ROLE OF PLACEMENT SCHEMES ON THE INTERACTION BETWEEN NETWORK AND CONTENT DELIVERY

A Dissertation Presented
by
ABHIGYAN SHARMA
Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of
DOCTOR OF PHILOSOPHY
July 2014
School of Computer Science

ABSTRACT

Degrees will be listed as:
B. Tech., INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR, INDIA
M. S., UNIVERSITY OF MASSACHUSETTS AMHERST
Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST
Directed by: Professor Arun Venkataramani

A placement strategy determines the locations at which content is placed in a network. An effective placement strategy improves availability of content in presence of failures, and makes judicious use of available resources, such as storage and bandwidth, to reduce the latency of content accesses.

While effective placement is known to improve content delivery in the Internet, this thesis studies the impact of placement strategies on the objectives of the underlying network. Our work is closely related to prior research on the interaction between network and content delivery. To our knowledge, this is the first effort to study how placement strategies shape this interaction and affect the objectives of networks and content delivery systems. We study this interaction in three scenarios: an Internet service provider (ISP) network in which all content is placed at a small number of randomly chosen locations, an ISP that controls both the underlying network and content delivery on its network and therefore has full flexibility in designing placement strategies, and a datacenter of a content delivery network (CDN) whose goal is energy-minimization, and makes both placement and routing decisions to that end. Based on experiments with real ISP and datacenter topologies and content access traces from a leading CDN, we show that content placement is a powerful factor in shaping the network traffic and that simple placement strategies are effective in improving cost-, performance- and energy-related metrics for networks and content delivery systems.

Several applications today generate dynamic content such as stock prices, weather information, and status updates posted on social media websites. Placement of dynamic content is a challenging problem due to a fundamental cost-vs.-performance trade-off: dynamic content replication at multiple locations improves latency of content accesses but increases update propagation costs. We design and implement Auspice, a system for placement of dynamic content across geo-distributed data centers. Auspice infers pockets of high demand for a name and uses a heuristic placement strategy to provide low request latency, low update cost, and high availability. An application suited for Auspice is a global name service to store name-to-address mappings of mobile devices. We extensively evaluate Auspice for an expected workload of such a global name service and show that it significantly outperforms both commercial managed DNS services as well as DHT-based replication alternatives to DNS.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The Internet is adequate for providing connectivity between any two end-hosts across the globe, but is inadequate to provide the high-quality experience end-users expect when interacting with web services today. The reasons for the Internet's shortcomings are numerous. Long propagation delays that are fundamentally limited by the speed of light, transient and persistent congestion on network links, and sub-optimal path selection by the Internet routing protocols to name a few [15, 41, 97, 108].

The Internet's shortcomings are masked and high-quality user experiences are enabled by global networks of datacenters that form "overlays" on top of the Internet. Such an overlay network, commonly called a *content delivery network* (CDN), uses algorithms for optimizing data transfer from the origin of the content to the end-user. These algorithms make several decisions to that end. First, they place content at locations close to end users [41]. Second, to serve each request the "best" server is selected whereby overloaded servers are avoided, and so are servers that are far away from the end-user [56]. Third, if default paths in the Internet are performing poorly, then better alternate paths that are chosen that route traffic via nodes in the overlay network, an approach termed as *overlay routing* [15].

The genesis of this thesis lies in the following question: *what influence do the decisions at the overlay have on the routing strategies of the underlying network, and vice versa?*

This intriguing question is one of long-standing interest in computer science research [106, 95, 69, 42, 54, 121]. Several related questions have been put forth. Does the interaction between network and content delivery yield sub-optimal results? How can we design cooperative mechanisms to leverage these interactions? Do cooperative mechanisms yield benefit in an Internet-like environment? We find that nearly all previous research has focused either on the interaction of overlay routing with the network routing [106, 95], or that of server selection strategies with the underlying routing [69, 42, 54, 121]. Somewhat surprisingly, the role of placement strategies on this interaction has received little attention. Nonetheless, content placement is an important factor in determining the performance and cost objectives of networks and content delivery systems as discussed below.

1.1 Placement affects networks and content delivery systems

The role of placement strategies in improving performance and costs of content delivery is well known [107]. Placement strategies commonly replicate content and in doing so, they increase availability of content, and reduce the latency between an end-user and the content. Replication improves user-perceived performance but increases costs, e.g., replicating a large video file increases the storage requirements [19], and replicating dynamic objects increases the cost of propagating updates to all replicas [14]. Placement strategies used today achieve a good cost-performance tradeoff because they exploit the *locality* inherent in real-world workloads [109, 31, 66, 46]. Placement strategies exploit geographic and temporal locality patterns by replicating content at those times and and at those locations where the content is more popular, and thereby improve performance while minimizing replication costs.

We observe that placement strategies also play a role in influencing cost and performance of the underlying network. A major cost component of the network is the cost of *provisioning capacity* on links in the network [6]. If a placement strategy replicates content at multiple locations, it enables users to fetch content from nearby locations, thereby reducing the aggregate traffic in the network. This traffic reduction helps network operators cut capacity provisioning costs. To ensure good performance, network operators use *traffic engineering* techniques [50], in which routes are computed based on network topology and traffic demand patterns to avoid congestion hotspots.[1] Placement strategies determine where content is available and therefore shape traffic demand patterns. Thus, placement strategies play an indirect role in traffic engineering decisions, and affect network performance as well.

The observation that placement strategies influence objectives of both networks and content delivery raises several questions that form the motivation of this thesis. How does a given placement strategy, especially widely-used placement strategies, such as least recently used (LRU) caching, affect the objectives of networks and content delivery systems? Can we design new placement strategies that better serve objectives of both content delivery and the underlying network? Finally, what is the relative important of content placement vs. other decisions, such as server selection and traffic engineering, in determining objectives of networks and content delivery systems.

1.2 Thesis statement

We state the thesis statement as follows:

Content placement is a powerful factor in shaping the network traffic and simple placement strategies are effective in improving cost-, performance- and energy-related metrics for networks and content delivery systems.

[1]We use the terms routing, network routing, and traffic engineering interchangeably.

1.3 Research problems

In defense of the above thesis, we investigate four research problems. We study the interaction between network and content delivery in following three scenarios focusing on the role of placement strategies: an Internet service provider (ISP) network in which content is placed at a small number of randomly chosen locations (Section 1.3.1 & Chapter 3), an ISP that controls both the underlying network and content delivery on its network and therefore has full flexibility in designing placement strategies (Section 1.3.2 & Chapter 4), and a datacenter of a content delivery network (CDN) whose goal is energy-minimization, and makes both placement and routing decisions to that end (Section 1.3.3 & Chapter 5). As most network traffic is generated by static content, such as video, audio and images [33], we study the interaction between network and content delivery assuming that content is static. The placement of dynamic content, the fourth problem in this thesis, is considered solely as a content delivery problem, unrelated to its interaction with the network. We design and implement a placement strategy for dynamic content stored across geo-distributed datacenters (Section 1.3.4 & Chapter 6).

1.3.1 Performance and cost optimization in Internet service providers

The first problem studies the interaction of ISP traffic engineering with a placement strategy that replicates content at a small number randomly chosen locations. We assume that end-users leverage replicated content by downloading in parallel from all locