with low operational costs.

This chapter proceeds along the following steps. We first describe an optimization problem featuring both placement and routing variables, whose solution gives a lower bound on the best achievable costs (Section 3.2). We then propose a scheme distributed between the decentralized CDN operators which identifies content replication and routing policies at the operator level (Section 3.3). We next develop content management and request routing strategies at the server-level within each operator (Sections 3.4 and 3.5). In conjunction, the operator- and server-level schemes are proven to achieve optimal network costs (Theorems 1,2 and 3). At a methodological level, these results rely on decomposition of optimization problems coupled with primal-dual techniques, and Lyapunov stability applied to fluid limits. In addition to this theoretical underpinning, our solution is further validated experimentally in Section 3.6. Simulations driven by BitTorrent traces, with all the associated features of real traffic (bursty, long-tail distribution, geographical heterogeneities) show that our approach reduces by more than half network costs, as compared to state-of-the-art solutions based on LRU cache management and nearest-neighbor routing. We present related work in Section 3.7 and conclude in Section 3.8.

3.2 Problem Formulation and Solution Structure

In this section, we introduce our system model and a global optimization problem that the content provider solves. We also propose a content distribution architecture that allows a scalable, efficient, and decentralized solution to the global problem. The key notations are summarized in Table 3.1.

3.2.1 System Model

The system consists of a set $\mathcal{B}$ of *boxes* where $B = |\mathcal{B}|$, and a distinguished node s, the *content server*. The content server s is owned and operated by a content provider, such as YouTube or Netflix, who wishes to deliver content (*e.g.*, videos or songs) to home users who subscribe its services. The boxes, such as set-top boxes or network-attached storage (NAS), are installed at users' homes, providing common Internet connectivity and limited storage. The collection of delivered content constitutes a set $\mathcal{C}$ where $C = |\mathcal{C}|$, called the *content catalog*. All content are

Notation	**Interpretation**
s	Content server.
$\mathcal{B}$	Set of all boxes in the system.
$\mathcal{C}$	Content catalog.
$\mathcal{D}$	Set of all set-of-box classes.
$\mathcal{B}^d$	Boxes in class $d \in \mathcal{D}$.
M^d	Storage capacity of boxes in $\mathcal{B}^d$.
U^d	Upload capacity of boxes in $\mathcal{B}^d$.
$\mathcal{F}_b$	Cache content of box b.
p_c^d	Replication ratio of content c in $\mathcal{B}^d$.
λ_c^d	Request rate for content $c \in \mathcal{C}$ in $\mathcal{B}^d$.
$w^{dd'}$	Cost of transfering content from $d' \in \mathcal{D} \cup \{s\}$ to $d \in \mathcal{D}$.
$r_c^{dd'}$	Rate of requests for c routed from $d \in \mathcal{D}$ to $d' \in \mathcal{D} \cup \{s\}$.
$r_c^{\cdot d}$	Aggregate rate of incoming requests for c to $d \in \mathcal{D} \cup \{s\}$.
R^d	$R^d = B^d U^d$, the total upload capacity in class d.

Table 3.1: Summary of key notations

replicated at server s. The storage and upload capacities of boxes are leased out to the content provider, which uses these resources to off-load part of the traffic load on the server s.

Box Classes

A content provider's service cover geographically-diverse regions and different ISPs. The customers present a diversity of their Internet connectivity (*e.g.*, bandwidth) and even box storage capacity. We partition the set $\mathcal{B}$ of boxes into D classes $\mathcal{B}^d$ of size $B^d = |\mathcal{B}^d|$, where $d \in \mathcal{D} = \{1, \ldots, D\}$. Such partitioning may correspond, *e.g.*, to grouping together boxes managed by the same ISP. Different levels of aggregation or granularity can also be used to refine the geographic diversity. For example, each class may comprise boxes within the same city or even the same city block. We allow classes to be heterogeneous, *i.e.*, the *storage* and *bandwidth capacities* of boxes may differ across classes. We denote by M^d the storage capacity of boxes in $\mathcal{B}^d$, *e.g.*, the number of items a box can store. We make the assumption that content are of an identical size—for instance, the original content are chopped into chunks of a fixed size and the catalog is viewed as a collection of chunks rather than the original items.

For each box $b \in \mathcal{B}^d$, let

$$\mathcal{F}_b \subset \mathcal{C}, \text{where } |\mathcal{F}_b| = M^d$$

be the set of content cached in box b. For each class d, let

$$p_c^d = \frac{\sum_{b \in \mathcal{B}^d} \mathbb{1}_{c \in \mathcal{F}_b}}{B^d}, \quad \forall c \in \mathcal{C} \tag{3.1}$$

be the fraction of boxes in $\mathcal{B}^d$ that store content $c \in \mathcal{C}$. We call p_c^d the *replication ratio* of item c in class d. As the total storage capacity of class d is $B^d M^d$, it is easy to see that, when all caches are full, the replication ratios satisfy

$$\sum_{c \in \mathcal{C}} p_c^d = M^d, \quad \forall d \in \mathcal{D}. \tag{3.2}$$

For a fixed set of replication ratio $\{p_c^d\}$, there are many combinations of the exact content placement profiles $\{\mathcal{F}_b\}_{b \in \mathcal{B}^d}$. Therefore, the replication ratio can be viewed as a class-wide description of content placement decisions.

We denote by U^d the upload capacity of boxes in $\mathcal{B}^d$. A box can upload at most U^d content items concurrently, each at a fixed rate. Alternatively, each box has U^d upload "slots": once a box receives a request for a content it stores, a free upload slot is taken to serve the request and upload the requested content, if it exists. For example, a box has $U^d = 5$Mbps dedicated upload capacity, and is able to serve at most 5 concurrent requests, *e.g.*, video streaming, each at 1Mbps rate. The service time, *e.g.*, the duration of a streaming session, is assumed to be exponentially distributed with one unit mean. Slots remain busy until the upload terminates, at which point they become free again. When all U^d slots of a box are busy, it is unable to serve any additional requests.

As such, we use a *loss model* [52], a key feature of our design, rather than a queueing model, to capture the service behavior of the system. Such a choice is based on several reasons. First, an incoming request must be immediately served by a box, or re-routed to the server otherwise, rather than waiting in the queue. Second, most of today's content services, such as video streaming, require a constant bit-rate and do not consume the extra bandwidth. Third, we primarily focus on a heavy traffic scenario in which a box's upload bandwidth is rarely under-utilized, as it is to the content provider's interests to offload traffic from the server as much as possible.

Request Load

Users (boxes) generate content requests at varying rates across different classes. In particular, each $b \in \mathcal{B}^d$ generates requests for content c according to a Poisson process with rate $\tilde{\lambda}_c^d$. The aggregate request rate for c in class d is $\lambda_c^d = \tilde{\lambda}_c^d B^d$, which scales proportionally to the class size. When a box $b \in \mathcal{B}^d$ storing $c \in \mathcal{C}$ (*i.e.*, $c \in \mathcal{F}_b$) generates a request for c, it is served by the local cache—no downloading is necessary. Otherwise, the request must be served by either the content server s or some other box, in $\mathcal{B}^d$ or in a different class.

We denote by $r_c^{dd'}$ the aggregate request rate routed from class d boxes to class d' boxes, and by r_c^{ds} the request rate routed *directly* to the server s. To meet users' demands, these rates must satisfy

$$r_c^{ds} + \sum_{d' \in \mathcal{D}} r_c^{dd'} = \lambda_c^d (1 - p_c^d), \quad \forall d \in \mathcal{D}, \tag{3.3}$$

i.e., requests not immediately served by local caches in class d are served by server s or a box in some class $d' \in \mathcal{D}$.

Loss Probabilities

Not all requests for content c that arrive at a class d can be served by boxes in d. For example, it is possible that no free upload slots in the class exist when the request for c arrives. In such a case, we assume that a request has to be "dropped" from class d and re-routed to the server s.

An important performance metric that a content provider cares about is the request loss probability, as it wishes to offload traffic from the server. Let ν_c^d be the *loss probability* of item c in class d, *i.e.*, the steady state probability that a request for a content item c is dropped upon its arrival and needs be re-routed to the server s. In general, ν_c^d depends on the following three factors: (a) the arrival rates $\{r_c^{\cdot d}\}_{c \in \mathcal{C}}$ of requests for different content, where $r_c^{\cdot d} = \sum_{d' \in \mathcal{D}} r_c^{d'd}$ is the aggregate request rate for content c received by class d, (b) the content placement profile $\{\mathcal{F}_b\}_{b \in \mathcal{B}^d}$ in class d boxes, and (c) the service assignment algorithm that maps incoming requests to boxes that serve them.

We say that the requests for item c are served *with high probability* (w.h.p.) in class d, if

$$\lim_{B^d \to \infty} \nu_c^d(B^d) = 0, \tag{3.4}$$

i.e., as the total number of boxes increases, the probability that a request for content c is dropped goes to zero. Two necessary constraints for (3.4) to hold for $d \in \mathcal{D}$ are:

$$\sum_{c \in \mathcal{C}} r_c^{\cdot d} < B^d U^d, \qquad \forall d \in \mathcal{D} \tag{3.5}$$

$$r_c^{\cdot d} < B^d U^d p_c^d, \qquad \forall c \in \mathcal{C}, d \in \mathcal{D}. \tag{3.6}$$

Constraint (3.5) states that the aggregate traffic load imposed on class d should not exceed the total upload capacity over all boxes. In addition, (3.6) states that the traffic imposed on d by requests for c should not exceed the total capacity of boxes storing c.

In Sections 3.4 and 3.5, we will show that (3.5) and (3.6) are also *sufficient* for (3.4), by presenting (a) a service assignment algorithm that map incoming requests to boxes, and (b) an algorithm for placing the content $\{\mathcal{F}_b\}_{b \in \mathcal{B}^d}$, such that requests for all content $c \in \mathcal{C}$ are served w.h.p. given that (3.5) and (3.6) hold.

3.2.2 A Global Optimization for Minimizing Costs

We next introduce a global optimization problem that allows the content provider to minimize its operational costs by reducing the cross traffic, while ensuring close-to-zero service loss probabilities.

Minimizing Cross-Traffic Costs

Serving a user request from one class d in another class d' requires transferring content across the class boundaries. The cross-traffic presents a significant operational cost to the content provider. For example, the content provider needs to pay the bandwidth cost at a fixed rate for its outgoing traffic [3]. In particular, we group boxes by ISPs, and the cross-traffic costs are dictated by the transit agreements between peering ISPs that may vary from one to another. As such, we denote $w^{dd'}$ as the unit bandwidth cost for routing traffic from ISP d to d'. As it is the content provider's goal to offload traffic from the central server s, we also introduce a cost w^{ds} that represents the unit traffic cost of serving class d requests by the server, such that $w^{ds} > w^{dd'}$ for any d'.

The total weighted cross-traffic costs in the system can be formulated as

$$\sum_{c\in\mathcal{C}}\sum_{d\in\mathcal{D}}\left(w^{ds}r_c^{ds}+\sum_{d'}\left(w^{dd'}r_c^{dd'}(1-\nu_c^{d'})+w^{ds}r_c^{dd'}\nu_c^{d'}\right)\right),$$

considering the fact that a fraction ν_c^d of content c requests arriving at class d are re-routed to the server due to losses. Given that (3.4) holds, *i.e.*, the loss probability is arbitrarily small as B grows large, the total system costs can be approximated as:

$$\sum_{c\in\mathcal{C},d\in\mathcal{D}}\left(w^{ds}r_c^{ds}+\sum_{d'\in\mathcal{D}}w^{dd'}r_c^{dd'}\right) \tag{3.7}$$

Joint Request Routing and Content Placement Optimization

We next present a global optimization problem that allows a content provider to minimize its operational cost, by controlling request routing and content placement decisions for each class d.

The content provider deploys, manages these decentralized CDN boxes, and pays the cross-traffic costs to ISPs. It is therefore to its interest to minimize the operational costs. In particular, the service provider needs to determine (a) the content $\mathcal{F}_b$ placed in each box b, and (b) where the request generated by each box should be directed to, if the content is not locally cached. Solving this problem over millions of boxes poses a significant scalability challenge. Further, deciding where to place content and how to route requests is, in general, a combinatorial problem and hence computationally intractable. To address these issues, we propose a divide-and-conquer approach that first solves a global optimization problem across classes, and then implement detailed decisions inside each class. Through such an approximation, we wish to minimize the global cost, and at the same while ensure that all requests are served *w.h.p.* as the system size scales.

Let $\boldsymbol{r}^d=\{r_c^{dd'}\}_{d'\in\mathcal{D}\cup\{s\},c\in\mathcal{C}}$ and $\boldsymbol{p}^d=\{p_c^d\}_{c\in\mathcal{C}}$ be the request rates and replication ratios in class d, respectively. Let

$$F^d(\boldsymbol{r}^d)=\sum_{c\in\mathcal{C}}\left(w^{ds}r_c^{ds}+\sum_{d'\in\mathcal{D}}w^{dd'}r_c^{dd'}\right) \tag{3.8}$$

be the total cost generated by class d traffic. A lower bound on the operator's cost is provided by the solution to the following linear program

$$\textbf{GLOBAL} \tag{3.9a}$$

$$\text{minimize} \quad \sum_{d\in\mathcal{D}} F^d(\boldsymbol{r}^d) \tag{3.9b}$$

$$\text{subject to} \quad \sum_{c\in\mathcal{C}} p_c^d = M^d, \ \forall d \in \mathcal{D} \tag{3.9c}$$

$$\sum_{d'\in\mathcal{D}} r_c^{dd'} + r_c^{ds} = \lambda_c^d(1-p_c^d), \ \forall c \in \mathcal{C}, d \in \mathcal{D} \tag{3.9d}$$

$$\sum_{c\in\mathcal{C}} r_c^{\cdot d} < R^d, \ \forall d \in \mathcal{D} \tag{3.9e}$$

$$r_c^{\cdot d} < R^d p_c^d, \ \forall c\in\mathcal{C}, d \in \mathcal{D} \tag{3.9f}$$

$$\text{variables} \quad r_c^{dd'} \geq 0, r_c^{ds} \geq 0, p_c^d \geq 0, \ \forall c \in \mathcal{C}, d, d' \in \mathcal{D}$$

where $R^d = B^d U^d$ is the total upload capacity in class d.

The objective of the optimization problem is to minimize the total cost incurred by content transfers. Constraints (3.9c) and (3.9d) correspond to equations (3.2) and (3.3); they state that the full storage capacity of each class is used and that all requests are eventually served, respectively. Constraints (3.9e) and (3.9f) correspond to (3.5) and (3.6), respectively. This optimization problem is a linear program and can be solved efficiently using standard optimization techniques. We later introduce a distributed solution that is scalable and appealing to the geo-distributed infrastructure, without the need of a centralized coordinator that is prone to single point of failure.

3.2.3 System Architecture

We propose an efficient solution to minimize the operator's costs, given that request rates and replication ratios are chosen so as to solve the optimization problem (3.9), and within each class service assignment and content placement are performed such that (3.4) holds, as we will describe later in Sections 3.4 and 3.5. To implement these decisions in a distributed fashion, we propose a scalable system architecture by introducing *class trackers.* These trackers are deployed by the content provider inside each class to manage content distribution for boxes. They collectively solve the global optimization problem (3.9) in a distributed manner, and at the same while manage content placement and service assignment within their class.

Each class tracker has a complete view of the current state of every box inside its own class. In particular, it has the full visibility to (a) which content are stored in each box, (b) how many free upload slots they have, and (c) which content they are uploading. These statistics can be measured and maintained locally at the tracker, as both placing content at boxes and assigning requests to boxes are handled through the trackers, which we discuss below. Class trackers have no *a-priori* knowledge of the states of boxes in other classes, and only require a lightweight exchange of summary statistics among themselves. In particular, the operations performed by the tracker in class d are as follows:

Global Optimization. The tracker in class d determines (a) the replication ratio, *i.e.*, the fraction p_c^d of boxes in the class that store content $c \in \mathcal{C}$, and (b) the rate of requests $r_c^{dd'}$, $d' \in \mathcal{D} \cup \{s\}$ that are forwarded to the server or to other class trackers by solving **GLOBAL** in a distributed fashion. When performing this optimization, the tracker takes into account the traffic of requests entering the class d, as well as certain *congestion signals* it receives from other class trackers.

Request Routing. Having determined the rates $r_c^{dd'}$, the tracker implements a routing policy for requests generated within its class. In particular, a box that has a cache miss contacts the tracker, which then determines where the request should be routed to (*e.g.*, the server, a box within class d, or the tracker in another class) so that the total rates to $d' \in \mathcal{D} \cup \{s\}$ are given by $r_c^{dd'}$.

Service Assignment. Whenever a request (either internal or external) for a content item is to be served by a box in class d, the tracker determines which box in $\mathcal{B}^d$ is going to serve this request. In particular, the tracker uses a service assignment policy that determines how incoming requests for content c should be mapped to a box b that stores c—*i.e.*, $c \in \mathcal{F}_b$. If an incoming request cannot be served, the tracker re-routes this request to the server s.

Content Placement. After deciding the replication ratios p_c^d of each content item in class d, the tracker allocates the content items to boxes. That is, for each box $b \in \mathcal{B}^d$, it determines $\mathcal{F}_b$ in a manner so that (3.1) is satisfied.

Mgmt Type	Macroscopic/Global	Microscopic/Local
Content	Replication Ratio	Content Placement
Request	Request Routing	Service Assignment

Table 3.2: Operations performed by a class tracker

The above operations are summarized in Table 3.2. They can be grouped into content management operations (replication ratio and placement) and request management operations (routing and service assignment). These operations are of different control granularities, and happen at different time-scales. For instance, content replication and request routing are operations that involve the macroscopic behavior of a class, characterized by the aggregate information such as the replication ratios p_c^d and the request rates $r_c^{dd'}$. In contrast, content placement and service assignment are decisions that involve the microscopic management of resources in a class at the level of individual boxes. In the following sections, we describe each of these operations in more detail: we first show how to determine replication ratio and request routing in a distributed manner, followed by the service assignment and the content placement policies that together provably guarantee that all incoming content requests to a class are served *w.h.p.*.

3.3 A Decentralized Solution to the Global Problem

We now present how the trackers solve **GLOBAL** to determine their content replication ($\boldsymbol{p}^d$) and request routing ($\boldsymbol{r}^d$) parameters in a distributed fashion. In short, trackers exchange messages and adapt these values over several rounds. Our solution ensures that both the request rates and the replication ratios adapt in a smooth fashion, *i.e.*, changes between two iterations are incremental and the system does not oscillate wildly. This is important as abrupt changes to $\boldsymbol{p}^d$ require reshuffling the content of many boxes, which results in a considerable cost in data transfers.

Our presentation proceeds as follows: we first discuss why **GLOBAL** is difficult to solve in a distributed fashion with standard methods, and then present our distributed implementation.

3.3.1 Standard Dual Decomposition

Consider the partial Lagrangian

$$\mathcal{L}(\boldsymbol{r},\boldsymbol{p};\boldsymbol{\alpha},\boldsymbol{\beta}) = \sum_d F^d(\boldsymbol{r}^d) + \sum_{d'} \beta^{d'}\big(\sum_{c,d} r_c^{dd'} - R^{d'}\big) + \sum_{d'}\sum_c \alpha_c^{d'}\big(\sum_d r_c^{dd'} - R^{d'}p_c^{d'}\big)$$

where $\alpha_c^{d'}, \beta^d$ are the dual variables (Lagrange multipliers) associated with the constraints (3.9e) and (3.9f), respectively. Observe that $\mathcal{L}$ is separable in the primal variables, *i.e.*, it can be written as $\mathcal{L}(\boldsymbol{r},\boldsymbol{p},\boldsymbol{\alpha},\boldsymbol{\beta}) = \sum_d \mathcal{L}^d(\boldsymbol{r}^d,\boldsymbol{p}^d;\boldsymbol{\alpha},\boldsymbol{\beta})$ where

$$\mathcal{L}^d(\boldsymbol{r}^d,\boldsymbol{p}^d,\boldsymbol{\alpha},\boldsymbol{\beta}) = F^d(\boldsymbol{r}^d) + \sum_{d'}\left(\beta^{d'}\sum_c r_c^{dd'} + \sum_c \alpha_c^{d'} r_c^{dd'}\right) - \beta^d R^d - \sum_c \alpha_c^d R^d p_c^d.$$

This suggests a standard dual decomposition algorithm [53] for solving **GLOBAL** . The algorithm runs in multiple rounds, during which the tracker in class d maintains and updates both the primal and dual variables $\boldsymbol{r}^d(t), \boldsymbol{p}^d(t), \boldsymbol{\alpha}^d(t), \beta^d(t)$, corresponding to its class. We write these as functions of the round $t = 0, 1, \ldots$, as they are adapted at each round.

The tracker in class d also maintains estimates of the following quantities. First, for every $c \in \mathcal{C}$, it maintains an estimate of λ_c^d, *i.e.*, the request rate of c from boxes within its own class. Second, it maintains estimates of the quantities $r_c^{\cdot d}$, *i.e.*, the request rate for content c to be served by boxes in $\mathcal{B}^d$. In practice, λ_c^d and $r_c^{\cdot d}$ can be estimated by the use of appropriate counters, one for each rate. These can be incremented appropriately, *e.g.*, whenever a box within $\mathcal{B}^d$ issues a new request or an external request is received. The desired rate estimates can be obtained at the end of each round by dividing the counters by the round duration T. At the begining of a new round, all counters are reset to zero.

Using the estimates $r_c^{\cdot d}$, the tracker updates the dual variables α_c^d, β^d at the end of each round, increasing them when a constraints (3.9e) and (3.9f) are violated or decreasing them when the constraints are loose. That is, for all $c \in \mathcal{C}$,

$$\alpha_c^d(t) = \max\big\{0, \alpha_c^d(t-1) + \gamma(t)(r_c^{\cdot d}(t-1) - R^d p_c^d)\big\} \quad (3.10a)$$

$$\beta^d(t) = \max\big\{0, \beta^d(t-1) + \gamma(t)\big(\textstyle\sum_c r_c^{\cdot d}(t-1) - R^d\big)\big\} \quad (3.10b)$$

where $\gamma(t)$ a decreasing gain factor. At the end round t, each tracker shares its current dual variables with all other trackers. Having all dual variables $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ in the sytem, the trackers adapt their primal variables, reducing traffic forwarded to congested classes and increasing traffic forwarded to uncongested ones. This can be performed by setting:

$$(\boldsymbol{r}^d, \boldsymbol{p}^d)(t+1) = \operatorname*{argmin}_{(\boldsymbol{r}^d,\boldsymbol{p}^d)\in\mathcal{I}^d} \mathcal{L}^d(\boldsymbol{r}^d, \boldsymbol{p}^d; \boldsymbol{\alpha}(t), \boldsymbol{\beta}(t)). \tag{3.11}$$

where $\mathcal{I}^d$ is the set of pairs $(\boldsymbol{r}^d, \boldsymbol{p}^d)$ defined by (3.9c) and (3.9d) as well as the non-negativity constraints. Combined, adaptations (3.10) and (3.11) are known to converge to a maximizer of the primal problem *when the functions $\mathcal{L}^d$ are strictly convex* (see, *e.g.*, [53] Section 3.4.2, pp. 229-230). Unfortunately, this is not the case in our setup, as $\mathcal{L}^d$ are linear in both $\boldsymbol{r}^d$ and $\boldsymbol{p}^d$.

In practice, the lack of strict convexity makes $\boldsymbol{r}^d$,$\boldsymbol{p}^d$ oscillate wildly with every application of (3.11). This is disastrous in our system; wide oscillations of $\boldsymbol{p}^d$ imply that a large fraction of boxes in $\mathcal{B}^d$ need to change their content in each iteration. This is both impractical and costly; ideally, we would like each iteration to change the content of each class smoothly, so that the cost of implementing these changes is negligible.

3.3.2 A Distributed Implementation

We use an interior point method that deals with the lack of strict convexity called *the method of multipliers* [53]. First, we convert (3.9e) and (3.9f) to equality constraints by introducing appropriate slack variables y^d, $\boldsymbol{z}^d = [z_c^d]_{c\in\mathcal{C}}$:

$$\sum_{c\in\mathcal{C}} r_c^{\cdot d} + y^d = R^d, \quad \forall d \in \mathcal{D} \tag{3.12a}$$

$$r_c^{\cdot d} + z_c^d = R^d p_c^d, \quad \forall c \in \mathcal{C}, d' \in \mathcal{D} \tag{3.12b}$$

$$y^d \geq 0,\ z_c^d \geq 0, \qquad \forall c \in \mathcal{C}, d \in \mathcal{D} \tag{3.12c}$$

Under this modification, **GLOBAL** has the following properties. First, the objective (3.9b) is separable in the local variables $(\boldsymbol{r}^d, \boldsymbol{p}^d, y^d, \boldsymbol{z}^d)$, corresponding to each class. Second, and the constraints (3.12a) and (3.12b) coupling the local variables are linear equalities. Finally, the remaining constraints (3.9c) and (3.9d) as well as the positivity constraints define a bounded

```
Tracker $d$ at the end of round $k$:
  Obtain estimates of $\lambda_c^d$, $r_c^{\cdot d}$, $c \in \mathcal{C}$.
// Update dual variables
  $s_{tot}^d \leftarrow \frac{1}{|\mathcal{D}|}\left(\sum_{c\in\mathcal{C}} r_c^{\cdot d} + y^d - R^d\right)$
  $\beta^d \leftarrow \beta_c^d + \theta s_{tot}^d$
  for each content $c$
    $s_c^d \leftarrow \frac{1}{|\mathcal{D}|}\left(r^{\cdot d}(t)_c + z_c^d(t) - R^d p_c^d(t)\right)$
    $\alpha_c^d \leftarrow \alpha_c^d + \theta s_c^d$
  end for
  Broadcast $\left(\boldsymbol{\alpha}^d, \beta^d, \boldsymbol{s}^d, s_{tot}^d\right)$ to other trackers $d' \in \mathcal{D}$
  Receive dual variables from all other trackers $d' \in \mathcal{D}$
// Update primal variables
  $(\boldsymbol{r}^d, \boldsymbol{p}^d, \boldsymbol{z}^d, y^d) \leftarrow \underset{\mathcal{I}^d}{\operatorname{argmin}} \mathbf{LOCAL}^d(\boldsymbol{r}^d, \boldsymbol{p}^d, \boldsymbol{z}^d, y^d, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{s}, s_{tot})$
```

Figure 3.1: Decentralized solution to the global problem **GLOBAL**.

convex domain for the local primal variables. As a result, the method of multipliers admits a distributed implementation (see [53], Example 4.4., pp. 249–251).

Applying this distributed implementation to **GLOBAL** we obtain the algorithm summarized in Fig. 3.1. The tracker in class d maintains the following local variables, which correspond to the primal and dual variables of (3.9):

$$\boldsymbol{r}^d(t), \boldsymbol{p}^d(t), \boldsymbol{z}^d(t), y^d(t), \boldsymbol{\alpha}^d(t), \beta^d(t).$$

As in the dual decomposition method, the tracker in class d also maintains estimates of λ_c^d, *i.e.*, the request rate of c from boxes within its own class, and $r_c^{\cdot d}$, *i.e.*, the request rate for content c to be served by boxes in $\mathcal{B}^d$. Using these estimates, the primal and dual variables are updated as follows. At the end of round t, the tracker in class d uses the estimates of $r_c^{\cdot d}$ to see whether constraints (3.12a) and (3.12b) are violated or not. In particular, the tracker computes the quantities:

$$s_{tot}^d(t) = \left(\sum_c r_c^{\cdot d}(t) + y^d(t) - R^d\right)/|\mathcal{D}|$$
$$s_c^d(t) = \left(r_c^{\cdot d}(t) + z_c^d(t) - R^d p_c^d(t)\right)/|\mathcal{D}|, \quad \forall c \in \mathcal{C}$$

and updates the dual variables as follows:

$$\beta^d(t) = \beta^d(t-1) + \theta(t) s^d_{tot}(t)$$
$$\alpha^d_c(t) = \alpha^d_c(t-1) + \theta(t) s^d_c(t), \quad \forall c \in \mathcal{C}$$

where $\{\theta(t)\}_{t\in\mathbb{N}}$ are positive and non-decreasing. Subsequently, the tracker broadcasts to *every other tracker in* $\mathcal{D}$ a message containing its new dual variables as well as the congestion signals $\boldsymbol{\alpha}^d(t), \beta^d(t), \boldsymbol{s}^d(t), s^d_{tot}(t)$. Note that these comprise $2(|\mathcal{D}|+1)$ values, in total. For any $d, d' \in \mathcal{D}$, let

$$G^{d'}_{tot}(\boldsymbol{r}^d, y^d) = \textstyle\sum_c r^{dd'}_c + \mathbb{1}_{d=d'} y^d,$$
$$G^{d'}_c(\boldsymbol{r}^d, \boldsymbol{p}^d, \boldsymbol{z}^d) = r^{dd'}_c + \mathbb{1}_{d=d'}(z^d_c - R^d p^d_c).$$

Intuitively, these capture the "contribution" of the primal variables of class d to the constraints (3.12a) and (3.12b) of class d'. After the tracker in class d has received all the messages sent by other trackers, it solves the following quadratic program:

$\mathbf{LOCAL}^d(\boldsymbol{r}^d(t), \boldsymbol{p}^d(t), \boldsymbol{z}^d(t), y^d(t), \boldsymbol{\alpha}(t), \boldsymbol{\beta}(t), \boldsymbol{s}(t), \boldsymbol{s}_{tot}(t))$

$$\begin{aligned}
\text{minimize} \quad & F^d(\boldsymbol{r}^d) && (3.13a)\\
& + \sum_{d'} \beta^{d'}(t) G^{d'}_{tot}(\boldsymbol{r}^d, y^d) + \sum_{d',c} \alpha^{d'}_c(t) G^{d'}_c(\boldsymbol{r}^d, \boldsymbol{p}^d, \boldsymbol{z}^d) && (3.13b)\\
& + \frac{\theta(t)}{2} \sum_{d'} \Bigg[\Big(G^{d'}_{tot}\left(\boldsymbol{r}^d - \boldsymbol{r}^d(t), y^d - y^d(t)\right) + s^{d'}_{tot}(t) \Big)^2 && (3.13c)\\
& + \sum_c \Big(G^{d'}_c\left(\boldsymbol{r}^d - \boldsymbol{r}^d(t), \boldsymbol{p}^d - \boldsymbol{p}^d(t), \boldsymbol{z}^d - \boldsymbol{z}^d(t)\right) + s^{d'}_c(t) \Big)^2 \Bigg] && (3.13d)\\
\text{subject to} \quad & (\boldsymbol{r}^d, \boldsymbol{p}^d, \boldsymbol{z}^d, y^d) \in \mathcal{J}^d, \ \forall d \in \mathcal{D} && (3.13e)\\
\text{variables} \quad & \boldsymbol{r}^d, \boldsymbol{p}^d, \boldsymbol{z}^d, y^d, \quad d \in \mathcal{D}
\end{aligned}$$

where $\mathcal{J}^d$ is the set of quadruplets $(\boldsymbol{r}^d, \boldsymbol{p}^d, \boldsymbol{y}^d, z^d)$ defined by (3.9c) and (3.9d) as well as the non-negativity constraints. $\mathbf{LOCAL}^d$ thus receives as input *all* the dual variables $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, the congestion variables $\boldsymbol{s}^d$, $\boldsymbol{s}^d_{tot}$, *as well as* all the local primal variables at round t. The last four are included in the quadratic terms appearing in the objective function, and ensure the smoothness of the changes to the primal variables from one round to the next. The following theorem follows directly from the analysis in [53] and establishes that this algorithm indeed converges to an optimal solution.

Theorem 8. *Assume that the tracker in class d correctly estimates $\lambda_c^d, r_c^{\cdot d}$ in each iteration, and that $\{\theta(t)\}_{t\in\mathbb{N}}$ is a non-decreasing sequence of non-negative numbers. Then* $\lim_{t\to\infty}(\boldsymbol{r}^d(t), \boldsymbol{p}^d(t)) = (\boldsymbol{r}_*^d, \boldsymbol{p}_*^d)$*, where $(\boldsymbol{r}_*^d, \boldsymbol{p}_*^d)_{d\in\mathcal{D}}$ is an optimal solution to* (3.9)*.*

3.4 Request Routing and Service Assignment

In this section, we study how to route user requests, based on the information of replication ratios $\{\boldsymbol{p}^d\}$ and request rates $\{\boldsymbol{r}^d\}$ calculated by the global cost optimization problem (3.9). Our request routing includes two resolutions: (a) *inter-domain* request routing, *e.g.*, a user request is first directed to a class d (either its home class or a remote class), and (b) *intra-domain* service assignment, *e.g.*, a user request arriving at class d is further assigned to a specific box within class d that is capable of serving this request. The key idea of our service assignment strategy is that, at any time, the following property holds: given conditions (3.9e) and (3.9f) are true for a class d, *all* requests routed to class d are served *w.h.p.*. Intuitively, our strategy ensures that, as the total box population size B grows, the probability that a request reaching a class cannot be accommodated converges to zero.

3.4.1 Inter-Domain Request Routing

Content requests that are not locally served due to cache misses, must be directed to another box inside its home class, a remote class, or the server s. The global optimization problem (3.9) calculates the inter-domain request routing decision, *i.e.*, $\boldsymbol{r}^d$, for every class d. Eq. (3.9d) dictates that for every content c, the designated forwarding rates to all classes and the server, sum up to $\lambda_c(1-p_c^d)$—the effective request rates coming out this class. In practice, a class d request is directed to another class (or server) $d' \in \mathcal{D}\cup\{s\}$ with a probability proportional to $r_c^{dd'}$. As a result, request volumes forwarded from class d to d' are independent Poisson processes with rates $r_c^{dd'}$. This provides a simple way to implement the request routing decisions optimized by the global problem.

3.4.2 Intra-Domain Service Assignment

We next show intra-domain service assignment policies, *i.e.*, selecting a specific box inside a class, given the inter-domain request routing decisions made in the first step. The service assignment policy determines which box an incoming request should be assigned to upon its arrival. If no available box is found, the request is re-routed to the server. In what follows, we drop the superscript d, since we primarily focus on the analysis of a single class. We therefore denote by $\mathcal{B}$ the set of boxes in the class, by $B = |\mathcal{B}|$ the size of the box population, by U and M their upload and storage capacities, respectively, and by r_c the arrival rates of requests for content c.

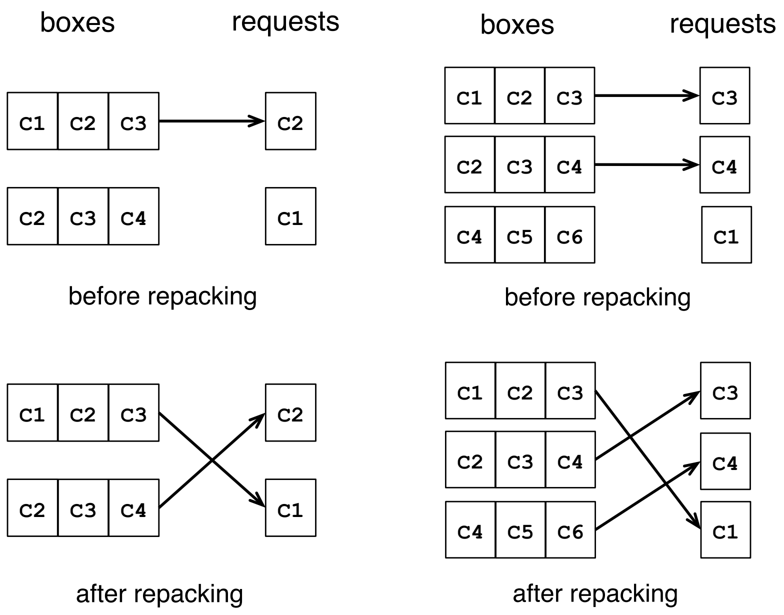
We study two service assignment policies—*repacking* and *uniform slot*—that map a request to a box that is capable of serving this request. For both policies, we establish a necessary and sufficient condition under which the system can serve all incoming traffic *asymptotically almost surely*, *i.e.*, *w.h.p.*, as the system size increases. In other words, we characterize the *capacity region* for the two service assignment policies, given a particular content placement decision by the system.

Uniform Slot Policy. Under this policy, an incoming request for content c is assigned to a box selected among all boxes currently storing c and having an empty upload slot. Each such box is selected with a probability proportional to the number of its empty slots. Equivalently, the request is matched to an upload slot selected uniformly from all free upload slots of boxes that can store c.

Definition 9. *Let X_b be the number of free upload slots of a box $b \in \mathcal{B}$. Under the uniform slot policy, an incoming request to class d for content c is matched to a slot selected uniformly at random among the $\sum_{b\in\mathcal{B}:c\in\mathcal{F}_b} X_b$ free slots over all boxes.*

Note that if this sum is zero—*i.e.*, no box storing content c has a free upload slot—the incoming request is re-routed to the server.

Repacking Policy. A limitation of the uniform slot policy is that it does not guarantee that an incoming request can be always accommodated, though the system is capable of serving the request by "migrating" existing services. One way to address this is through *repacking* [54, 55], as illustrated in the example of Figure 3.2(a).



(a) one-hop migration (b) two-hop migration

Figure 3.2: The repacking policy improves the resource utilization by allowing existing downloads to be migrated.

In this example, a request for content c_1 arrives but no box storing c_1 currently has a free slot. There exists a box b that stores c_1 but is using one of its upload slots for serving another content c_2. In addition, there exists another box b' that both stores c_2 and has a free slot. Now consider that if the download of content c_2 is migrated from b to b', it immediately releases a free slot at box b and allows the incoming request for c_1 to be served. Even if all boxes storing c_2 have no empty slots, it is possible that they can free one of their busy slots, through a chain of migration events, as illustrated in Figure 3.2(b). We term this strategy "repacking". Note that the operation of repacking does not require any change of existing content placement on any box, and thus can be implemented in practice.

In general, a repacking operation might involve multiple migrations of existing services, in order to free resources for an incoming request. To define it more precisely, consider a bipartite graph $G(V_{\text{req}} \cup V_{\text{slot}}, E)$, where V_{req} is the set of request vertices that each corresponds to a request, and V_{slot} is the set of slot vertices that each corresponds to an upload slot from some box. E is the set of edges, where an edge between a request and a slot exists if the slot belongs to a box that can serve the request, *i.e.*, storing the content that is requested for. The service capacity of the system is characterized by the following lemma:

Lemma 10. *All requests in V_{req} can be served if and only if there exists a maximum matching $M \subset E$ on G that is incident to all nodes in V_{req}.*

We define the repacking policy formally by introducing a matching problem as follows. At any time, let the graph $G(V_{\text{req}} \cup V_{\text{slot}}, E)$ include existing requests being served, and M be the maximum matching that represent the existing service assignment. When a new request arrives, we expand the graph G by adding it to V_{req} and creating edges between the request and all upload slots in V_{slot} that can serve it. If a new maximum matching M' exists, the incoming request is said to be served. In particular, M' can be adapted from M by finding an augmenting path that includes the new request in the bipartite graph. In practice, the augmenting path advises how existing requests should be migrated. We now formally define the *repacking* policy as follows:

Definition 11. *Under the repacking policy, an incoming request is matched to a slot by migrating existing service assignment, through the search of an augmenting path that finds a maximum matching in the expanded bipartite graph.*

If there exists no maximum matching, the request is dropped and re-routed to the server. As the size of the bipartite graph G cannot exceed $2B^dU^d$, the repacking policy can be implemented within a polynomial time in the number of boxes.

The repacking policy achieves a lower request dropping rate than the uniform slot policy, as allowing services to be migrated expectedly improves the resource utilization. However, this comes at the cost of a higher complexity, since we need to solve a maximum matching problem at the arrival of every new request. Further, migrating existing services may introduce additional user latency. [54] shows that, given certain conditions on content placement and request rates, the repacking policy achieves zero request dropping rate with a high probability, *i.e.*, all incoming requests directed to a class can be successfully served. While the uniform slot policy is simple and easy to implement in practice, it also begs the question whether it achieves the same optimal performance as the repacking policy.

3.4.3 Optimality of Uniform Slot Policy

The effectiveness of a service assignment policy depends on (i) how contents are placed across boxes, and (ii) the request rates for different contents. We study the conditions under which incoming requests can be successfully served with zero dropping rate, *i.e.*, we characterize the *capacity region* of the two service assignment policies. We first introduce a condition under which the repacking policy achieves zero dropping rate, and extend the result to the uniform slot policy. Given that the repacking policy is more sophisticated than the uniform slot policy, one might expect that the latter exhibits a higher loss probability. Nonetheless, we prove that the uniform slot policy also achieves the same performance asymptotically, *i.e.*, when the system size (number of boxes) is large. This suggests a practical design choice: the simple, light-weighted uniform-slot policy without losing the performance optimality.

Consider a collection of contents $\mathcal{F} \subset \mathcal{C}$ such that $|\mathcal{F}| = M$. Let $\mathcal{B}_{\mathcal{F}} = \{b \in \mathcal{B} : \mathcal{F}_b = \mathcal{F}\}$ be the set of boxes that store exactly $\mathcal{F}$. These sets partition $\mathcal{B}$ into sub-classes, each comprising boxes that store identical contents. As the number of boxes $B = |\mathcal{B}|$ goes to infinity, request arrival rates r_c and the size of each subclass $B_{\mathcal{F}} = |\mathcal{B}_{\mathcal{F}}|$ scale proportionally to B. That is, the following quantities

$$\rho_c = r_c/BU, \quad \forall c \in \mathcal{C}$$
$$\beta_{\mathcal{F}} = B_{\mathcal{F}}/B, \quad \forall \mathcal{F} \subset \mathcal{C}$$

are constants that do not depend on B. The scaling follows from our traffic model: as the number of boxes increases, the content demand, the aggregate storage, and the total upload capacity grow proportionally with the system population.

We formally define the performance metric as follows. Let ν_c be the *loss probability* of content c, *i.e.*, the steady state probability that a request for content c is dropped (and re-routed to the server) upon its arrival. We say that the requests for item c are served *asymptotically almost surely* (w.h.p.) if

$$\lim_{B \to \infty} \nu_c(B) = 0. \tag{3.14}$$

We characterize the capacity region of the uniform slot and the repacking policies, by answering the following question: given a fixed content placement across boxes, what are the conditions on request arrival rates such that (3.14) holds? Consider the following condition:

$$\sum_{c \in A} r_c < \sum_{\mathcal{F}:\mathcal{F}\cap A \neq \emptyset} B_{\mathcal{F}} U, \quad \forall A \subseteq \mathcal{C}. \tag{3.15}$$

Eq. (3.15) states that for any set of contents $A \subseteq \mathcal{C}$, the total request rates for these contents do not exceed the aggregate upload capacity of boxes that store at least one content in A. This condition is similar to the Hall's Theorem that establishes a maximum matching in a bipartite graph, however comes with changing traffic and graph. [54] shows that, if the repacking policy is used, condition (3.15) is *sufficient* for every content request to be served *w.h.p.*, *i.e.*, (3.14) holds. We present the following theorem, which establishes that if the uniform slot policy is used, (3.15) is *sufficient* for every content request to be served *w.h.p.*.

Theorem 12. *Given that* (3.15) *holds, and that requests are assigned to boxes according to the uniform slot policy,* (3.14) *is true,* i.e., *requests for every content $c \in \mathcal{C}$ are served* w.h.p..

We stress that it is an asymptotic result: for any system with a finite size, the repacking policy outperforms the uniform slot policy in terms of the loss probabilities ν_c. This, however, comes at the cost of a more complicated implementation for the repacking policy, which requires solving a maximum matching problem upon each arrival of content requests. In practice, a variant of the repacking policy that only involves migration chains within a limited length, *e.g.*, re-assigning requests that incur one or two migrations, can be used to further lower loss probabilities at a computational trade-off. However, as the system size B increases, the relative benefit vanishes. We later show in the evaluation that the uniform slot policy achieves a very close performance to the repacking policy in practice, under realistic traffic traces and system settings.

We present the detailed proof of Theorem 12 below. Readers can skip the proof without affecting the understanding of other sections.

Proof of Theorem 12. We partition the set of boxes $\mathcal{B}$ according to the contents they store in their cache. In particular, each of the boxes in $\mathcal{B}$ are grouped into L sub-classes $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_L$, where all boxes in the i-th class $\mathcal{B}_i$, $1 \leq i \leq L$, store the same set of contents $\mathcal{F}_i \subset \mathcal{C}$ such that $|\mathcal{F}_i| = M$.

We denote by $\mathcal{L} = \{1, 2, \dots, L\}$ and by $B_i = |\mathcal{B}_i|$ the number of boxes in the i-th sub-class. For $A \subseteq \mathcal{C}$, let

$$\mathsf{class}(A) = \{i \in \mathcal{L} : \mathcal{F}_i \cap A \neq \emptyset\}$$

be the set of sub-classes of boxes storing at least one content in A. And let

$$\beta_i = \frac{B_i U}{BU},\ i \in \mathcal{L} \quad \text{and} \quad \rho_c = \frac{r_c}{BU},\ c \in \mathcal{C}.$$

Note that, by our scaling assumption, when $B \to \infty$, the above quantities remain constant. Note that (3.15) can be rewritten as:

$$\sum_{c \in A} \rho_c < \sum_{j \in \mathsf{class}(A)} \beta_i,\ \forall A \subseteq \mathcal{C}.$$

Let X_i be the number of empty slots in the i-th subclass. Then, under the uniform slot service assignment policy, the stochastic process $\mathbf{X} : \mathbb{R}_+ \to \mathbb{N}^L$ is a Markov process and can be described as follows:

$$X_i(t) = X_i(0) + E_i^+ \left(\int_0^t B_i U - X_i(\tau) d\tau \right) - E_i^- \left(\int_0^t \sum_{c \in \mathcal{F}_i} r_c \frac{X_i(\tau)}{\sum_{j : c \in \mathcal{F}_j} X_j(\tau)} d\tau \right), \quad i \in \mathcal{L}, \tag{3.16}$$

where E_i^+, E_i^-, $i \in \mathcal{L}$, are independent unit-rate Poisson processes. Note that in the above we assume by convention that for all $i \in \mathcal{L}$ and all $c \in \mathcal{F}_i$,

$$\frac{X_i(\tau)}{\sum_{j : c \in \mathcal{F}_j} X_j(\tau)} = 0$$

whenever $\sum_{j : c \in \mathcal{F}_j} X_j(\tau) = 0$.

We will say that a mapping $\mathbf{x} : \mathbb{R}_+ \to [0, 1]^L$ is a *fluid trajectory* of the system if it satisfies the following set of equations:

$$x_i(t) = x_i(0) + \beta_i t - \int_0^t x_i(\tau) d\tau - \sum_{c \in F_i} \rho_c \int_0^t z_{c,i}(\mathbf{x}(\tau)) d\tau, \quad i \in \mathcal{L}. \tag{3.17}$$

where $z_{c,i}$, $c \in \mathcal{F}_i$, are functions satisfying

$$z_{c,i}(\mathbf{x}) = \frac{x_i}{\sum_{j:c\in\mathcal{F}_j} x_j}, \text{ if } \sum_{j:c\in\mathcal{F}_j} x_j > 0, \text{ and} \tag{3.18a}$$

$$z_{c,i}(\mathbf{x}) > 0, \sum_{i\in\mathsf{class}(\{c\})} z_{c,i} \leq 1 \text{ otherwise} \tag{3.18b}$$

Given a vector $\mathbf{x}_0 \in [0,1]^L$, we define $S(\mathbf{x}_0)$ to be the set of fluid trajectories defined by integral equation (3.17) with initial condition $\mathbf{x}(0) = \mathbf{x}_0$.

Lemma 13. *Consider a sequence of positive numbers $\{B^k\}_{k\in\mathbb{N}}$ such that $\lim_{k\to\infty} B^k = +\infty$, and a sequence of initial conditions $\mathbf{X}^k(0) = [x_i^k]_{1\leq i\leq L}$ s.t. the limit $\lim_{k\to\infty} \frac{1}{B^k}\mathbf{X}^k(0) = \mathbf{x}_0$ exists. Let $\{\mathbf{X}^k(t)\}_{t\in\mathbb{R}_+}$ denote the Markov process given by (3.16) given that $B = B_k$, and consider the rescaled process*

$$\mathbf{x}^k(t) = \frac{1}{B_k U}\mathbf{X}^k(t), \quad t \in \mathbb{R}_+.$$

Then for all $T > 0$ and all $\epsilon > 0$,

$$\lim_{k\to\infty} \mathbf{P}\left(\inf_{\mathbf{x}\in S(\mathbf{x}_0)} \sup_{t\in[O,T]} |\mathbf{x}^k(t) - \mathbf{x}(t)| \geq \epsilon\right) = 0$$

Proof. The proof is adapted from Massoulié [56]. The only key steps that we need to verify for our system is that for every i and every $c \in F_i$, (a) $X_i/\sum_{j:c\in\mathcal{F}_j} X_j$ is bounded (by 1) and (b) at its points of discontinuity $\mathbf{x}'$ (that is, such that $\sum_{j:c\in F_j} X_j$ is zero), then $\limsup_{\mathbf{x}\to\mathbf{x}'} X_i/\sum_{j:c\in\mathcal{F}_j} X_j = 1$ (where convergence is taken, *e.g.*, in the $\|.\|_1$ norm). Both are easy to verify. $\square$

Lemma 13 implies that (a) the set of fluid trajectories $S(\mathbf{x})$ is non-empty and (b) the rescaled process $\mathbf{x}^k$ converges on every finite set $[0,T]$ to a fluid trajectory, as $B \to \infty$, in probability. We therefore turn our attention to studying the asymptotic behavior of such fluid trajectories.

Given an $\mathbf{x}_0 \in [0,1]^L$, consider a fluid trajectory $\mathbf{x} \in S(\mathbf{x}_0)$. Since $z_{i,c}$ are bounded by 1, (3.17) implies that x is Lipschitz continuous (with parameter $\sum_c \rho_c$). Hence, by Rademacher's theorem, $\dot{x}$ exists almost everywhere and is given by

$$\dot{x}_i = \beta_i - x_i - \sum_{c\in F_i} \rho_c z_{c,i}(\mathbf{x}), \quad i \in \mathcal{L}. \tag{3.19}$$

Define the *limit set* [57] of ODE (3.19) to be

$$J = \lim_{t\to\infty} \bigcup_{\mathbf{y}\in[0,1]^L} \{\mathbf{x}(s), s \geq t : \mathbf{x}(0) = \mathbf{y}\}.$$

Then, Lemma 13 implies (see Thm. 3 in Benaïm and Le Boudec [58]) that, as k tends to infinity, the support of the steady state probability of $\mathbf{X}^k$ converges to a subset of J. Thus, to show that the probability that queries for every content item succeed asymptotically almost surely, it suffices to show that $x_i^* > 0$ for every $\mathbf{x}^* \in J$.

This is indeed the case. Let $I_0(\mathbf{x}) = \{i : x_i = 0\}$ be the zero-valued coordinates of $\mathbf{x}$ and

$$C(I) = \{c \in \mathcal{C} : \mathsf{class}(\{c\}) \subseteq I\}$$

denote the set of items stored only by classes in $I \subseteq \mathcal{L}$. Consider the following candidate Lyapunov function:

$$G(\mathbf{x}) = \sum_{i\in\mathcal{L}} \beta_i \log(x_i) - \sum_{i\in\mathcal{L}} x_i - \sum_{c\in\mathcal{C}} \rho_c \log\left(\sum_{j\in\mathsf{class}(\{c\})} x_j\right),$$

if $\mathbf{x} > 0$ and $G(\mathbf{x}) = -\infty$ otherwise.

Lemma 14. *Under* (3.15), *G is continuous in* $[0,1]^L$.

Proof. Indeed, consider a $\mathbf{x}'$ such that $I \equiv I_0(\mathbf{x}') \neq \emptyset$. Consider a sequence $\mathbf{x}^k \in [0,1]^L$, $k \in \mathbb{N}$, s.t. $\mathbf{x}^k \to \mathbf{x}$ in the $\|\cdot\|_\infty$ norm (or any equivalent norm in $\mathbb{R}^L$). We need to show that $\lim_{k\to\infty} G(\mathbf{x}^k) = -\infty$. If $I_0(\mathbf{x}^k) \neq \emptyset$ for some k, then $G(\mathbf{x}^k) = -\infty$; hence, w.l.o.g., we can assume that $\mathbf{x}^k \in (0,1]^L$. Then $G(\mathbf{x}^k) = A^k + B^k$, where

$$A^k = \sum_{i\in\mathcal{L}\setminus I} \beta_i \log(x_i^k) - \sum_{i\in\mathcal{L}} x_i^k - \sum_{c\in\mathcal{C}\setminus C(I)} \rho_c \log\left(\sum_{j\in\mathsf{class}(\{c\})} x_j^k\right), \quad \text{and}$$

$$B^k = \sum_{i\in I} \beta_i \log(x_i^k) - \sum_{c\in C(I)} \rho_c \log\left(\sum_{j\in\mathsf{class}(\{c\})} x_j^k\right).$$

The by the continuity of the log function, and the fact that $x_i' > 0$ for all $i \notin I$, it is easy to see that $\lim_{k\to\infty} A^k$ exists and is finite. On the other hand, if $C(I) = \emptyset$, B^k obviously converges to $-\infty$. Assume thus that $C(I)$ is non-empty. Partition I into classes $I_1, \ldots, I_m$ s.t. $x_i^k = y_\ell^k$ for all

$i \in I_\ell$ (*i.e.*, all coordinates in a class assume the same value).

$$\begin{aligned} B^k &\leq \sum_{i\in I}\beta_i \log(x_i^k) - \sum_{c\in C(I)} \rho_c \log\left(\max_{j\in \mathsf{class}(\{c\})} x_j^k\right) \\ &= \sum_{\ell=1}^m \log(y_\ell^k) \sum_{i\in I_\ell}\beta_i - \sum_{\ell=1}^m \log(y_\ell^k) \sum_{c\in C(I)} \rho_c \mathbb{1}_{y_\ell^k = \max_{j\in \mathsf{class}(\{c\})} x_j^k} \\ &\leq \sum_{\ell=1}^m \log(y_\ell^k)\left(\sum_{i\in I_\ell}\beta_i - \sum_{c\in C(I_\ell)}\rho_c\right) \end{aligned}$$

since $\mathbb{1}_{y_\ell^k = \max_{j\in \mathsf{class}(\{c\})} x_j^k} \leq \mathbb{1}_{\mathsf{class}(\{c\})\cap I_\ell = \emptyset}$ and $\log(y_\ell^k) < 0$ for k large enough. Under (3.15), the above quantity tends to $-\infty$ as $k \to \infty$, and the lemma follows. $\square$

Suppose that $I_0(\mathbf{x}(t)) = \emptyset$, *i.e.*. $\mathbf{x}(t) > 0$. Then (3.19) gives

$$\frac{d(\log x_i)}{dt} = \frac{\dot{x}_i}{x_i} = \frac{\beta_i}{x_i} - 1 - \sum_{c\in F_i}\rho_c \frac{1}{\sum_{j:c\in F_j} x_j} = \frac{\partial G}{\partial x_i}$$

Hence,

$$\frac{dG(\vec{x}(t))}{dt} = \sum_i \frac{\partial G}{\partial x_i}\dot{x}_i = \sum_i x_i \left(\frac{\partial G}{\partial x_i}\right)^2 \geq 0,$$

i.e., when at $\mathbf{x}$, G is increasing as time progresses under the dynamics (3.19). This, implies that if $\mathbf{x}(t) > 0$ then the fluid trajectory will stay bounded away from any $\mathbf{x}'$ s.t. $I_0(\mathbf{x}) \neq \emptyset$, as $G(\mathbf{x}') = -\infty$ and by Lemma 14 to reach such an x' the quantity $G(\mathbf{x}(t))$ would have to decrease, a contradiction.

Suppose now that $I = I_0(\mathbf{x}(t)) \neq \emptyset$. We will show that $I_0(\mathbf{x}(t+\delta) = \emptyset$, for small enough δ. Our previous analysis for the case $I_0(\mathbf{x})$ therefore applies and the theorem, as the limit set L cannot include points $\mathbf{x}^*$such that $I_0(\mathbf{x}^0) \neq \emptyset$. By (3.17) and (3.18), fluid trajectories are Lipschitz continuous; hence, for δ small enough, $x_i(t+\delta) > 0$ for all $i \notin I$. Moreover, by (3.19)

$$\begin{aligned} \frac{d\sum_{i\in I} x_i}{dt} &= \sum_{i\in I}\beta_i - \sum_{i\in I}\sum_{c\in F_i}\rho_c z_{c,i}(\mathbf{x}) \overset{(3.18a)}{=} \sum_{i\in I}\beta_i - \sum_{i\in I}\sum_{c\in F_i\cap C(I)}\rho_c z_{c,i}(\mathbf{x}) \\ &\overset{(3.18b)}{\geq} \sum_{i\in I}\beta_i - \sum_{c\in C(I)}\rho_c \overset{(3.15)}{>} 0. \end{aligned}$$

Hence, for δ small enough, there exists at least one $i \in I$ such that $x_i(t+\delta) > 0$. Given that x_i, $i \notin I$, will stay bounded away from zero within this interval, this implies that within δ time (where δ small enough) all coordinates in I will become positive. Hence, $I_0(\mathbf{x}(t+\delta) = \emptyset$, for small enough δ. $\square$

3.5 Content Placement

Condition (3.15) stipulates that every subset of $\mathcal{C}$ should be stored by enough boxes to serve incoming requests for all content in this set. In this section, we describe a content placement scheme under which, if the conditions (3.9e) and (3.9f) of **GLOBAL** hold, then so does (3.15). As a result, provided that (3.9e) and (3.9f) hold, this content placement scheme in combined with the uniform slot policy ensure that all requests are served *w.h.p.*. As the replication ratios $\boldsymbol{p}^d$ change from one round of the optimization to the next, so does content placement policies; our algorithm exploits the fact that changes in $\boldsymbol{p}^d$ are smooth, by reconfiguring placement with as few content exchanges as possible.

3.5.1 Designated Slot Placement

For every box $b \in \mathcal{B}^d$, we identify a special storage slot which we call the *designated slot.* We denote the content of this slot by D_b and the remaining contents of b by $L_b = \mathcal{F}_b \setminus \{D_b\}$. For all $c \in \mathcal{C}$, let $\mathcal{E}_c^d = \{b \in \mathcal{B}^d : D_b = c\}$ be the set of boxes storing c in their designated slot. As $\mathcal{E}_c^d$ are disjoint, we have

$$\sum_{\mathcal{F}:\mathcal{F}\cap A \neq \emptyset} B_{\mathcal{F}}^d U^d = \sum_{b \in \mathcal{B}^d : \mathcal{F}_b \cap A \neq \emptyset} U^d = \sum_{b \in \mathcal{B}^d : D_b \in A} U^d + \sum_{b \in \mathcal{B}^d : D_b \notin A} U^d \geq \sum_{c \in A} |\mathcal{E}_c^d| U^d, \quad \forall A \subseteq \mathcal{C}.$$

Hence, the following lemma holds:

Lemma 15. *If* $|\mathcal{E}_c^d| > r_c^{\cdot d}/U^d$ *then* (3.15) *holds.*

Lemma 15 implies that the exact fraction of boxes that store content c in their designated slot should exceed $r_c^{\cdot d}/B^d U^d$ but not p_c^d. The following lemma states that such fractions can be easily computed if the conditions in **GLOBAL** hold.

Lemma 16. *Given a class d, consider $r_c^{\cdot d}$ and p_c^d, $c \in \mathcal{C}$, for which* (3.9e) *and* (3.9f) *hold. There exist $q_c^d \in [0,1]$, $c \in \mathcal{C}$, such that*

$$\sum_c q_c^d = 1, \quad 0 \le r_c^{\cdot d}/B^d U^d < q_c^d \le p_c^d \le 1, \ \forall c \in \mathcal{C}. \tag{3.20}$$

Moreover, such q_c^d can be computed in $O(|\mathcal{C}|\log|\mathcal{C}|)$ time.

Proof. We show a constructive proof below to calculate q_c^d that satisfy (3.20). If $M^d = 1$, the lemma trivially holds for $q_c^d = p_c^d$. Now suppose that $M^d \ge 2$. Let $\epsilon = 1 - \sum_{c\in\mathcal{C}} r_c^{\cdot d}/B^d U^d$ and $\epsilon_c = p_c^d - r_c^{\cdot d}/B^d U^d$. From (3.9e) and (3.9f), we have that $\epsilon > 0$ and $\epsilon_c > 0$. Sort ϵ_c in an increasing fashion, so that $\epsilon_{c_1} \le \epsilon_{c_2} \le \ldots \le \epsilon_{c_{|\mathcal{C}|}}$. If $\epsilon_{c_1} \ge \epsilon$, then the lemma holds for $q_c^d = r_c^{\cdot d}/B^d U^d + \epsilon/|\mathcal{C}|$. Assume thus that $\epsilon_{c_1} < \epsilon$. Let $k = \max\{j : \sum_{i=1}^{j} \epsilon_{c_i} < \epsilon\}$. Then $1 \le k < |\mathcal{C}|$, as $\sum_{c\in\mathcal{C}} \epsilon_c \overset{(3.9c)}{=} M - 1 + \epsilon > M - 1 > \epsilon$ for $M \ge 2$. Then, $\epsilon' = \epsilon - \sum_{i=1}^{k} \epsilon_{c_i} > 0$, by the definition of k. Let $q_{c_i}^d = r_c^{\cdot d}/BU + \epsilon_{c_i}$ for $i \le k$ and $q_{c_i}^d = r_c^{\cdot d}/BU + \epsilon'/(|\mathcal{C}| - k)$ for $i > k$. Then $q_{c_i}^d = p_{c_i}^d > \lambda_c/BU$ for $i \le k$. For $i > k$, $q_{c_i}^d > r_c^{\cdot d}/BU$ as $\epsilon' > 0$ while $\epsilon'/(|\mathcal{C}| - k) \le \epsilon' \le \epsilon_{c_{k+1}}$, as otherwise $\sum_{i=1}^{k+1} \epsilon_{c_i} < 1$, a contradiction. Moreover, $\epsilon_{c_{k+1}} \le \epsilon_{c_i}$ for all $i \ge k$, so $q_{c_i}^d \le r_{c_i}^{\cdot d}/BU + \epsilon_{c_i} \le p_c^d$. Finally, $\sum_c q_c^d = \sum_{i=1}^{k}(r_{c_i}^{\cdot d}/BU + \epsilon_{c_i}) + \sum_{i=k+1}^{\mathcal{C}} r_{c_i}^{\cdot d}/BU + \epsilon' = \sum_c r_c^{\cdot d}/BU + \epsilon = 1$, and the lemma follows. □

In other words, if (3.9e) and (3.9f) of **GLOBAL** hold, ensuring that requests for all contents are served *w.h.p.* in class d is acheived by placing content c in the designated slot of at least $q_c^d B^d$ boxes, where $q_c^d B^d$ are determined as in Lemma 16. We call such a placement scheme a *designated slot placement*. Below, we describe an algorithm that, given ratios q_c^d and p_c^d, places content in class d in a way that these ratios are satisfied.

3.5.2 An Algorithm Constructing a Designated Slot Placement

For simplicity, we drop the superscript d in the remainder of this section, though we are referring to content placement in a single class. We focus on the scenario where we are given an initial content placement $\{\mathcal{F}_b\}_{b\in\mathcal{B}}$ over B boxes in set $\mathcal{B}$. Our algorithm, outlined in Figure 3.3, receives this placement as well as target replication ratios q'_c and p'_c, $c \in \mathcal{C}$, satisfying (3.20). It outputs a new

```
Input: Initial placement $\{F_b\}_{b\in\mathcal{B}}$ and target ratios $q'_c$, $p'_c$
Let $A^+ := \{c \in \mathcal{C} : q_c > q'_c\}$, $A^- := \{c :\in \mathcal{C} : q_c < q'_c\}$;
while there exists $b \in \mathcal{B}$ s.t. $D_b \in A^+$ and $L_b \cap A^- \neq \emptyset$
  Pick $c \in L_b \cap A^-$, and swap it locally with the content of $D_b$.
  Update $\boldsymbol{q}$, $\boldsymbol{\pi}$, $A^+$, $A^-$ accordingly
while there exists $b \in \mathcal{B}$ s.t. $D_b \in A^+$ and $L_b \cap A^- = \emptyset$
  Pick $c \in A^-$ and place $c$ in the designated slot $D_b$;
  Update $\boldsymbol{q}$, $\boldsymbol{\pi}$, $A^+$, $A^-$ accordingly
Let $C^+ := \{c : \pi_c > \pi'_c\}$; $C^+ := \{c : \pi_c < \pi'_c\}$;
$C^0 := \mathcal{C} \setminus (C^+ \cup C^-)$.
Let $G := \{\ b \in \mathcal{B}$ s.t. $C_+ \cap L_b \neq \emptyset$ and $C_- \setminus (D_b \cup L_b) \neq \emptyset\ \}$;
while $(G \neq \emptyset)$ or (there exists $c \in C^-$ s.t. $(\pi_c - \pi'_c)B \geq 2$)
  if $(G \neq \emptyset)$ then
    Pick any $b \in G$
    Replace some $c \in C_+ \cap L_b$ with some $c' \in C_- \setminus (D_b \cup L_b)$;
  else
    Pick $c \in C^-$ s.t. $(\pi_c - \pi'_c)B \geq 2$.
    Find a box $b$ that does not store $c$.
    Pick $c' \in C_0 \cap L_b$ and replace $c'$ with $c$.
  update $G$, $\boldsymbol{\pi}$, $C^+$, $C^-$, $C^0$ accordingly.
```

Figure 3.3: Placement Algorithm

content placement $\{\mathcal{F}'_b\}_{b\in\mathcal{B}}$ in which q'_cB boxes store c in their designated slot, while *approximately* p'_cB boxes store c overall. Moreover, it does so *with as few changes of box contents as possible.*

We assume that q'_cB and p'_cB are integers—for large B, this is a good approximation. Let q_c, p_c be the corresponding designated slot and overall fractions in the input placement $\{F_b\}_{b\in\mathcal{B}}$. Let $\pi_c = p_c - q_c$, $\pi'_c = p'_c - q'_c$. A lower bound on the number of cache modification operations needed to go to the target replication ratios is given by $B\delta/2$, where $\delta = \sum_c |p_c - p'_c|$. Our algorithm's performance in terms of cache modification operations will be expressed as a function of the cache size M and of the quantities $\alpha = \sum_c |q_c - q'_c|$, $\beta = \sum_c |\pi_c - \pi'_c|$:

Theorem 17. *The content placement algorithm in Fig. 3.3 leads to a content replication $\{F'_b\}_{b\in\mathcal{B}}$ in which exactly q'_cB boxes store c in their designated slot, and p''_cB boxes store c overall, where $\sum_c |p'_c - p''_c|B < 2M$, and $|p'_c - p''_c|B \leq 1$, for all $c \in \mathcal{C}$. The total number of write operations is at most $B[\alpha + (M-1)(\alpha + \beta)]/2$.*

In other words, the algorithm produces a placement in which at most $2M$ contents are either under or over-replicated, each one only by one replica.

Proof of Thm. 17. To modify the designated slots, the algorithm picks any over-replicated content c in set $A_+ = \{c : q_c > q'_c\}$. For any user holding c in its designated slot, it checks whether it holds in its normal slots an under-replicated content $c' \in A_- = \{c : q_c < q'_c\}$. If such content exists, it renames the corresponding slot as "designated" and the slot holding c as "normal". This incurs no cache delete-write, and reduces the l_1 norm—*i.e.*, the imbalance—between the vectors $B\boldsymbol{q}$ and $B\boldsymbol{q'}$ by 2. This is repeated until an under-replicated content c' cannot be found within the normal cache slots of boxes storing some $c \in A_+$. If there still are over-replicated items in A_+, some $c' \in A_-$ is selected arbitrarily and overwrites c within the designated slot. This again reduces imbalance by 2, involving one delete-write operation.

At the end of this phase, the replication rates within the designated slots have reached their target $B\vec{q'}$, incurring a cost of at most $B\alpha/2$ operations. The resulting caches are free of duplicate copies. Also, after these operations, the intermediate replication rates within the normal cache slots $B\pi_c$, verify $|B\pi_c - B\pi''_c| \leq |Bq_c - Bq'_c|$.

We now consider the transformation of the intermediate replication rates π_c into replication rates π'_c. To this end, we distinguish contents c that are over-replicated, under-replicated and perfectly replicated by introducing $C_+ = \{c : \pi_c > \pi'_c\}$, $C_- = \{c : \pi_c < \pi'_c\}$, $C_0 = \{c : \pi_c = \pi'_c\}$.

For any box b, if there exists $c \in C_+ \cap L_b$, and $c' \in C_- \setminus (D_b \cup L_b)$, the algorithm replaces c by c' within L_b, thereby reducing the l_1 distance—*i.e.*, the imbalance—between vectors $B\boldsymbol{\pi}$ and $B\boldsymbol{\pi'}$ by 2. We call the corresponding operation a *greedy reduction.* Eventually, the algorithm may arrive at a configuration where no such changes are possible. Then, for any box b such that $C_+ \cap L_b$ is not empty, necessarily $C_- \subset (L_b \cup D_b)$. Hence, the size of C_- is at most $M - 1$.

In that case, the algorithm picks some content c' that is under-replicated by at least 2 replicas, and finds a user b which does not hold c', *i.e.* $c' \in C_- \setminus (D_b \cup L_b)$. It also selects some content c within $C_0 \cap L_b$: such content must exist, since $|C_-| \leq M - 1$, and necessarily $C_- \cap L_b \subset C_- \setminus \{c'\}$ has size strictly less than $M - 1$, the size of L_b; the remaining content c must belong to C_0 since otherwise we could have performed a greedy reduction.

We then replace content c by content c'. This does not change the imbalance, but augments the size of set C_-: indeed content c is now under-replicated (one replica missing). We then try to do a greedy reduction, *i.e.*, a replacement of an over-replicated content by c if possible. If not, we repeat the previous step, *i.e.*, by identifying some content under-replicated by at least 2, and creating a new replica in place of some perfectly replicated item, thereby augmenting the size of C_- while maintaining imbalance. In at most $M - 1$ steps, the algorithm inflates the size of C_- to at least M, at which stage we know that some greedy imbalance reduction can be performed. This procedure terminates when the size of C_- is at most $M-1$, and each of the corresponding contents is missing only one replica. The imbalance, which was after the termination of the designated slot placement at most $N\sum_c |\pi_c - \pi'_c| \leq B(\alpha+\beta)$, has been reduced to at most $2M$. The number of write operations involved in each imbalance reduction by 2 is at most $M-1$. Thus the total number of write operations in this second phase is upper-bounded by $(M-1)(B/2)(\alpha+\beta)$. $\square$

3.6 Performance Evaluation

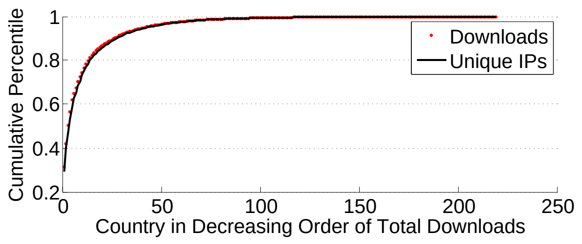
In this section, we evaluate our algorithms under synthesized traces and a real-life BitTorrent traffic trace. We implement an event-driven simulator that captures box-level content placement and service assignment. In particular, we implement the solutions we proposed in Section 3.3 (**LOCAL**^{d} optimization), Section 3.4 (Uniform Slot Policy), Section 3.5.1 (Designated Slot Placement). We also implement ISP trackers that execute the decentralized solution in Figure 3.1, *e.g.*, redirecting requests, estimating demands and exchanging messages for decentralized optimization.

In the rest of this section, all evaluations are performed using a cost matrix with uniform $[0,1]$ class-pairwise cost. The cost to itself is always zero, and the cost is set to constant 3 if a content is downloaded from the server either due to dropped request or designated routing decision.

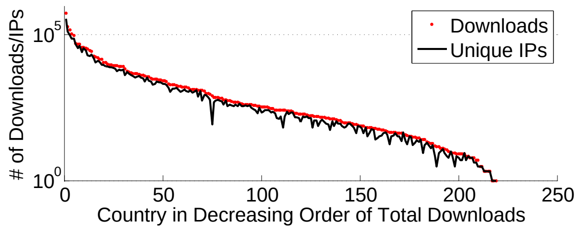
3.6.1 Uniform-slot service assignment

First, we show that the uniform-slot policy achieves close-to-optimal service assignment, given that content-wise capacity constraints are respected. We focus on a single ISP and assign requests to individual boxes under the uniform-slot policy. Figure 3.5 shows the dropping probability under various settings of B and M. We utilize a synthesized trace with Poisson arrivals and exponentially distributed service time. Content popularity follows a Zipf distribution and we let the total traffic rate grow proportionally to the system population.

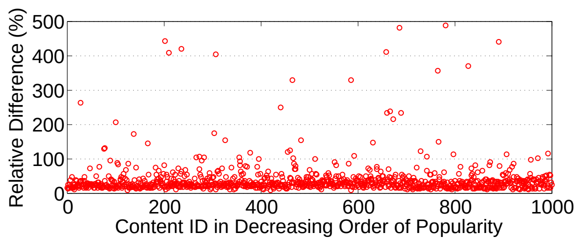
As predicted by the theory, the dropping probability quickly vanishes as B grows. For the same B, the dropping probability is higher when M is larger. This is due to the undermined accuracy of content placement when M is large. In addition, when the storage is rich, our scheme tries to allocate popular content in caches in order to minimize cost. This results in more requests



(a)



(b)



(c)

Figure 3.4: Characterization of real-life BitTorrent trace. (a) Cumulative counts of downloads/boxes. (b) Per-country counts of downloads/boxes. (c) Predictability of content demand in one hour interval over one month period.

from less popular content, which are originally sent to the server. Serving less popular content understandably incur more drops.

3.6.2 Synthesized trace based simulation

We next show the efficiency of our full solution with decentralized optimization, content placement scheme and uniform-slot policy. We again utilize a synthesized trace generated as in the previous one. Content popularities are heterogeneous in different classes. Figure 3.6 illustrates the average empirical cost per request, compared to the fluid prediction under distributed optimization and the optimal cost computed offline. The distributed algorithm quickly converges to the global

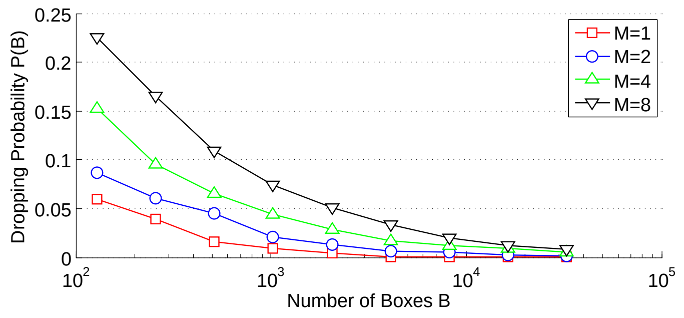


Figure 3.5: Dropping probability decreases fast with uniform slot strategy. Simulation in a single class with a catalog size of $C = 100$.

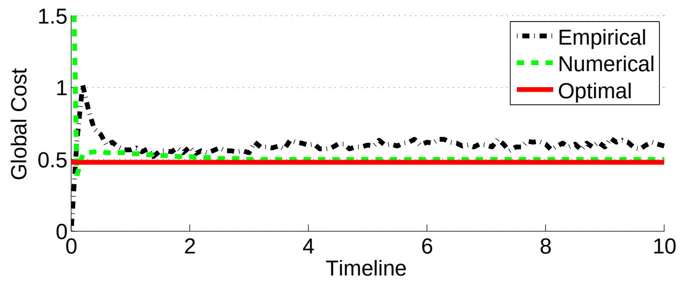


Figure 3.6: Performance of full solution with decentralized optimization, content placement scheme and uniform-slot policy, under the parameter settings $C = 1000, D = 10, \bar{B} = 1000, \bar{U} = 3, \bar{M} = 4$.

optimum after a handful of iterations. The results suggest that in the steady state, the empirical cost achieved by our solution is very close to the offline optimal.

3.6.3 BitTorrent trace-based simulation

We finally employ a real-life trace collected from the Vuze network, one of the most popular BitTorrent clients. To collect the traces, we ran 1000 DHT nodes with randomly-chosen IDs, and logged all DHT *put* messages routed to our nodes. During 30 days we traced the downloads of around 2 million unique files by 8 million unique IP addresses. We determined the country of each IP using Maxmind's GeoLite City database [59].

To limit the runtime of our simulations, we trim the traces as follows. We consider only the top 1,000 most popular files, which contribute 52% of the total downloads. Figure 3.4(a) illustrates the total number of download events and unique IPs, grouped by countries in a decreasing order

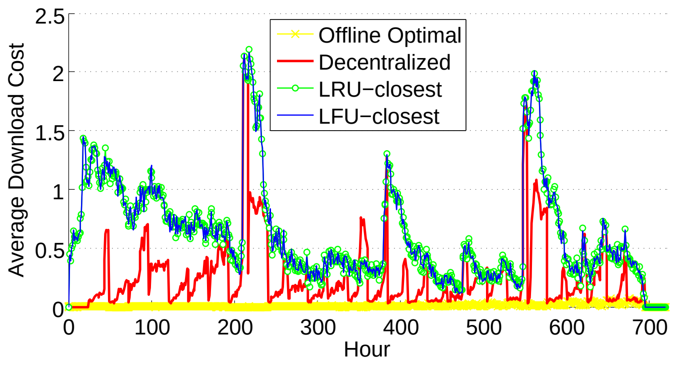


Figure 3.7: Performance of different algorithms over a real 30-day BitTorrent trace.

of total downloads during the 30-day period, and the cumulative counts in Figure 3.4(b). The traces show that one user issues, on average, approximately one content request. We select the top 20 countries as classes in our simulation.

So far we have shown the efficiency of our algorithm given fixed request rates. In practice, the content popularity may change over time and cannot be accurately predicted from the history. The content demand should remain sufficiently stable in order for our algorithm to work well and not oscillate wildly. In our simulation, we measure content demands based on a 24-hour interval, and use the demand rates from the previous day as the prediction for the next. Figure 3.4(c) shows the relative difference between the predicted rate and the true rate, grouped by content in decreasing order of popularity. Each data point is averaged over the entire 30-day period. The results show that demand is stable for the majority of content. The ones with high variations come from new content that reach their popularity peaks during the first few days after their generations.

In the following simulations, we let each user have 2 upload slots and 2 storage slots. We compare our solution to two common caching algorithms, *e.g.*, Least Recently Used (LRU) and Least Frequently Used (LFU). The request routing follows the “closest” approach: when a download request is issued, we go over all classes in increasing cost order, *e.g.*, starting from the requester’s class. The request is accepted whenever a class is found for which there exists at least one box with one or more free slots that stores the requested content. If no such class is found, the request is redirected to the server. We run the trace under four content placement and routing algorithms: our solution, offline optimal, LRU-closest and LFU-closest. All algorithms start with the same configuration of content placement. Figure 3.7 shows the average download cost for the four cases. The results show that our solution significantly reduces the cross-traffic between classes most of the time compared to the two heuristics. The spikes are mainly due to incorrectly predicting the demand of content with highly varying popularities. The offline algorithm works best because of its perfect demand prediction and its instant memory reshuffles.

3.7 Related Work

Peer-assistance of CDNs has been studied from several perspectives. Recent work has shown that it can reduce CDN server traffic [60] and energy consumption [8] by more than 60%. Early research compared the efficiency of prefetching policies for peer-assisted VoD [61], and bandwidth allocation across different P2P swarms [62]. Issues regarding peer incentivization [63, 64, 65] have also been studied.

Minimizing cross-traffic is extensively studied in the context of "ISP-friendly" P2P system design, whose goal is to encourage content downloads within the same ISP. This is known to reduce both ISP cross-traffic and download delays, being thus mutually benefitial to ISPs and peers alike [66, 67]. Typical implementations bias the selection of download sources towards nearby peers; peer proximity can be inferred either through client-side mechanisms [44] or through a service offered by the ISP [68, 69, 19]. In the latter case, the ISP can explicitly recommend which neighbors to download content from by solving an optimization problem that minimizes cross-traffic [19]. In the context of peer-assisted CDNs, an objective that minimizes a weighted sum of cross-traffic and the load on the content server can also be considered [69]. Prior work on ISP-friendliness reduces cross traffic solely by performing *service assignment* to suitable peers. In our context, we have an additional control knob on top of service assignment, namely *content placement*: our optimization selects not only where requests are routed, but also where content is stored.

In the *cooperative caching problem*, clients generate a set of requests for items, that need to be mapped to caches that can serve them; each client/cache pair assignment is also associated with an access cost. The goal is to decide how to place items in caches and assign requests to caches that can serve them, so that the total access cost is minimized. The problem is NP-hard, and a 10-approximation algorithm is known [70]. Motivated by CDN topologies, Borst *et al.* [71]

obtain lower approximation ratios as well as competitive online algorithms for the case where cache costs are determined by weights in a star graph. A polynomial algorithm is known in the case where caches are organized in a hierarchical topology and the replica accessed is always the nearest replica [72]. Our work significantly departs from the above studies by explicitly dealing with bandwidth constraints, assuming a stochastic demand for items and proposing an adaptive, distributed algorithm for joint content placement and service assignment among multiple *classes* of identical caches.

Finally, recent work [73, 74, 54] has considered cache management specifically in the context of P2P VoD systems, and is in this sense close to our work. However the multi-group dimension is not addressed in these works; also, at the exception of [54], the request service policies in [73, 74] differ from ours, which is based on a loss model.

3.8 Summary

We offer a solution to regulate cross-traffic and minimize content delivery costs in decentralized CDN's. We present an optimal request routing scheme that can nicely accommodate user demands, an effective service mapping algorithm that is easy to implement within each operator, and an adaptive content caching algorithm with low operational costs. Through a live BitTorrent trace-based simulation, we demonstrate that our distributed algorithm is simultaneously scalable, accurate and responsive.

Chapter 4

DONAR: Decentralized Server Selection for Cloud Services

This chapter focuses on the challenge of a *distributed implementation* of traffic management solutions across a large number of geographically distributed clients [10]. Geo-replicated services need an effective way to direct client requests to a particular location, based on performance, load, and cost. This chapter presents DONAR, a distributed system that can offload the burden of replica selection, while providing these services with a sufficiently expressive interface for specifying mapping policies. Most existing approaches for replica selection rely on either central coordination (which has reliability, security, and scalability limitations) or distributed heuristics (which lead to suboptimal request distributions, or even instability). In contrast, the distributed mapping nodes in DONAR run a simple, efficient algorithm to coordinate their replica-selection decisions for clients. The protocol solves an optimization problem that jointly considers both client performance and server load, allowing us to show that the distributed algorithm is stable and effective. Experiments with our DONAR prototype—providing replica selection for CoralCDN and the Measurement Lab—demonstrate that our algorithm performs well "in the wild." Our prototype supports DNS- and HTTP-based redirection, IP anycast, and a secure update protocol, and can handle many customer services with diverse policy objectives.

4.1 Introduction

Cloud services need an effective way to direct clients across the wide area to an appropriate service location (or "replica"). For many companies offering distributed services, managing replica selection is an unnecessary burden. In this chapter, we present the design, implementation, evaluation, and deployment of DONAR, a decentralized replica-selection system that meets the needs of these services.

4.1.1 A Case for Outsourcing Replica Selection

Many networked services handle replica selection themselves. However, even the simplest approach of using DNS-based replica selection requires running a DNS server that tracks changes in which replicas are running and customizes the IP address(es) returned to different clients. These IP addresses may represent single servers in some cases. Or, they may be virtualized addresses that each represent a cluster of collocated machines, with an on-path load balancer directing requests to individual servers. To handle wide-area replica selection *well*, these companies need to (i) run the DNS server at multiple locations, for better reliability and performance, (ii) have these nodes coordinate to distribute client requests across the replicas, to strike a good trade-off between client performance, server load, and network cost, and (iii) perhaps switch from DNS to alternate techniques, like HTTP-based redirection or proxying, that offer finer-grain control over directing requests to replicas.

One alternative is to outsource the entire responsibility for running a Web-based service to a CDN with ample server and network capacity (*e.g.*, Akamai [6]). Increasingly, cloud computing offers an attractive alternative where the cloud provider offers elastic server and network resources, while allowing customers to design and implement their own services. Today, such customers are left largely to handle the replica-selection process on their own, with at best limited support from individual cloud providers [75] and third-party DNS hosting platforms [76, 77].

Instead, companies should be able to manage their own distributed services while "outsourcing" replica selection to a third party or their cloud provider(s). These companies should merely specify high-level *policies*, based on performance, server and network load, and cost. Then, the replica-selection system should realize these policies by directing clients to the appropriate replicas and adapting to policy changes, server replica failures, and shifts in the client demands. To be effective, the replica-selection system must satisfy several important goals for its customers. It must be:

- **Expressive:** Customers should have a sufficiently expressive interface to specify policies based on (some combination of) performance, replica load, and server and bandwidth costs.
- **Reliable:** The system should offer reliable service to clients, as well as stable storage of customer policy and replica configuration data.
- **Accurate:** Client requests should be directed to the service replicas as accurately as possible, based on the customer's replica-selection policy.
- **Responsive:** The replica-selection system should respond quickly to changing client demands and customer policies without introducing instability.
- **Flexible:** The nodes should support a variety of replica-selection mechanisms (*e.g.*, DNS and HTTP-redirection).
- **Secure:** Only the customer, or another authorized party, should be able to create or change its selection policies.

In this chapter, we present the design, implementation, evaluation, and deployment of DONAR, a decentralized replica-selection system that achieves these goals. DONAR's distributed algorithm solves a formal optimization problem that jointly considers both client locality, server load, and policy preferences. By design, DONAR facilitates replica selection for *many* services, however its underlying algorithms remain relevant in the case of a single service performing its own replica selection, such as a commercial CDN.

4.1.2 Decentralized Replica-Selection System

The need for reliability and performance should drive the design of a replica-selection system, leading to a *distributed* solution that consists of multiple *mapping nodes* handling a diverse mix of clients, as shown in Figure 4.1. These mapping nodes could be HTTP ingress proxies that route client requests from a given locale to the appropriate data centers, the model adopted by Google and Yahoo. Or the mapping nodes could be authoritative DNS servers that resolve local queries for the names of Web sites, the model adopted by Akamai and most CDNs. Furthermore, these DNS servers may use IP anycast to leverage BGP-based failover and to minimize client request

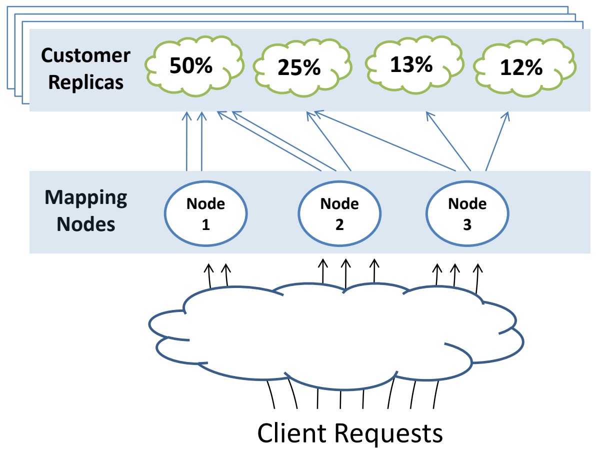


Figure 4.1: DONAR uses distributed mapping nodes for replica selection. Its algorithms can maintain a weighted split of requests to a customer's replicas, while preserving client–replica locality to the greatest extent possible.

latency. Whatever the mechanism for interacting with clients, each mapping node has only a partial view of the global space of clients. As such, these mapping nodes need to make different decisions; for example, node 1 in Figure 4.1 directs all of its clients to the leftmost replica, whereas node 3 divides its clients among the remaining three replicas.

Each mapping node needs to know how to both direct its clients and adapt to changing conditions. The simplest approach would be to have a central coordinator that collects information about the mix of clients per customer service, as well as the request load from each mapping node, and then informs each mapping node how to direct future requests. However, a central coordinator introduces a single point of failure, as well as an attractive target for attackers trying to bring down the service. Further, it incurs significant overhead for the mapping nodes to interact with the controller. While some existing services do perform centralized computation—for example, Akamai uses a centralized hierarchical stable-marriage algorithm for assigning clients to its CDN servers [78]—the overhead of backhauling client information can be more prohibitive when it must be done for *each* customer service using DONAR. Finally, a centralized solution adds additional delay, making the system less responsive to sudden changes in client request rates (*i.e.*, flash crowds).

To overcome these limitations, DONAR runs a *decentralized* algorithm among the mapping nodes themselves, with minimal protocol overhead that does not grow with the number of clients. Designing a distributed algorithm that is simultaneously scalable, accurate, and responsive is an open challenge, not addressed by previous heuristic-based [79, 80, 81, 82] or partially centralized [83] solutions. Decentralized algorithms are notoriously prone to *oscillations* (where the nodes over-react based on their own local information) and *inaccuracy* (where the system does not balance replica load effectively). DONAR must avoid falling into these traps.

4.1.3 Research Contributions and Roadmap

In designing, implementing, deploying, and evaluating DONAR, we make three main research contributions:

Simple and expressive interface for customer policies (Section 4.2): DONAR has a simple policy interface where each replica location specifies a *split weight* or a *bandwidth cap*, and expected client performance is captured through a *performance penalty*. These three sets of parameters may change over time. We show that this interface can capture diverse replica-selection goals based on client proximity, server capacity, 95th-percentile billing, and so on. The API leads us to introduce a formal optimization problem that jointly consider customers' (sometimes conflicting) preferences.

Stable, efficient, and accurate distributed replica-selection algorithm (Section 4.3): DONAR consists of a distributed collection of mapping nodes for better reliability and performance. Since these nodes each handle replica selection for a different mix of clients, they cannot simply solve the optimization problem independently. Rather than resorting to a centralized architecture that pushes results to each mapping node, we decompose the optimization problem into a distributed algorithm that requires only minimal coordination between the nodes. We show that our decentralized algorithm provably converges to the solution of the optimization problem, ensuring that DONAR does not over or under react to shifts in client demands.

Scalable, secure, reliable, and flexible prototype system (Section 4.4): Our DONAR prototype implements the distributed optimization algorithm, supports DNS- and HTTP-based replica selection, and stores customer data in a scalable storage system. DONAR uses IP anycast for fast failover and good client redirection performance. The prototype includes a secure update protocol for policy changes and receives a periodic feed of IP2Geo data [9] to support policies based on client location. Our prototype is used to provide distributed DNS resolution for the Measurement Lab testbed [84] and for a portion of the CoralCDN service [85].

Experiments in Section 4.5 evaluate both our distributed algorithm operating at scale (through trace-driven simulations of client requests to CoralCDN) and a small-scale deployment of our prototype system (providing DNS-based replica selection to jointly optimize client proximity and server load for part of CoralCDN). These experiments demonstrate that DONAR offers effective, customized replica selection in a scalable and efficient fashion. Section 4.6 compares DONAR to related work, and Section 4.7 concludes.

4.2 Configurable Mapping Policies

DONAR realizes its customers' high-level policy goals, while shielding them from the complicated internals of distributed replica selection. By *customer*, we mean a service provider that outsources replica selection to DONAR. Customers configure their policies by communicating directly with any DONAR mapping node. This section first motivates and introduces DONAR's policy API. It then describes a number of application scenarios that leverage this interface to express sophisticated mapping policies.

4.2.1 Customer Goals

Customers use DONAR to optimally pair clients with service replicas. What customers consider "optimal" can differ: some may seek simply to minimize the network latency between clients and replicas, others may seek to balance load across all replicas, while still others may try to optimize the assignment based on the billing costs of their network operators or hosting services. In general, however, we can decompose a customer's preferences into those associated with the *wide-area network* (the network performance a client would experience if directed to a particular replica) and those associated with *its own replicas* (the load on the servers and the network at each location). DONAR considers *both* factors in its replica-selection algorithm.

Minimizing network costs. Mapping policies commonly seek to pair clients with replicas that offer good performance. While replica load can affect client-perceived performance, the network path has a significant impact as well. For example, web transfers or interactive applications often seek small network round-trip times (RTTs), while bulk transfers seek good network throughput (although RTT certainly can also impact TCP throughput).

DONAR's algorithms use an abstract $cost(c, i)$ function to indicate the performance penalty between a particular client–replica pair. This function allows us a considerable amount of expressiveness. For instance, to optimize latency, *cost* simply can be RTT, directly measured or estimated via network coordinates. If throughput is the primary goal, *cost* can be calculated as the penalty of network congestion. In fact, the *cost* function can be any shape—*e.g.*, a translated logistic function of latency or congestion—to optimize the worst case or percentile-based performance. The flexibility of the *cost* function also allows for intricate policies, *e.g.*, always mapping one client to a particular server replica, or preferring a replica through a peering link over a transit link.

DONAR's current implementation starts with a shared *cost* function for all of its customers, which represents expected path latency. As a result, our customers do not need to calculate and submit large and complex cost functions independently. In the case where a customer's notion of cost differs from that shared function, our interface allows them to override DONAR's default mapping decisions (discussed in more detail in the following section).

To estimate path latency, services like DONAR could use a variety of techniques: direct network measurements [86, 6, 87], virtual coordinates [88, 89], structural models of path performance [90, 11], or some hybrid of these approaches [82]. DONAR's algorithm can accept any of these techniques; research towards improving network cost estimation is very complementary to our interests. Our current prototype uses a commercial IP geolocation database [9] to estimate network proximity, although this easily could be replaced with an alternative solution.

Balancing client requests across replicas. Unlike pairwise network performance, which is largely shared amongst DONAR's customers, traffic distribution preferences vary widely from customer to customer. In the simplest scenarios, services may want to equally balance request rates across replicas, or they may want decisions based solely on network proximity (*i.e.*, *cost*). More advanced considerations, however, are both possible and important. For example, given 95th-percentile billing mechanisms, customers could try to minimize costs by reducing the frequency of peak consumption, or to eliminate overage costs by rarely exceeding their committed rates. In any replica-mapping scheme, request capacity, bandwidth cost, and other factors will influence the preferred rate of request arrival at a given replica. The relative importance of these factors may vary from one replica to the next, even for the same customer.

4.2.2 Application Programming Interface

Through considering a number of such policy preferences—which we review in the next section—we found that enabling customers to express two simple factors was a powerful tool. DONAR allows its customers to dictate a replica's (i) *split weight*, w_i, the desired proportion of requests that a particular replica i should receive out of the customer's total set of replicas, or (ii) *bandwidth cap*, B_i, the upper-bound on the exact number of requests that replica i should receive. In practice, different requests consume different amounts of server and bandwidth resources; we expect customers to set B_i based on the relationship between the available server/bandwidth resources and the average resources required to service a request.

These two factors enable DONAR's customers to balance load between replicas or to cap load at an individual replica. For the former, a customer specifies w_i and ϵ_i to indicate that it wishes replica i to receive a w_i fraction of the total request load[1], but is willing to deviate up to ϵ_i in order to achieve better network performance. If P_i denotes the true proportion of requests directed to i, then

$$|P_i - w_i| \leq \epsilon_i$$

This ϵ_i is expressed in the same unit as the split rate; for example, if a customer wants each of its ten replicas to receive a split rate of 0.1, setting ϵ_* to 0.02 indicates that each replica should receive 10% $\pm$ 2% of the request load.

Alternatively, a customer can also specify a bandwidth cap B_i per replica, to require that the exact amount of received traffic at i does not exceed B_i. If B is the total constant load across all replicas, then

$$B \cdot P_i \leq B_i$$

[1]In reality, the customer may select weights that do not sum to 1, particularly since each replica may assign its weight independently. DONAR simply *normalizes* the weights to sum to 1, *e.g.*, proportion $w_i / \sum_j w_j$ for replica i.

Functionality	DONAR API Call
create a DONAR service	`s = create ()`
add a replica instance	`i = add (s, repl, ttl)`
set split weight	set (s, i, w_i, ϵ_i)
set bandwidth cap	set (s, i, B_i)
match a client-replica pair	`match (s, clnt, i)`
prefer a particular replica	`preference (s, clnt, i)`
remove a replica instance	`remove (s, i)`

Figure 4.2: DONAR's Application Programming Interface

Note that if a set of B_i's become infeasible given a service's traffic load, DONAR reverts to w_i splits for each instance. If those are not present, excess load is spread equally amongst replicas.

Figure 4.2 shows DONAR's customer API. A customer creates a DONAR service by calling `create()`. A customer uses `add()` and `remove()` to add or remove a replica instance from its service's replica set. The record will persist in DONAR for the specified time-to-live period (*ttl*), unless it is explicitly removed before this period expires. A short *ttl* serves to make `add()` equivalent to a soft-state heartbeat operation, where an individual replica can execute it repeatedly to express its liveness. To communicate its preferred request distribution, the customer uses `set()` for a replica instance i. This function takes either (w_i, ϵ_i) or B_i, but never both (as the combination does not make logical sense with respect to the *same* replica at the same instant in time). We do allow a customer simultaneously to use both w_i and B_i for different subsets of replicas, which we show later by example.

Some customers may want to impose more explicit constraints on specific client-replica pairs, for cost or performance reasons. For example, a customer may install a single replica inside an enterprise network for handling requests *only* from clients within that enterprise (and, similarly, the clients in the enterprise should *only* go to that server replica, no matter what the load is). A customer expresses this hard constraint using the `match()` call. Alternatively, a customer may have a strong preference for directing certain clients to a particular replica—*e.g.*, due to network peering arrangements—giving them priority over other clients. A customer expresses this policy using the `preference()` call. In contrast to `match()`, the `preference()` call is a soft constraint; in the (presumably rare) case that replica i cannot handle the total load of the high-priority clients, some of these client requests are directed to other, less-preferred replicas. These two options mimic common primitives in today's commercial CDNs.

In describing the API, we have not specified exactly who or what is initiating these API function calls—*e.g.*, a centralized service manager, individual replicas, etc.—because DONAR imposes no constraint on the source of customer policy preferences. As we demonstrate next, different use cases require updating DONAR from different sources, at different times, and based on different inputs.

4.2.3 Expressing Policies with DONAR's API

Customers can use this relatively simple interface to implement surprisingly complex mapping policies. It is important to note that, internally, DONAR will minimize the network cost between clients and replicas within the constraints of any customer policy. To demonstrate use of the API, we describe increasingly intricate scenarios.

Selecting the closest replica. To implement a "closest replica" policy, a customer simply sets the bandwidth caps of all replicas to infinity. That is, they impose no constraint on the traffic distribution. This update can be generated from a single source or independently from each replica.

Balancing workload between datacenters. Consider a service provider running multiple datacenters, where datacenter i has w_i servers (dedicated to this service), and jobs are primarily constrained by server resources. The service provider's total capacity at these datacenters may change over time, however, due to physical failures or maintenance, the addition of new servers or VM instances, the repurposing of resources amongst services, or temporarily bringing down servers for energy conservation. In any case, they want the proportion of traffic arriving at each datacenter to reflect current capacity. Therefore, each datacenter i locally keeps track of its number of active servers as w_i, and calls `set(`s, i, w_i, ϵ_i`)`, where ϵ_i can be any tolerance parameter. The datacenter need only call `set()` when the number of active servers changes. DONAR then proportionally maps client requests according to these weights. This scenario is simple because (i) w_i is locally measurable within each datacenter, and (ii) w_i is easily computable, *i.e.*, decreasing w_i when servers fail and increasing when they recover.

	Multi-Homing	**Replicated Service**
Available Resources to Optimize	Local links with different bandwidth capacities and costs	Wide-area replicas with different bandwidth capacities and costs
Resource Allocation	Proportion of traffic to assign each link	Proportion of new clients to assign each replica

Figure 4.3: Multi-homed route control versus wide-area replica selection

Enforcing heuristics-based replica selection. DONAR implements a superset of the policies that are used in legacy systems to achieve heuristics-based replica selection. For example, OASIS [82] allows replicas to withdraw themselves from the server pool once they reach a given self-assessed load threshold, and then maps clients to the closest replica remaining in the pool. To realize this policy with DONAR, each replica i independently calls `set`(s, i, B_i), where B_i is the load threshold it estimates from previous history. B_i can be dynamically updated by each replica, by taking into account the prior observed performance given its traffic volume. DONAR then strictly optimizes for network cost among those replicas still under their workload threshold B_i.

Optimizing 95th-percentile billing costs. Network transit providers often charge based on the 95th-percentile bandwidth consumption rates, as calculated over all 5-minute or 30-minute periods in a month. To minimize such "burstable billing" costs, a distributed service could leverage an algorithm recently proposed for multi-homed route control under 95-percentile billing [91]. This algorithm, while intended for traffic engineering in multi-homed enterprise networks, could also be used to divide clients amongst service replicas; Figure 4.3 summarizes the relationship between these two problems. The billing optimization could be performed by whatever management system the service already runs to monitor its replicas. This system would need to track each replica's traffic load, predict future traffic patterns, run a billing cost optimization algorithm to compute the target split ratio w_i for the next time period, and output these parameters to DONAR through `set`$(s, i, w_i, 0)$ calls. Of course, any mapping service capable of proportionally splitting requests (such as weighted round-robin DNS) could achieve such dynamic splits, but only by ignoring client network performance. DONAR accommodates frequently updated split ratios while still assuring that clients reach a nearby replica.

Notation	Interpretation
$\mathcal{N}$	Set of mapping nodes
$\mathcal{C}_n$	Set of clients for node n, $\mathcal{C}_n \subseteq \mathcal{C}$, the set of all clients
$\mathcal{I}$	Set of service replicas
R_{nci}	Proportion of traffic load that is mapped to replica i from client c by node n
α_{cn}	Proportion of node n's traffic load from client c
s_n	Proportion of total traffic load (from $\mathcal{C}$) on node n
w_i	Traffic split weight of replica i
P_i	True proportion of requests directed to replica i
B_i	Bandwidth cap on replica i
B	Total traffic load across all replicas
ϵ_i	Tolerance of deviation from desired traffic split

Table 4.1: Summary of key notations

Shifting traffic to under-utilized replicas. Content providers often deploy services over a large set of geographically diverse replicas. Providers commonly want to map clients to the closest replica unless pre-determined bandwidth capacities are violated. Under such a policy, certain nodes may see too *few* requests due to an unattractive network location (*i.e.*, they are rarely the "closest node"). To remedy this problem, providers can leverage both traffic options in combination. For busy replicas, they `set(`s, i, B_i`)` to impose a bandwidth cap, while for unpopular replicas, they can `set(`$s, i, k, 0$`)`, so that replica i receives at least some fixed traffic proportion k. In this way, they avoid under-utilizing instances, while offering the vast majority of clients a nearby replica.

4.3 Replica Selection Algorithms

This section first formulates the global replica-selection problem, based on the mapping policies we considered earlier. We then propose a decentralized solution running on distributed mapping nodes. We demonstrate that each mapping node—locally optimizing the assignment of its client population and judiciously communicating with other nodes—can lead to a globally optimal assignment. We summarize the notation we use in Table 4.1.

4.3.1 Global Replica-Selection Problem

We have discussed two important policy factors that motivate our replica-selection decisions. Satisfying one of these components (*e.g.*, network performance), however, typically comes at the expense of the other (*e.g.*, accurate load distribution). As DONAR allows customers to freely express their requirements—through their choice of w_i, ϵ_i, and B_i—customers can express their willingness to trade-off performance for load distribution.

To formulate the replica-selection problem, we introduce a network model. Let $\mathcal{C}$ be the set of clients, and $\mathcal{I}$ the set of server replicas. Denote R_{ci} as the proportion of traffic from client c routed to replica i, which we solve for in our problem. The global client performance (penalty) can be calculated as

$$perf^g = \sum_{c \in \mathcal{C}} \sum_{i \in \mathcal{I}} R_{ci} \cdot cost(c, i) \tag{4.1}$$

Following our earlier definitions, let $P_i = \sum_{c \in \mathcal{C}} R_{ci}$ be the true proportion of requests directed to replica i. Our goal is to minimize this performance penalty, *i.e.*, to match each client with a good replica, while satisfying the customer's requirements. That goal can be expressed as the following optimization problem $\mathbf{RS^g}$:

$$\begin{aligned} \text{minimize} \quad & perf^g && \text{(4.2a)} \\ \text{subject to} \quad & |P_i - w_i| \leq \epsilon_i && \text{(4.2b)} \\ & B \cdot P_i \leq B_i, \forall i && \text{(4.2c)} \end{aligned}$$

B is the total amount of traffic, a constant parameter that can be calculated by summing the traffic observed at all replicas. Note that for each replica i, either constraint (4.2b) or (4.2c) is active, but not both.

The optimization problem can also handle the `match()` and `preference()` constraints outlined in Section 4.2.2. A call to `match()` imposes a *hard* constraint that client c only uses replica i, and vice versa. This is easily handled by removing the client and the replica from the optimization problem entirely, and solving the problem for the remaining clients and replicas. The `preference()` call imposes a *soft* constraint that client c has priority for mapping to replica i, assuming the replica can handle the load. This is easily handled by scaling down $cost(c, i)$ by some large constant factor so the solution maps client c to replica i whenever possible.

4.3.2 Distributed Mapping Service

Solving the global optimization problem directly requires a central coordinator that collects all client information, calculates the optimal mappings, and directs clients to the appropriate replicas. Instead, we solve the replica-selection problem using a *distributed* mapping service. Let $\mathcal{N}$ be the set of mapping nodes. Every mapping node $n \in \mathcal{N}$ has its own view $\mathcal{C}_n$ of the total client population, *i.e.*, $\mathcal{C}_n \subseteq \mathcal{C}$. A mapping node n receives a request from a client $c \in \mathcal{C}_n$. The node maps the client to a replica $i \in \mathcal{I}$ and returns the result to that client. In practice, each client c can represent a group of aggregated end hosts, *e.g.*, according to their postal codes. This is necessary to keep request rates per-client stable (see Section 4.5.2 for more discussion). Therefore, we allow freedom in directing one client to one or multiple replicas. We then expand the decision variable R_{ci} to R_{nci}, which is the fraction of traffic load that is mapped to replica i from client c by node n, *i.e.*, $\sum_i R_{nci} = 1$.

Each DONAR node monitors requests from its own client population. Let s_n be the normalized traffic load on node n (that is, n's fraction of the total load from $\mathcal{C}$). Different clients may generate different amounts of workload; let $\alpha_{cn} \in [0, 1]$ denote the proportion of n's traffic that comes from client c (where $\sum_{c \in \mathcal{C}_n} \alpha_{cn} = 1$). The information of R_{nci} and α_{cn} can be measured at DONAR nodes locally, whereas collecting s_n and P_i requires either central aggregation of each node's local client load and decisions, or each node to exchange its load with its peers. The distributed node deployment also allows DONAR to check the feasibility of problem (4.2a) easily: given local load information, calculate the total load B and determine whether (4.2a) accepts a set of $\{w_i, B_i\}_{i \in \mathcal{I}}$ parameters.

The global replica-selection problem $\mathbf{RS^g}$, after introducing mapping nodes and the new variable R_{nci}, remains the same formulation as (4.2a)-(4.2c), with

$$\begin{aligned} perf^g &= \sum_{n \in \mathcal{N}} s_n \sum_{c \in \mathcal{C}_n} \alpha_{cn} \sum_{i \in \mathcal{I}} R_{nci} \cdot cost(c, i) \qquad (4.3) \\ P_i &= \sum_{n \in \mathcal{N}} s_n \sum_{c \in \mathcal{C}_n} \alpha_{cn} \cdot R_{nci} \end{aligned}$$

A simple approach to solving this problem is to deploy a central coordinator that collects all necessary information and calculates the decision variables for *all* mapping nodes. We seek to avoid this centralization, however, for several reasons: (i) coordination between all mapping nodes is required; (ii) the central coordinator becomes a single point-of-failure; (iii) the coordinator requires information about every client and node, leading to $O(|\mathcal{N}| \cdot |\mathcal{C}| \cdot |\mathcal{I}|)$ communication and computational complexity; and (iv) as traffic load changes, the problem needs to be recomputed and the results re-disseminated. This motivates us to develop a decentralized solution, where each DONAR node runs a distributed algorithm that is simultaneously scalable, accurate, and responsive to change.

4.3.3 Decentralized Selection Algorithm

We now derive a decentralized solution for the global problem $\mathbf{RS^g}$. Each DONAR node will perform a smaller-scale local optimization based on its client view. We later show how local decisions converge to the global optimum within a handful of algorithmic iterations. We leverage the theory of optimization decomposition to guide our design. Consider the global performance term $perf^g$ (4.3), which consists of local client performance contributed by each node:

$$perf^g = \sum_{n \in \mathcal{N}} perf^l_n$$

where

$$perf^l_n = s_n \sum_{c \in \mathcal{C}_n} \alpha_{cn} \sum_{i \in \mathcal{I}} R_{nci} \cdot cost(c, i) \tag{4.4}$$

Each node optimizes local performance on its client population, plus a load term imposed on the replicas. For a replica i with a split-weight constraint (the case of bandwidth-cap constraint follows similarly and is shown in the final algorithm),

$$load = \sum_{i \in \mathcal{I}} \lambda_i \big((P_i - w_i)^2 - \epsilon_i^2\big) \tag{4.5}$$

where λ_i is interpreted as the unit price of violating the constraint. We will show later how to set this value dynamically for each replica. Notice that the load associates with decision variables from all nodes. To decouple it, rewrite P_i as

$$P_i = \sum_{n \in \mathcal{N}} P_{ni} = P_{ni} + \sum_{n' \in \mathcal{N} \setminus \{n\}} P_{n'i} = P_{ni} + P_{-ni}$$

```
Initialization
  For each replica $i$: Set an arbitrary price $\lambda_i \geq 0$.
  For each node $n$: Set an arbitrary decision $R_{nci}$.

Iteration
  For each node $n$:
  (1)  Collects the latest $\{P_{n'i}\}_{i\in\mathcal{I}}$ for other $n'$.
  (2)  Collects the latest $\lambda_i$ for every replica $i$.
  (3)  Solves $\mathbf{RS_n^l}$.
  (4)  Computes $\{P_{ni}\}_{i\in\mathcal{I}}$ and updates the info.
  (5)  With probability $1/|\mathcal{N}|$, for every replica $i$:
  (6)      Collects the latest $P_i = \sum_n P_{ni}$.
  (7)      Computes $\lambda_i \leftarrow \max\{0, \lambda_i + \theta((P_i - w_i)^2 - \epsilon_i^2)\}$,
            or $\lambda_i \leftarrow \max\{0, \lambda_i + \theta((B \cdot P_i)^2 - B_i^2)\}$.
  (8)      Updates $\lambda_i$.
  (9)  Stops if $\{P_{n'i}\}_{i\in\mathcal{I}}$ from other $n'$ do not change.
```

Table 4.2: Decentralized solution of server selection

where $P_{ni} = s_n \sum_{c\in\mathcal{C}_n} \alpha_{cn} \cdot R_{nci}$ is the traffic load contributed by node n on replica i—that is, the requests from those clients directed to i by DONAR node n—and P_{-ni} is the traffic load contributed by nodes other than n, independent of n's decisions.

Then the local replica selection problem for node n is formulated as the following optimization problem $\mathbf{RS_n^l}$:

$$\text{minimize} \quad perf_n^l + load_n \tag{4.6a}$$

$$\text{variables} \quad R_{nci}, \forall c \in \mathcal{C}_n, i \in \mathcal{I} \tag{4.6b}$$

where $load_n = load, \forall n$. To solve this local problem, a mapping node needs to know (i) local information about s_n and α_{cn}, (ii) prices λ_i for all replicas, and (iii) the aggregated P_{-ni} information from other nodes. Equation (4.6a) is a quadratic programming problem, which can be solved efficiently by standard optimization solvers (we evaluate computation time in Section 4.5.1).

We formally present the decentralized algorithm in Table 4.2. At the initialization stage, each node picks an arbitrary mapping decision, *e.g.*, one that only optimizes local performance. Each replica sets an arbitrary price, say $\lambda_i = 0$. The core components of the algorithm are the local updates by each mapping node, and the periodic updates of replica prices. Mapping decisions are made at each node n by solving $\mathbf{RS^l_n}$ based on the latest information. Replica prices (λ_i) are updated based on the inferred traffic load to each i. Intuitively, the price increases if i's split weight or bandwidth requirement is violated, and decreases otherwise. This can be achieved via additive updates (shown by θ). We both collect and update the P_{ni} and λ_i information through a data store service, as discussed later.

While the centralized solution requires $O(|\mathcal{N}| \cdot |\mathcal{C}| \cdot |\mathcal{I}|)$ communication and computation at the coordinator, the distributed solution has much less overhead. Each node needs to share its mapping decisions of size $|\mathcal{I}|$ with all others, and each replica's price λ_i needs to be known by each node. This implies $|\mathcal{N}|$ messages, each of size $O((|\mathcal{N}|-1) \cdot |\mathcal{I}| + |\mathcal{I}|) = O(|\mathcal{N}| \cdot |\mathcal{I}|)$. Each node's computational complexity is of size $O(|\mathcal{C}_n| \cdot |\mathcal{I}|)$.

The correctness of the decentralized algorithm is given by the following theorem:

Theorem 18. *The distributed algorithm shown in Table 4.2, converges to the optimal solution of* $\mathbf{RS^g}$*, given that (i) each node n iteratively solves* $\mathbf{RS^l_n}$ *in a circular fashion,* i.e., $n = 1, 2, \ldots |\mathcal{N}|, 1, \ldots,$ *and (ii) each replica price λ_i is updated in a larger timescale,* i.e., *after all nodes' decisions converge given a set of* $\{\lambda_i\}_{i\in\mathcal{I}}$.

Proof. It suffices to show that the distributed algorithm is an execution of the dual algorithm that solves $\mathbf{RS^g}$. We only show the case of split weight constraint (4.2b), and the case of bandwidth cap constraint (4.2c) would follow similarly.

First, following the Lagrangian method for solving an optimization problem, we derive the Lagrangian of the global problem $\mathbf{RS^g}$. Constraint (4.2b) is equivalent to

$$(P_i - w_i)^2 \leq \epsilon_i^2$$

The Lagrangian of $\mathbf{RS^g}$ is written as:

$$\begin{aligned} L(R, \boldsymbol{\lambda}) &= perf^g + \sum_{i\in\mathcal{I}} \lambda_i \big((P_i - w_i)^2 - \epsilon_i^2\big) \\ &= \sum_{n\in\mathcal{N}} perf^l_n + load \end{aligned} \tag{4.7}$$

where $\lambda_i \geq 0$ is the Lagrange multiplier (replica price) associated with the split weight constraint on replica i, and $R = \{R_{nci}\}_{n\in\mathcal{N}, c\in\mathcal{C}_n, i\in\mathcal{I}}$ is the primal variable. The dual algorithm requires to minimize the Lagrangian (4.7) for a given set of $\{\lambda_i\}_{i\in\mathcal{I}}$.

$$\begin{array}{rl} \text{minimize} & L(R,\lambda) \\ \text{variable} & R \end{array}$$

A distributed algorithm implied by condition (i) solves (4.7), because each node n iteratively solving $\mathbf{RS^l_n}$ in a circular fashion, simply implements the nonlinear Guass-Seidel algorithm (per [32, Ch. 3, Prop. 3.9], [92, Prop. 2.7.1]):

$$R_n^{(t+1)} = \operatorname{argmin} \mathbf{RS^g}(\ldots, R_{n-1}^{(t+1)}, R_n, R_{n+1}^{(t)}, \ldots) \tag{4.8}$$

where $R_n = \{R_{nci}\}_{c\in\mathcal{C}_n, i\in\mathcal{I}}$ denotes node n's decision variable. Given a set of $\{\lambda_i\}_{i\in\mathcal{I}}$, the distributed solution converges because: first, the objective function (4.7) is continuously differentiable and convex on the entire set of variables. Second, each step of $\mathbf{RS^l_n}$ is a minimization of (4.7) with respect to its own variable R_n, assuming others are held constant. Third, the optimal solution of each $\mathbf{RS^l_n}$ is uniquely attained, since its objective function is quadratic. The three conditions together ensure that the limit point of the sequence $\{R_n\}_{n\in\mathcal{N}}^{(t)}$, minimizes (4.7) for a given set of $\{\lambda_i\}_{i\in\mathcal{I}}$.

Second, we need to solve the master dual problem:

$$\begin{array}{rl} \text{maximize} & f(\boldsymbol{\lambda}) \\ \text{subject to} & \boldsymbol{\lambda} \geq 0 \end{array}$$

where $f(\boldsymbol{\lambda}) = \max_R L(R, \boldsymbol{\lambda})$, which is solved in the first step. Since the solution to (4.7) is unique, the dual function $f(\boldsymbol{\lambda})$ is differentiable, which can be solved by the following gradient projection method:

$$\lambda_i \leftarrow \max\left\{0, \lambda_i + \theta\big((P_i - w_i)^2 - \epsilon_i^2\big)\right\},\ \forall i$$

where $\theta > 0$ is a small positive step size. Condition (ii) guarantees the dual prices $\boldsymbol{\lambda}$ are updated in a larger timescale, as the dual algorithm requires.

The duality gap of $\mathbf{RS^g}$ is zero, and the solution to each $\mathbf{RS_n^l}$ is also unique. This finally guarantees that the equilibrium point of the decentralized algorithm is also the optimal solution of the global problem $\mathbf{RS^g}$. $\square$

The correctness of the distributed algorithm relies on an appropriate ordering of local updates from each node, *i.e.*, in a round-robin fashion as shown in (4.8), and a less frequent replica price update. In practice, however, we allow nodes and replicas to update *uncoordinatedly* and *independently.* We find that the algorithm's convergence is not sensitive to this lack of coordination, which we demonstrate in our evaluation. In fact, the decentralized solution works well even at the scale of thousands of mapping nodes. For a given replica-selection problem, the decentralized solution usually converges within a handful of iterations, and the equilibrium point is also the optimal solution to the global problem.

4.4 DONAR's System Design

This section describes the design of DONAR, which provides distributed replica selection for large numbers of customers, each of whom have their own set of service replicas and different high-level preferences over replica selection criteria. DONAR implements the policy interface and distributed optimization mechanism we defined in the last two sections. Each DONAR node must also reliably handle client requests and customer updates. DONAR nodes should be geographically dispersed themselves, for greater reliability and better performance. Our current deployment, for example, consists of globally dispersed machines on both the PlanetLab [93] and VINI [94] network platforms.

Figure 4.4 depicts a single DONAR node and its interactions with various system components. This section is organized around these components. Section 4.4.1 discusses how DONAR nodes combine customer policies, mapping information shared by other nodes, and locally-available cost information, in order to optimally map clients to customers' service replicas. Section 4.4.2 describes DONAR's front-end mechanisms for performing replica selection (*e.g.*, DNS, HTTP redirection, HTTP proxying, etc.), and Section 4.4.3 details its update protocol for registering and updating customer services, replicas, and policies. Finally, Section 4.4.4 describes DONAR's back-end

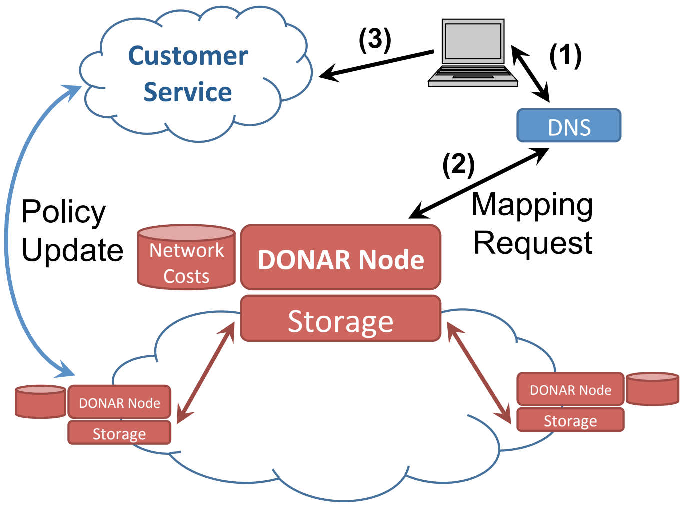


Figure 4.4: Interactions on a DONAR node

distributed data store (for reliably disseminating data) and use of IP Anycast (for reliably routing client requests).

4.4.1 Efficient Distributed Optimization

Given policy preferences for each customer, DONAR nodes must translate high-level mapping goals into a specific set of rules for each node. DONAR's policy engine realizes a variant of the algorithm described in Table 4.2. Particularly, all nodes act asynchronously in the system, so no round-based coordination is required. We demonstrate in Section 4.5 that this variant converges in practice.

Request rate and cost estimation. Our model assumes that each node has an estimate of the request rate per client. As suggested in Section 4.3, each "client" represents a group of similarly located end-hosts (we also refer to this entity as a "client region"). When a DONAR node comes online and begins receiving client requests, it tracks the request volume per unit time per client region. While our optimization algorithm models a static problem—and therefore constant request rates— true request rates vary over time. To address this limitation, DONAR nodes use an exponentially-weighted moving average of previous time intervals (with $\alpha = 0.8$ and 10 minute intervals in our current deployment). Still, rapid changes in a particular client region might lead to suboptimal mapping decisions. Using trace data from a popular CDN, however, we show in Section 4.5.2 that relative request rates per region do not significantly vary between time intervals.

Our model also assumes a known *cost(c,i)* function, which quantifies the cost of pairing client c with instance i. In our current deployment, each DONAR node has a commercial-grade IP geolocation database [9] that provides this data and is updated weekly. We use a *cost(c, i)* function that is the normalized Euclidean distance between the IP prefixes of the client c and instance i (like that described in Section 4.3).

Performing local optimization. DONAR nodes arrive at globally optimal routing behavior by periodically re-running the local optimization problem. In our current implementation, nodes each run the same local procedure at regular intervals (about every 2 minutes, using some randomized delay to desynchronize computations).

As discussed in Section 4.3.3, the inputs to each local optimization are the aggregate traffic information sent by all remote DONAR nodes $\{B \cdot P_{-ni}\}_{i \in \mathcal{I}}$; the customer policy parameters, $\{w_i, B_i\}_{i \in \mathcal{I}}$; and the proportions of local traffic coming from each client region to that node, $\{\alpha_{cn}\}_{c \in \mathcal{C}_n}$. The first two inputs are available through DONAR's distributed data store, while nodes locally compute their clients' proportions. Given these inputs, the node must decide how to best map its clients. This reduces to the local optimization problem $\mathbf{RS_n^l}$, which minimizes a node's total client latency given the constraints of the customer's policy specification and the mapping decisions of other nodes (treating their most recent update as a static assignment). The outcome of this optimization is a new set of local rules $\{R_{nci}\}_{c \in \mathcal{C}_n i \in \mathcal{I}}$, which dictates the node's mapping behavior. Given these new mapping rules, the node now expects to route different amounts of traffic to each replica. It then computes these new expected traffic rates per instance, using the current mapping policy and its historical client request rates $\{\alpha_{cn}\}_{c \in \mathcal{C}_n}$. It then updates its existing per-replica totals $\{B \cdot P_i\}_{i \in \mathcal{I}}$ in the distributed data store, so that implications of its new local policy propagate to other nodes.

If the new solution violates customer constraints—bandwidth caps are overshot or split's exceed the allowed tolerance—the node will update the constraint multipliers $\{\lambda_i\}_{i \in \mathcal{I}}$ with probability of $1/|\mathcal{N}|$. Thus, in the case of overload, the multipliers will be updated, on average, once per cycle of local updates.

4.4.2 Providing Flexible Mapping Mechanisms

A variety of protocol-level mechanisms are employed for wide-area replica selection today. They include (i) dynamically generated DNS responses with short TTLs, according to a given policy, (ii) using HTTP Redirection from a centralized source and/or between replicas, and (iii) using persistent HTTP proxies to tunnel requests.

To offer customers maximum flexibility, DONAR offers all three of these mechanisms. To use DONAR via DNS, a domain's owner will register each of its replicas as a single `A` record with DONAR, and then point the `NS` records for its domain (*e.g.*, `example.com`) to `ns.donardns.org`. DONAR nameservers will then respond to requests for `example.com` with an appropriate replica given the domain's selection criteria.

To use DONAR for HTTP redirection or proxying, a customer adds HTTP records to DONAR—a record type in DONAR update messages—such as mapping `example.com` to `us-east.example.com`, `us-west.example.com`, etc. DONAR resolves these names and appropriately identifies their IP address for use during its optimization calculations.[2] This customer then hands off DNS authority to DONAR as before. When DONAR receives a DNS query for the domain, it returns the IP address of the client's nearest DONAR node. Upon receiving the corresponding HTTP request, a DONAR HTTP server uses requests' `Host:` header fields to determine for which customer domains their requests correspond. It queries its local DONAR policy engine for the appropriate replica, and then redirects or proxies the request to that replica.

The DONAR software architecture is designed to support the easy addition of new protocols. DONAR's DNS nameserver and HTTP server run as separate processes, communicating with the local DONAR policy engine via a standardized socket protocol. Section 4.4.5 describes additional implementation details.

4.4.3 Secure Registration and Dynamic Updates

Since DONAR aims to accommodate many simultaneous customers, it is essential that all customer-facing operations be completely automated and not require human intervention. Additionally, since DONAR is a public service, it must authenticate client requests and prevent replay attacks. To

[2] In this HTTP example, customers need to ensure that each name resolves either to a single IP address or a set of collocated replicas.

meet these goals we have developed a protocol which provides secure, automatic account creation and facilitates frequent policy updates.

Account creation. In DONAR, a customer account is uniquely identified by a private/public key pair. DONAR reliably stores its customers' public keys, which are used to cryptographically verify signatures on account updates (described next). Creating accounts in DONAR is completely automated, *i.e.*, no central authority is required to approve account creation. To create a DONAR account, a customer generates a public/private key-pair and simply begins adding new records to DONAR, signed with the private key. If a DONAR node sees an unregistered public key in update messages, it generates the SHA-1 hash of the key, `hash`, and allocates the domain `<hash>.donardns.net` to the customer.

Customers have the option of validating a domain name that they own (*e.g.*, `example.com`). To do so, a customer creates a temporary CNAME DNS record that maps `validate-<hash> .example.com` to `donardns.net`. Since only someone authoritative for the example.com namespace will be able to add this record, its presence alone is a sufficient proof of ownership. The customer then sends a validation request for `example.com` to a DONAR node. DONAR looks for the validation CNAME record in DNS and, if found, will replace `<hash>.donardns.net` with `example.com` for all records tied to that account.

DONAR Update Protocol (DUP). Customers interact with DONAR nodes through the DONAR Update Protocol (DUP). Operating over UDP, DUP allows customers to add and remove new service replicas, express the policy parameters for these replicas as described in Section 4.3, as well as implicitly create and explicitly verify accounts, per above. DUP is similar in spirit to the DNS UPDATE protocol [95] with some important additional features. These include mandatory RSA signatures, nonces for replay protection, DONAR-specific meta-data (such as *split weight* or *bandwidth cap*), and record types outside of DNS (for HTTP mapping). DUP is record-based (like DNS), allowing forward compatibility as we add new features such as additional policy options.

4.4.4 Reliability through Decentralization

DONAR provides high availability by gracefully tolerating the failure of individual DONAR nodes. To accomplish this, DONAR incorporates reliable distributed data storage (for customer records) and ensures that clients will be routed away from failed nodes.

Distributed Data Storage. DONAR provides distributed storage of customer record data. A customer update (through DUP) should be able to be received and handled by *any* DONAR node, and the update should then be promptly visible throughout the DONAR network. There should be no central point of failure in the storage system, and the system should scale-out with the inclusion of new DONAR nodes without any special configuration.

To provide this functionality, DONAR uses the CRAQ storage system [96] to replicate record data and account information across all participating nodes. CRAQ automatically re-replicates data under membership changes and can provide either strong or eventual consistency of data. Its performance is optimized for read-heavy workloads, which we expect in systems like DONAR where the number of client requests likely will be orders of magnitude greater than the number of customer updates. DONAR piggybacks on CRAQ's group-membership functionality, built on Zookeeper [97], in order to alert DONAR nodes when a node fails. While such group notifications are not required for DONAR's availability, this feature allows DONAR to quickly recompute and reconverge to optimal mapping behavior following node failures.

Route control with IP anycast. While DONAR's storage system ensures that valuable data is retained in the face of node failures, it does not address the issue of routing client requests away from failed nodes. When DONAR is used for DNS, it can partially rely on its clients' resolvers performing automatic failover between authoritative nameservers. This failover significantly increases resolution latency, however. Furthermore, DONAR node failure presents an additional problem when used with HTTP-based selection. Most web browsers do not failover between multiple A records (in this case, DONAR HTTP servers), and browsers—and now browser plugins like Java and Flash as well—purposely "pin" DNS names to specific IP addresses to prevent "DNS rebinding" attacks [98].[3] These cached host-to-address bindings often persist for several minutes. To address both cases, DONAR is designed to work over IP anycast, not only for answering DNS queries but also for processing updates.

In our current deployment, a subset of DONAR nodes run on VINI [94], a private instance of PlanetLab which allows tighter control over the network stack. These nodes run an instance of Quagga that peers with TransitPortal instances at each site [99], and thus each site announces

[3] A browser may pin DNS–IP mappings even after observing destination failures; otherwise, an attacker may forge ICMP "host unreachable" messages to cause it to unpin a specific mapping.

DONAR's /24 prefix through BGP. If a node loses connectivity, the BGP session will drop and the Transit Portal will withdraw the wide-area route. To handle application-level failures, a watchdog process on each node monitors the DONAR processes and withdraws the BGP route if a critical service fails.

4.4.5 Implementation

The software running on each DONAR node consists of several modular components which are detailed in Figure 4.5. They constitute a total of approximately 10,000 lines of code (C++ and Java), as well as another 2,000 lines of shell scripts for system management.

DONAR has been running continuously on Measurement Lab (M-Lab) since October 2009. All services deployed on M-Lab have a unique domain name:

⟨`service`⟩.⟨`account`⟩`.donar.measurement-lab.org`

that provides a closest-node policy by default, selecting from among the set of M-Lab servers. Two of the most popular M-Lab services—the Network Diagnostic Tool (NDT) [100], which is used for the Federal Communication Commission's Consumer Broadband Test, and NPAD [101]—are more closely integrated with DONAR. NDT and NPAD run the DUP protocol, providing DONAR nodes with status updates every 5 minutes.

Since December of 2009, DONAR has also handled 15% CoralCDN's DNS traffic [85], around 1.25 million requests per day. This service uses an equal-split policy amongst its replicas.

DONAR's current implementation supports three front-end mapping mechanisms, implemented as separate processes for extensibility, which communicate via the main DONAR policy engine over a UNIX domain socket and a record-based ASCII protocol. The DNS front-end is built on the open-source PowerDNS resolver, which supports customizable storage backends. DONAR provides a custom HTTP server for HTTP redirection, built on top of the libmicrohttpd embedded HTTP library. DONAR also supports basic HTTP tunneling *i.e.*, acting as a persistent proxy between clients and replicas, via a custom built HTTP proxy. However, due to the bandwidth constraints of our deployment platform, it is currently disabled.

At the storage layer, DONAR nodes use CRAQ to disseminate customer records (public key, domain, and replica information), as well as traffic request rates to each service replica (from each

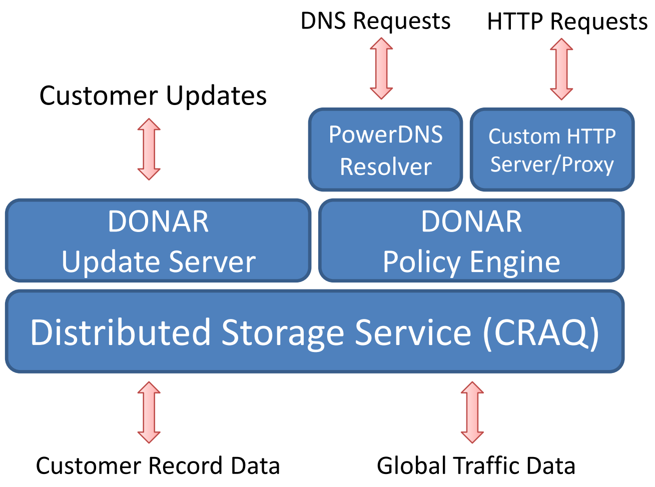


Figure 4.5: Software architecture of a DONAR node

DONAR node). CRAQ's key-value interface offers basic set/get functionality; DONAR's primitive data types are stored with an XDR encoding in CRAQ.

The DONAR policy engine is written in C++, built using the Tame extensions [102] to the SFS asynchronous I/O libraries. In order to assign client regions to replica instances, the engine solves a quadratic program of size $|\mathcal{C}_n| \cdot |\mathcal{I}|$, using a quadratic solver from the MOSEK [103] optimization library.

The DONAR update server, written in Java, processes DUP requests from customers, including account validation, policy updates, and changes in the set of active replicas. It uses CRAQ to disseminate record data between nodes. While customers can build their own applications that directly speak DUP, we also provide a publicly-available Java client that performs these basic operations.

4.5 Evaluation

Our evaluation of DONAR is in three parts. Section 4.5.1 simulates a large-scale deployment given trace request data and simulated mapping nodes. Section 4.5.2 uses the same dataset to verify that client request volumes are reasonably stable from one time period to the next. Finally, Section 4.5.3 evaluates our prototype performing real-world replica selection for a popular CDN.

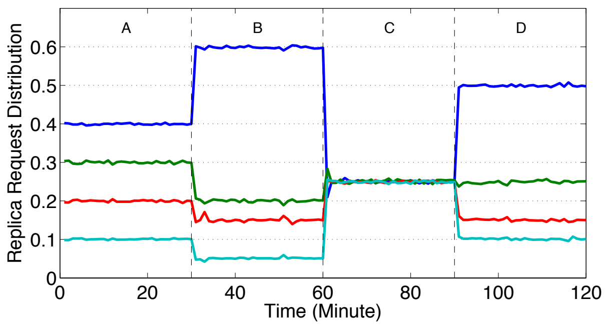


Figure 4.6: DONAR adapts to split weight changes

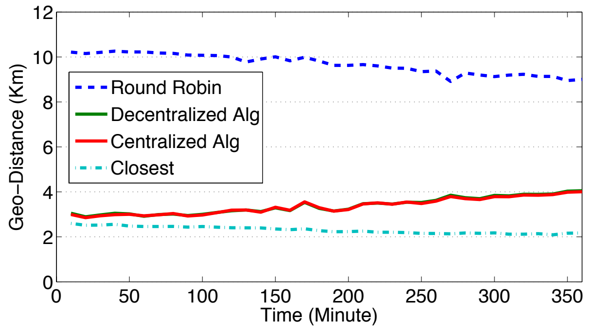


Figure 4.7: Network performance implications of replica-selection policies

4.5.1 Trace-Based Simulation

We use trace data to demonstrate the performance and stability of our decentralized replica-selection algorithm. We analyzed DNS log files from CoralCDN. Our dataset consists of 9,918,780 requests over a randomly-selected 24-hour period (July 28, 2009). On that day, CoralCDN's infrastructure consisted of 76 DNS servers which dispatched clients to any of 308 HTTP proxies distributed world-wide. Locality information was obtained through Quova's commercial geolocation database [9].

In the trace-based simulation, we aggregate clients by geographic location. This results in 371 distinct client regions in our trace. We choose 10 hypothetical mapping nodes to represent a globally-distributed set of authoritative nameservers. Each request is assigned to the nearest nameserver, partitioning the client space. Four replica instances are selected from locations in the US-East, US-West, Europe, and Asia.

We feed each mapping node with client request rates and distributions, which are inferred from the trace for every 10-minute interval. In the evaluation, we allow DONAR nodes to communicate *asynchronously*, *i.e.*, they do not talk in a circular fashion as in Table 4.2. Instead, we overlap updates such that there is 40% probability that at least half of nodes update simultaneously. All nodes perform an update once each minute. We use the quadratic programming solver in MOSEK [103], and each local optimization takes about 50ms on a 2.0GHz dual core machine.

Load split weight. Customers can submit different sets of load split weights over time, and we show how DONAR dynamically adapts to such changes. Figure 4.6 shows the replica request distribution over a 2-hour trace. We vary the desired split weight four times, at 0, 30, 60 and 90 minutes. Phase A shows replica loads quickly converging from a random initial point to a split weight of 40/30/20/10. Small perturbations occur at the beginning of every 10 minutes, since client request rates and distributions change. Replica loads quickly converge to the original level as DONAR re-optimizes based on the current load. In Phase B, we adjust the split weight to 60/20/15/5, and replica loads shift to the new level usually within 1 or 2 minutes. Note that the split weight and traffic load can change simultaneously, and DONAR is very responsive to these changes. In Phase C, We implement an equal-split policy and Phase D re-balances the load to an uneven distribution. In this experiment we chose $\epsilon_i = 0.01$, so there is very little deviation from the exact split weight. This example demonstrates the nice responsiveness and convergence property of the decentralized algorithm, even when the local optimizations run asynchronously.

Network performance. We next investigate the network performance under an equal-split policy among replicas, *i.e.*, all four replicas expect 25% of load and tolerate $\epsilon_i = 1\%$ deviation. We use a 6-hour trace from the above dataset, starting at 9pm EST. We compare DONAR's decentralized algorithm to three other replica-selection algorithms. *Round Robin* maps incoming requests to the four replicas in a round-robin fashion, achieving equal load distribution. *Centralized Alg* uses a central coordinator to calculate mapping decision for all nodes and all clients, and thus does not require inter-node communication. *Closest* always maps a client to the closest replica and achieves the best network performance. The performance of these algorithms is shown in Figure 4.7. The best (minimum) distance, realized by Closest, is quite stable over time. Round Robin achieves the worst network performance, about 300%–400% more than the minimum, since

	% of clients to closest replica	**mean**	**variance**
$\epsilon_i = 0$	54.88%	4.0233	0.0888
$\epsilon_i = 1\%$	69.56%	3.3662	0.1261
$\epsilon_i = 5\%$	77.13%	3.0061	0.1103
$\epsilon_i = 10\%$	86.34%	2.8690	0.4960
closest replica	100%	2.3027	0.0226

Figure 4.8: Sensitivity analysis of using tolerance parameter ϵ_i

75% of requests go to sub-optimal replicas. DONAR's decentralized algorithm can achieve much better performance, realizing 10%–100% above the minimum distance in exchange for better load distribution. Note that the decentralized solution is very close to that of a central coordinator.

It is also interesting to note that DONAR's network cost is increasing. This can be explained by diurnal patterns in different areas: the United States was approaching midnight while Asia reached its traffic peak at noon. Requiring 25% load at each of the two US servers understandably hurts the network performance.

Sensitivity to tolerance parameter. When submitting split weights, a customer can use ϵ_i to strike a balance between strict load distribution and improved network performance. Although an accurate choice of ϵ_i depends on the underlying traffic load, we use our trace to shed some light on its usage. In Figure 4.8, we employ an equal-split policy, and try $\epsilon_i = 0, 1\%, 5\%$ and 10%. We show the percentage of clients that are mapped to the closest replica, and the mean performance and variance, over the entire 6-hour trace. We also compare them to the closet-replica policy. Surprisingly, tolerating a 1% deviation from a strict equal split allows 15% more clients to map to the closest replica. 5% tolerance can further improve 7% of nodes, and an additional 9% improvement is possible for a 10% tolerance. This demonstrates that ϵ_i provides a very tangible mechanism for trading off network performance and traffic distribution.

4.5.2 Predictability of Client Request Rate

So far we have shown rapid convergence given a temporarily fixed set of request rates per client. In reality, client request volume will vary, and DONAR's *predicted* request volume for a given client may not accurately forecast client traffic. For DONAR to work well, client traffic rates must be sufficiently predictable under a granularity that remains useful to our customers. Our current

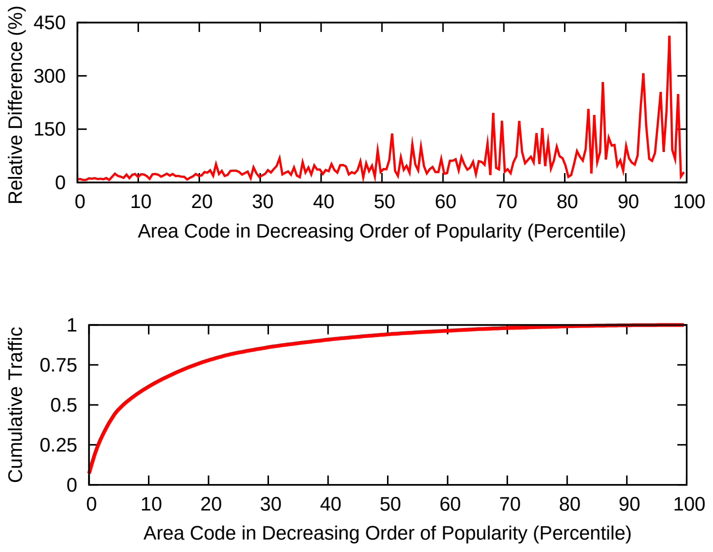


Figure 4.9: Stability of area code request rates

deployment uses a fixed interval size of 10 minutes. We now show via analysis of the same trace data, that request rates are sufficiently predictable on this timescale.

Figure 4.9 (top) plots the relative difference between our estimated rate and the true rate for each client group, *i.e.*, a value of zero indicates a perfect prediction of request volume. Each data point is the average difference over a 2-hour interval for one client. Figure 4.9 (bottom) is a CDF of all traffic from these same clients, which shows that the vast majority of incoming traffic belongs to groups whose traffic is very stable. The high variation in request rate of the last 50% of groups accounts for only 6% of total traffic.

Coarser-grained client aggregation (*i.e.*, larger client groups) will lead to better request stability, but at the cost of locality precision. In practice, services prefer a fine granularity in terms of rate intervals and client location. Commercial CDN's and other replicated services, which see orders-of-magnitude more traffic than CoralCDN, would be able to achieve much finer granularity while keeping rates stable, such as tracking requests per minute, per IP prefix.

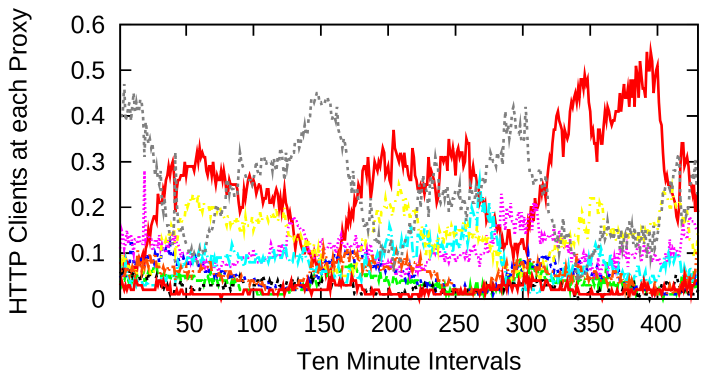


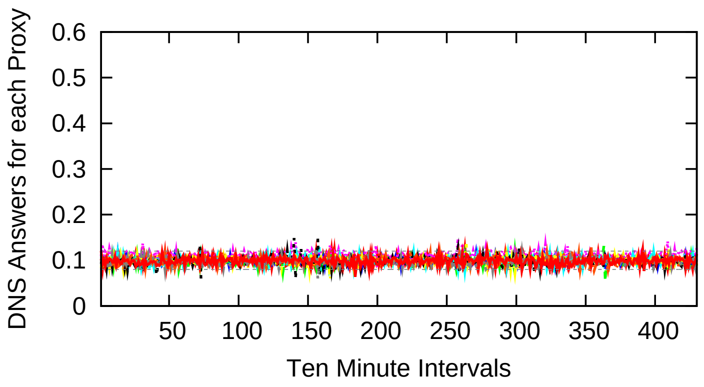
Figure 4.10: Server request loads under "closest replica" policy

4.5.3 Prototype Evaluation

We now evaluate our DONAR prototype when used to perform replica selection for CoralCDN. CoralCDN disseminates content by answering HTTP requests on each of its distributed replicas. Since clients access CoralCDN via the `nyud.net` domain suffix, they require a DNS mechanism to perform replica selection. For our experiments, we create a DONAR account for the `nyud.net` suffix, and add ten CoralCDN proxies as active replicas. We then point a subset of the CoralCDN NS records to DONAR's mapping nodes in order to direct DNS queries.

Closest Replica. We first implement a "closest replica" policy by imposing no constraint on the distribution of client requests. We then track the client arrival rate at each replica, calculated in 10-minute intervals over the course of three days. As Figure 4.10 demonstrates, the volume of traffic arriving at each replica varies highly according to replica location. The busiest replica in each interval typically sees ten times more traffic than the least busy. For several periods a single server handles more than 40% of requests. The diurnal traffic fluctuations, which are evident in the graph, also increase the variability of traffic on each replica.

Each CoralCDN replica has roughly the same capacity, and each replica's performance diminishes with increased client load, so a preferable outcome is one in which loads are relatively uniform. Despite a globally distributed set of replicas serving distributed clients, a naïve replica selection strategy results in highly disproportionate server loads and fails to meet this goal. Furthermore, due to diurnal patterns, there is no way to statically provision our servers in order to



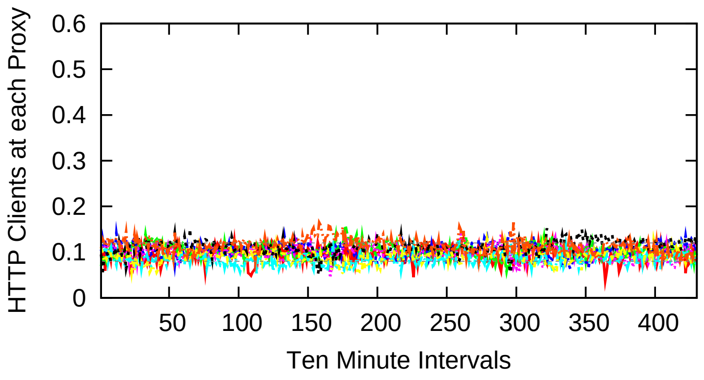


Figure 4.11: Proportional traffic distribution observed by DONAR (Top) and CoralCDN (Bottom), when an equal-split policy is enacted by DONAR. Horizontal gray lines represent the ϵ tolerance $\pm 2\%$ around each split rate.

equalize load under this type of selection policy. Instead, we require a dynamic mapping layer to balance the goals of client proximity and load distribution, as we show next.

Controlling request distribution. We next leverage DONAR's API to dictate a specific distribution of client traffic amongst replicas. In this evaluation we partition the ten Coral servers to receive equal amounts of traffic (10% each) each with an allowed deviation of 2%. We then measure both the mapping behavior of each DONAR node and the client arrival rate at each CoralCDN replica. Figure 4.11 demonstrates the proportion of requests mapped to each replica as recorded by DONAR nodes. Replica request volumes fluctuate within the allowed range as DONAR adjusts to changing client request patterns. The few fluctuations which extend past the "allowed" tolerance are due to inaccuracies in request load prediction and intermediate solutions

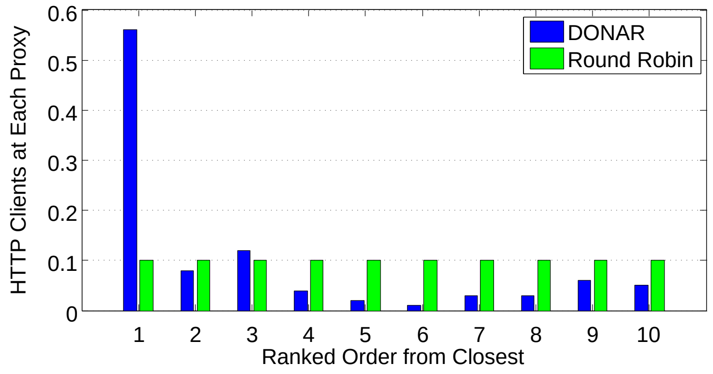


Figure 4.12: Client performance during equal split

which (as explained in Section 4.3) may temporarily violate constraints. Figure 4.11 (Bottom) depicts the arrival rate of clients at each CoralCDN node. The discrepancy in these graphs is an artifact of the choice to use DNS-based mapping, a method which offers complete transparency at the cost of some control over mapping behavior (as discussed in Section 4.6). Nonetheless, request distribution remains within 5% of desired during most time periods. Customers using HTTP-based mechanisms would see client arrival exactly equal to that observed by DONAR.

Measuring client performance. We next evaluate the performance of CoralCDN clients in the face of specific the traffic distribution requests imposed in the prior experiment. Surprisingly, DONAR is able to sufficiently balance client requests, while pairing clients with nearby servers with very high probability. This distinguishes DONAR from simple weighted load balancing schemes, which forgo a consideration of network performance in favor of achieving a specific split. While DONAR nodes define fractional rules for each client, in practice nearly every client is "pinned" to a particular replica at optimum. We can thus show the proportion of clients paired with their nth preferred replica. Figure 4.12 plots this data, contrasting DONAR with a round-robin mapping policy. With DONAR, more than 50% of clients are mapped to the nearest node and 75% are mapped within the top three, a significant improvement over the traditional round-robin approach.

4.6 Related Work

Network distance estimation. A significant body of research has examined techniques for network latency and distance estimation, useful for determining `cost(c,i)` in DONAR. Some work focused on reducing the overhead for measuring the IP address space [104, 90, 105, 106]. Alternatively, virtual coordinate systems [88, 89] estimated latency based on synthetic coordinates. More recent work considered throughput, routing problems, and abnormalities as well [11, 87]. Another direction is geographic mapping techniques, including `whois` data [107, 108], or extracting information from location-based naming conventions for routers [109, 110].

Configurable Mapping Policies. Existing services provide limited customization of mapping behavior. OASIS [82] offers a choice of two heuristics which jointly consider server load and client-server proximity. ClosestNode [111] supports locality policies based on programmatical on-demand network probing [86]. Amazon's EC2, a commercial service, allows basic load balancing between virtual machine instances. For those who are willing to set up their own distributed DNS infrastructure, packages like MyXDNS [112] facilitate extensive customization of mapping behavior, though only by leaving all inputs and decisions up to the user.

Efficacy of Request Redirection. Several studies have evaluated the effectiveness of DNS-based replica selection, including the accuracy of using DNS resolvers to approximate client location [113] or the impact of caching on DNS responsiveness [114]. Despite these drawbacks, DNS remains the preferred mechanism for sever selection by many industry leaders, such as Akamai [6]. While HTTP tunneling and request redirection offer greater accuracy and finer-grain control for services operating on HTTP, they introduce additional latency and overhead.

4.7 Summary

DONAR enables online services to offer better performance to their clients at a lower cost, by directing client requests to appropriate server replicas. DONAR's expressive interface supports a wide range of replica-selection policies, and its decentralized design is scalable, accurate, and responsive. Through a live deployment with CoralCDN, we demonstrate that our distributed algorithm is accurate and efficient, requiring little coordination among the mapping nodes to adapt to changing client demands. In our ongoing work, we plan to expand our CoralCDN and M-Lab deployments and start making DONAR available to other services. In doing so, we hope to identify more ways to meet the needs of networked services.

Chapter 5

Conclusion

Wide-area traffic management is a well-studied problem in the area of transit networks. The advent of cloud computing presents a new opportunity to CSPs for coordination between traffic-management tasks that are previously controlled by different institutions. Today's CSPs face a number of significant challenges in managing their traffic across wide-area networks, including (i) the limited visibility among administratively separated groups that control different management decisions, (ii) the loss of efficiency due to independent decision makings in contrast to a coordinated solution, and (iii) the poor scalability as the need rises for running a joint traffic management across a large number of geo-distributed data centers, applications, and clients. This dissertation follows a *principled* approach to address these problems with a set of new networking solutions, a scalable architectural design, and theoretical results with provable optimality guarantees. In this chapter, we first summarize our main contributions in this dissertation in Section 5.1. We then discuss how to combine different solutions together to provide a new traffic management system in Section 5.2. We briefly propose future research directions in Section 5.3, and conclude with Section 5.4.

5.1 Summary of Contribution

In this section, we revisit the design requirements that are raised in Chapter 1, and summarize our contributions in this dissertation.

First, we observe that CSPs who have their own backbones, *e.g.*, an ISP who wishes to deploy CDN for serving its customers, have the opportunity to coordinate between network routing and server selection decisions. Traditionally, CSPs do not make optimal decisions for both, due to the limited visibility that the traffic engineering group and the CDN have into each other. To understand who should provide what information, we develop three abstractions with an increasing amount of cooperation between two parties, starting from (1) today's *status quo* that relies on end-to-end measurement at the edge, to (2) a partial cooperation by exposing the network topology and routes at the core, and towards (3) a full cooperation by communicating their objectives. We rigorously show that cooperation model (2) leads to a global optimum when traffic engineering and CDN match in their interests, however, in general, suffers from the performance sub-optimality and, as a counter-intuitive observation, might be hurt by the extra visibility. Through a re-design of the two systems by introducing a Nash bargaining solution, we alleviate the performance efficiency loss while keeping the functional separation of existing systems. As a design example motivated by information sharing, this works provides a spectrum of design choices for CSPs on the trade-off space between performance, complexity, and architectural changes.

Second, we study how to maximize performance and minimize cost for CSPs that provide geographically-replicated content services over multiple ISP networks. We propose a joint content placement and server selection solution for last-mile decentralized CDNs located within users' homes. The need for a joint design presents significant challenges that, different traffic management decisions have varied resolutions, *e.g.*, ranging from an individual server to an ISP, and may happen at different time-scales, *e.g.*, ranging from seconds to hours, which is hard to solve in practice. We take a divide-and-conquer approach, and offer a new set of network solutions to capture these heterogeneities. Our proposed algorithms are simple, provably optimal, and can be implemented by a scalable architecture. As a design example motivated by joint control, this work sheds light on overcoming the issues of resource heterogeneity, computational tractability, and implementation complexity that are important for CSPs who wishes to perform a joint optimization over different traffic management decisions.

Third, we address the need for CSPs to perform traffic management operations over a large number of data centers, applications and clients, *e.g.*, offering reliable replica-selection service for geographically distributed clients. Most existing replica-selection systems run on a set of distributed mapping nodes for directing client requests, however, require either central coordination (which has reliability, security, and scalability limitations) or distributed heuristics (which are notorious for optimality, accuracy, and stability). We propose a distributed solution to coordinate server-selection decisions among a set of mapping nodes, allowing them to collectively optimize client performance and enforce management policies such as server load-balancing. As a design example motivated by distributed implementation, this work offers a new paradigm for CSPs to perform effective and distributed traffic management for geo-distributed online services.

The three proposed solutions in this dissertation offer complementary methods for CSPs to manage their traffic across wide-area networks. In practice, each of these design paradigms can be applied to specific systems that CSPs wish to deploy, for the purpose of sharing information, joint control, and distributed implementation.

5.2 Synergizing Three Traffic Management Solutions

A CSP might choose to implement one of the proposed traffic management solutions based on its needs, however, combining these solutions brings additional benefits to a CSP. In this section, we briefly show how to integrate the three solutions to provide a synergistic traffic management system, as described below.

An emerging trend of content distribution on the Internet is to place content close to the edge, *e.g.*, end users, for better proximity and caching. As such, today's CSPs, such as social network providers, wish to deploy the CDN at multiple ISPs, or subscribe to multiple CDNs, for a better geographic coverage. The service provider can leverage our solution in Chapter 3 to decide how to place a large catalog of content, *e.g.*, user data in a social network, across CDNs (located at different ISPs), and how to direct user request at the granularity of CDNs (or ISPs). Within each CDN (or an ISP), our solution in Chapter 2 can be used to coordinate server-selection and network routing in an ISP network. Further, DONAR that we propose in Chapter 4, can be used to provide the replica-selection service for each CDN. Combining these solutions simplifies the management task of a CSP by separating functionalities between different institutions. A CSP can focus on the coordination between a CDN and traffic engineering inside a single ISP network, without the need to worry about traffic management decisions affected by other CDNs and ISPs. In addition, the CDN-specific server-selection task can be outsourced to DONAR to handle customized management policies, without the need to separately implement a replica-selection scheme for each CDN. Therefore, a CSP is freed from the more complicated and error-prone management that it has to handle otherwise.

5.3 Open Problems and Future Work

As the Internet is increasingly a platform for online services, there are a number of important questions that CSPs need to further investigate beyond our work in this dissertation.

5.3.1 Optimizing CSP Operational Costs

Offering good performance to clients at a reasonable cost is the life blood of any CSP. In making traffic management decisions, cloud service providers should weigh operational costs for resources like bandwidth and electricity. This dissertation provides simple abstractions to handle primarily bandwidth costs, by minimizing cross-ISP traffic, and expressing server-load policies in DONAR. In practice, the bandwidth pricing model can be more complicated, such as the 95th-percentile billing used by network transit providers to charge bandwidth consumption, or a tiered pricing plan that is being adopted by cloud-computing platforms like Amazon AWS [1]. On the other hand, the power draw is one of the major capital outlay in modern data centers [115]. The electricity utility costs may vary spatially, *e.g.*, from one data center location to another, and temporally, *e.g.*, from the peak hour to the valley hour. CSPs have the opportunity to minimize their costs by directing clients to cheap servers, routing traffic on inexpensive links, and migrating jobs to under-utilized data centers. Bandwidth pricing has been proposed to reduce such costs [116]. Innovative solutions are needed to capture these location-dependent, time-varying costs, allowing CSPs to have a more accurate and effective control on their operational costs.

5.3.2 Traffic Management Within a CSP Backbone

Large online service providers, such as Google and Yahoo, have their dedicated network backbones. In addition to serving Web-like traffic, some services need to send a large volume of data, *e.g.*, search indices, app updates, or backup storage to multiple data centers. On the surface, the traffic management problem within a CSP backbone seems similar to the traffic engineering performed within an ISP backbone. However, the two problems differ in a number of important ways. First, the CSP has control over the server (*e.g.*, the sending rates) and the network (*e.g.*, which path carries the traffic, or rate split in the case of multiple paths). This presents the opportunity for a joint control over rate allocation and network routing. Second, the CSP hosts a large number of services and applications that have different performance goals (*e.g.*, low latency vs. high throughput), and different priorities (*e.g.*, voice vs data backup). Third, in contrast to intra-domain traffic engineering that handles a relatively stable traffic matrix (*e.g.*, client demands), some CSP services have an infinite traffic backlog—like analytics for search—that wish to consume whatever spare bandwidth the network can offer. This introduces new performance goal, *e.g.*, maximizing throughput, rather than minimizing congestion. All scenarios above raise the need for new approaches to manage traffic in a new CSP network environment.

5.3.3 Long Term Server Placement

As the traffic demand for online services continues to grow, with the increasing popularity of video streaming, social networks and mobile apps, CSPs need to make long-term decisions for upgrading their infrastructure, including improving their peering connectivity with ISPs, and deploying more data centers and CDN servers. To offer more reliable services to clients, CSPs are often faced with two types of choices: (i) placing a small number of servers and establishing better peering contracts in terms of bandwidth, such as Level-3, or (ii) deploying a large number of servers in many ISPs that have better edge access to end users, such as Akamai. In making these decisions, CSPs need to consider the user performance (*e.g.*, latency, throughput), operational costs (*e.g.*, bandwidth, power), and capital investment (*e.g.*, servers, routers). As such, new performance and cost models are needed to drive the long-term planning of server placement and ISP peering selection.

5.3.4 Traffic Management within Data Centers

CSPs usually exploit economies of scale by consolidating a large number of servers in their data centers. Cloud services allow tenants to subscribe a handful of servers or virtual machines (VMs) to host their applications. As the demand for cloud-based services continue to rise, data centers increasingly need efficient traffic management to improve resource utilization inside the network.

Compared to wide-area networks, intra data center traffic management share many common ingredients, such as network routing and content placement. The statistical multiplexing of data center resources, such as CPU and memory, also introduces a new control knob, server (VM) placement—servers from all tenants must be placed with great care to ensure their aggregate resource usage on the network and hosts is fulfilled. [117] takes a first step to jointly manage routing and VM placement, leaving many interesting research directions similar to those addressed in this dissertation.

5.4 Concluding Remarks

In conclusion, this dissertation presented the design, evaluation, and implementation of a set of new wide-area traffic management solutions for large-scale cloud services. We have proposed (i) a cooperation scheme between a network provider and a content provider in the face of a growing trend of content-centric networking, (ii) a joint content distribution solution across a federation of decentralized CDNs, and (iii) a distributed mapping system that outsources server-selection for geo-replicated online services.

The evolution of the Internet has driven these exciting research topics, allowing us to revisit previous approaches we use to manage a network. Today's cloud service providers usually apply *ad hoc* techniques for controlling their traffic. As different parts of the network become a single platform for online services, we have unique opportunities to design the network from a clean slate, following the philosophy of *networking as a discipline* [118]—"*Protocols were built to master complexity, but it is the ability to extract simplicity that lays the intellectual foundations. Abstractions are keys to extracting simplicity.*". We believe that, the abstractions developed in this dissertation are promising enablers for designing wide-area networking solutions in a more systematic, automated and effective way, and will shed light on many others that are yet to be discovered.

Bibliography

[1] Amazon Web Services, "http://aws.amazon.com/," 2010.

[2] R. Kohavi, R. M. Henne, and D. Sommerfield, "Practical guide to controlled experiments on the web: Listen to your customers not to the hippo," in *Proc. ACM SIGKDD*, 2007.

[3] Z. Zhang, M. Zhang, A. Greenberg, Y. C. Hu, R. Mahajan, and B. Christian, "Optimizing cost and performance in online service provider networks," in *Proc. USENIX NSDI*, 2010.

[4] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the electric bill for Internet-scale systems," in *Proc. ACM SIGCOMM*, 2009.

[5] "AT&T." http://www.att.com/.

[6] "Akamai." http://www.akamai.com/.

[7] "Netflix." http://www.netflix.com/.

[8] V. Valancius, N. Laoutaris, L. Massoulie, C. Diot, and P. Rodriguez, "Greening the Internet with nano data centers," in *Proc. CoNEXT*, 2009.

[9] Quova. http://www.quova.com/.

[10] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford, "DONAR: Decentralized server selection for cloud services," in *Proc. ACM SIGCOMM*, 2010.

[11] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iPlane: An information plane for distributed services," in *Proc. USENIX OSDI*, 2006.

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

[13] W. Jiang, R. Zhang-Shen, J. Rexford, and M. Chiang, "Cooperative content distribution and traffic engineering in an ISP network," in *Proc. ACM SIGMETRICS*, 2009.

[14] W. B. Norton, "Video Internet: The next wave of massive disruption to the U.S. peering ecosystem," Sept 2006. Equinix white paper.

[15] AT&T, "U-verse." http://uverse.att.com/.

[16] Verizon, "FiOS." http://www.verizon.com/fios/.

[17] A.-J. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante, "Drafting behind Akamai (Travelocity-based detouring)," in *Proc. ACM SIGCOMM*, 2006.

[18] V. Aggarwal, A. Feldmann, and C. Scheideler, "Can ISPs and P2P users cooperate for improved performance?," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 3, pp. 29–40, 2007.

[19] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz, "P4P: Provider Portal for (P2P) Applications," in *Proc. ACM SIGCOMM*, 2008.

[20] D. DiPalantino and R. Johari, "Traffic engineering versus content distribution: A game theoretic perspective," in *Proc. IEEE INFOCOM*, 2009.

[21] J. F. Nash, "The bargaining problem," *Econometrica*, vol. 28, pp. 155–162, 1950.

[22] D. P. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE J. on Selected Areas in Communications*, vol. 24, no. 8, pp. 1439–1451, 2006.

[23] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *Proc. IEEE INFOCOM*, 2000.

[24] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J.McManus, "RFC 2702: Requirements for Traffic Engineering Over MPLS," September 1999.

[25] D. Xu, M. Chiang, and J. Rexford, "Like-state routing with hop-by-hop forwarding can achieve optimal traffic engineering," in *Proc. IEEE INFOCOM*, 2008.

[26] J. Wardrop, "Some theoretical aspects of road traffic research," *the Institute of Civil Engineers*, vol. 1, no. 2, pp. 325–378, 1952.

[27] T. Roughgarden and Éva Tardos, "How bad is selfish routing?," *J. ACM*, vol. 49, no. 2, 2002.

[28] L. Qiu, Y. R. Yang, Y. Zhang, and S. Shenker, "On selfish routing in Internet-like environments," in *Proc. ACM SIGCOMM*, 2003.

[29] M. Littman and J. Boyan, "A distributed reinforcement learning scheme for network routing," Tech. Rep. CMU-CS-93-165, Robotics Institute, Carnegie Mellon University, 1993.

[30] W. Jiang, R. Zhang-Shen, J. Rexford, and M. Chiang, "Cooperative content distribution and traffic engineering in a provider network," Tech. Rep. TR-846-08, Department of Computer Science, Princeton University, 2008.

[31] M. J. Osborne and A. Rubinstein, *A Course in Game Theory.* MIT Press, 1999.

[32] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods.* Prentice Hall, 1989.

[33] K. Binmore, A. Rubinstein, and A. Wolinsky, "The Nash bargaining solution in economic modelling," *RAND Journal of Economics*, vol. 17, pp. 176–188, 1986.

[34] J. He, R. Zhang-Shen, Y. Li, C.-Y. Lee, J. Rexford, and M. Chiang, "DaVinci: Dynamically Adaptive Virtual Networks for a Customized Internet," in *Proc. CoNEXT*, 2008.

[35] D. P. Bertsekas, *Nonlinear Programming.* Athena Scientific, 1999.

[36] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," in *Proc. ACM SIGCOMM*, 2002.

[37] "Abilene." http://www.internet2.edu.

[38] M. Roughan, M. Thorup, and Y. Zhang, "Performance of estimated traffic matrices in traffic engineering," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, pp. 326–327, 2003.

[39] W. Jiang, R. Zhang-Shen, J. Rexford, and M. Chiang, "Cooperative content distribution and traffic engineering," in *ACM NetEcon*, August 2008.

[40] W. Jiang, D.-M. Chiu, and J. C. S. Lui, "On the interaction of multiple overlay routing," *Perform. Eval.*, vol. 62, no. 1-4, pp. 229–246, 2005.

[41] S. C. Lee, W. Jiang, D.-M. C. Chiu, and J. C. Lui, "Interaction of ISPs: Distributed resource allocation and revenue maximization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 2, pp. 204–218, 2008.

[42] G. Shrimali, A. Akella, and A. Mutapcic, "Cooperative interdomain traffic engineering using Nash bargaining and decomposition," in *Proc. IEEE INFOCOM*, 2007.

[43] M. J. Freedman, C. Aperjis, and R. Johari, "Prices are right: Managing resources and incentives in peer-assisted content distribution," in *Proc. International Workshop on Peer-To-Peer Systems*, February 2008.

[44] D. R. Choffnes and F. E. Bustamante, "Taming the torrent: a practical approach to reducing cross-ISP traffic in peer-to-peer systems," in *Proc. ACM SIGCOMM*, 2008.

[45] Y. Liu, H. Zhang, W. Gong, and D. Towsley, "On the interaction between overlay routing and underlay routing," in *Proc. IEEE INFOCOM*, 2005.

[46] R. T. Ma, D. Chiu, J. C. Lui, V. Misra, and D. Rubenstein, "On cooperative settlement between content, transit and eyeball internet service providers," in *Proc. CoNEXT*, 2008.

[47] J. W. Jiang, S. Ioannidis, L. Massoulie, and F. Picconi, "Orchestration of massively distributed cdns," tech. rep., Department of Computer Science, Princeton University, 2011.

[48] "Cisco visual networking index: Forecast and methodology, 2010-2015."

[49] "https://datatracker.ietf.org/wg/cdni/charter/."

[50] "Nanodatacenters: http://www.nanodatacenters.eu/."

[51] "People's CDN: http://pcdn.info/."

[52] K. W. Ross, *Multiservice Loss Networks for Broadband Telecommunications Networks.* Springer-Verlag, 1995.

[53] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods.* Athena Scientific, 1997.

[54] B. R. Tan and L. Massoulie, "Optimal content placement for peer-to-peer video-on-demand systems," in *Proc. IEEE INFOCOM*, 2011.

[55] F. Kelly, "Loss networks," *The Annals of Applied Probability*, no. 1, 1991.

[56] L. Massoulié, "Structural properties of proportional fairness," *The Annals of Applied Probability*, vol. 17, no. 3, 2007.

[57] H. Kushner and G. Yin, *Stochastic approximation and recursive algorithms and applications.* Springer, 2003.

[58] M. Benaïm and J.-Y. LeBoudec, "A class of mean field interaction models for computer and communication systems," *Perform. Eval*, pp. 11–12, 2008.

[59] "http://www.maxmind.com/app/geolitecity."

[60] C. Huang, A. Wang, J. Li, and K. W. Ross, "Understanding hybrid CDN-P2P: why Limelight needs its own Red Swoosh," in *NOSSDAV*, 2008.

[61] C. Huang, J. Li, and K. W. Ross, "Peer-assisted VoD: Making Internet video distribution cheap," in *Proc. International Workshop on Peer-To-Peer Systems*, 2007.

[62] R. S. Peterson and E. G. Sirer, "Antfarm: Efficient content distribution with managed swarms," in *Proc. USENIX NSDI*, 2009.

[63] Y. Chen, Y. Huang, R. Jana, H. Jiang, M. Rabinovich, B. Wei, and Z. Xiao, "When is P2P technology beneficial for IPTV services," in *NOSSDAV*, 2007.

[64] Y. F. Chen, Y. Huang, R. Jana, H. Jiang, M. Rabinovich, J. Rahe, B. Wei, and Z. Xiao, "Towards capacity and profit optimization of video-on-demand services in a peer-assisted IPTV platform," *Multimedia Systems*, vol. 15, no. 1, pp. 19–32, 2009.

[65] V. Misra, S. Ioannidis, A. Chaintreau, and L. Massoulié, "Incentivizing peer-assisted services: A fluid Shapley value approach," in *Proc. ACM SIGMETRICS*, 2010.

[66] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang, "Improving traffic locality in bittorrent via biased neighbor selection," in *Proc. International Conference on Distributed Computing Systems*, 2006.

[67] R. Cuevas, N. Laoutaris, X. Yang, G. Siganos, and P. Rodriguez, “Deep diving into BitTorrent locality,” in *Proc. IEEE INFOCOM*, 2011.

[68] V. Aggarwal, A. Feldmann, and C. Scheideler, “Can ISPs and P2P users cooperate for improved performance?,” *ACM SIGCOMM Computer Communication Review*, vol. 37, pp. 29–40, July 2007.

[69] J. Wang, C. Huang, and J. Li, “On ISP-friendly rate allocation for peer-assisted VoD,” in *Proc. ACM Multimedia*, 2008.

[70] I. Baev, R. Rajaraman, and C. Swamy, “Approximation algorithms for data placement in arbitrary networks,” *SIAM Journal of Computing*, vol. 38, no. 4, 2008.

[71] S. C. Borst, V. Gupta, and A. Walid, “Distributed caching algorithms for content distribution networks,” in *Proc. IEEE INFOCOM*, 2010.

[72] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, “Placement algorithms for hierarchical cooperative caching,” *Journal of Algorithms*, vol. 38, 2001.

[73] Y. Zhou, T. Z. J. Fu, and D. M. Chiu, “Modeling and analysis of P2P replication to support VoD service,” in *Proc. IEEE INFOCOM*, 2011.

[74] W. Wu and J. C. S. Lui, “Exploring the optimal replication strategy in P2P-VoD systems: characterization and evaluation,” in *Proc. IEEE INFOCOM*, 2011.

[75] AmazonAWS, “Elastic load balancing.” http://aws.amazon.com/ Elasticloadbalancing/, 2010.

[76] DynDNS. http://www.dyndns.com/.

[77] UltraDNS. http://www.ultradns.com/.

[78] B. Maggs, “Personal communication,” 2009.

[79] M. Colajanni, P. S. Yu, and D. M. Dias, “Scheduling algorithms for distributed web servers,” in *Proc. International Conference on Distributed Computing Systems*, May 1997.

[80] M. Conti, C. Nazionale, E. Gregori, and F. Panzieri, “Load distribution among replicated Web servers: A QoS-based approach,” in *Workshop Internet Server Perf.*, May 1999.

[81] V. Cardellini, M. Colajanni, and P. S. Yu, “Geographic load balancing for scalable distributed web systems,” in *MASCOTS*, August 2000.

[82] M. J. Freedman, K. Lakshminarayanan, and D. Mazières, “OASIS: Anycast for any service,” in *Proc. USENIX NSDI*, May 2006.

[83] M. Pathan, C. Vecchiola, and R. Buyya, “Load and proximity aware request-redirection for dynamic load distribution in peering CDNs,” in *OTM*, November 2008.

[84] MeasurementLab. http://www.measurementlab.net/.

[85] M. J. Freedman, E. Freudenthal, and D. Mazières, “Democratizing content publication with Coral,” in *Proc. USENIX NSDI*, March 2004.

[86] B. Wong, A. Slivkins, and E. G. Sirer, “Meridian: A lightweight network location service without virtual coordinates,” in *Proc. ACM SIGCOMM*, 2005.

[87] R. Krishnan, H. V. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, and J. Gao, “Moving beyond end-to-end path information to optimize CDN performance,” in *Proc. ACM SIGCOMM*, 2009.

[88] E. Ng and H. Zhang, “Predicting Internet network distance with coordinates-based approaches,” in *Proc. IEEE INFOCOM*, 2002.

[89] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, “Vivaldi: A decentralized network coordinate system,” in *Proc. ACM SIGCOMM*, 2004.

[90] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, “IDMaps: A global Internet host distance estimation service,” *Trans. Networking*, October 2001.

[91] D. K. Goldenberg, L. Qiu, H. Xie, Y. R. Yang, and Y. Zhang, “Optimizing cost and performance for multihoming,” in *Proc. ACM SIGCOMM*, 2004.

[92] D. P. Bertsekas, *Nonlinear Programming.* Athena Scientific, 1999.

[93] “PlanetLab.” http://www.planet-lab.org/.

[94] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, “In VINI veritas: Realistic and controlled network experimentation,” in *Proc. ACM SIGCOMM*, 2006.

[95] S. Thomson, Y. Rekhter, and J. Bound, “Dynamic updates in the domain name system (DNS UPDATE),” 1997. RFC 2136.

[96] J. Terrace and M. J. Freedman, “Object storage on CRAQ: High-throughput chain replication for read-mostly workloads,” in *USENIX Annual Technical Conference*, June 2009.

[97] Zookeeper. http://hadoop.apache.org/zookeeper/, 2010.

[98] D. Dean, E. W. Felten, and D. S. Wallach, “Java security: From HotJava to Netscape and beyond,” in *Symp. Security and Privacy*, May 1996.

[99] V. Valancius, N. Feamster, J. Rexford, and A. Nakao, “Wide-area route control for distributed services,” in *USENIX Annual Technical Conference*, June 2010.

[100] Internet2, “Network Diagnostic Tool (NDT).” http://www.internet2.edu/performance/ndt/, 2010.

[101] M. Mathis, J. Heffner, and R. Reddy, “Network Path and Application Diagnosis (NPAD).” http://www.psc.edu/networking/projects/pathdiag/, 2010.

[102] M. Krohn, E. Kohler, and F. M. Kaashoek, “Events can make sense,” in *USENIX Annual Technical Conference*, August 2007.

[103] MOSEK, “http://www.mosek.com/.”

[104] J. Guyton and M. Schwartz, “Locating nearby copies of replicated Internet servers,” in *Proc. ACM SIGCOMM*, 1995.

[105] W. Theilmann and K. Rothermel, “Dynamic distance maps of the Internet,” in *Proc. IEEE INFOCOM*, 2001.

[106] Y. Chen, K. H. Lim, R. H. Katz, and C. Overton, “On the stability of network distance estimation,” *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 2, pp. 21–30, 2002.

[107] “IP to Lat/Long server.” http://cello.cs.uiuc.edu/cgi-bin/slamm/ip2ll/, 2005.

[108] D. Moore, R. Periakaruppan, and J. Donohoe, “Where in the world is netgeo.caida.org?,” in *Proc. INET*, June 2000.

[109] V. N. Padmanabhan and L. Subramanian, "An investigation of geographic mapping techniques for Internet hosts," in *Proc. ACM SIGCOMM*, 2001.

[110] M. J. Freedman, M. Vutukuru, N. Feamster, and H. Balakrishnan, "Geographic locality of IP prefixes," in *Proc. Internet Measurement Conference*, October 2005.

[111] B. Wong and E. G. Sirer, "ClosestNode.com: An open access, scalable, shared geocast service for distributed systems," *SIGOPS OSR*, vol. 40, no. 1, 2006.

[112] H. A. Alzoubi, M. Rabinovich, and O. Spatscheck, "MyXDNS: A resquest routing DNS server with decoupled server selection," in *WWW*, May 2007.

[113] Z. M. Mao, C. D. Cranor, F. Douglis, M. Rabinovich, O. Spatscheck, and J. Wang, "A precise and efficient evaluation of the proximity between Web clients and their local DNS servers," in *USENIX Annual Technical Conference*, June 2002.

[114] J. Pang, A. Akella, A. Shaikh, B. Krishnamurthy, and S. Seshan, "On the responsiveness of DNS-based network control," in *Proc. Internet Measurement Conference*, October 2004.

[115] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 68–73, 2009.

[116] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "The price is right: Towards location-independent costs in datacenters," in *Proc. SIGCOMM Workshop on Hot Topics in Networking*, 2011.

[117] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM placement and routing for data center traffic engineering," in *Proc. IEEE INFOCOM Mini Conference*, 2012.

[118] S. Shenker, "The future of networking, and the past of protocols." Presented at distinguished colloquium series, Computer Science Dept., Princeton University, 2011.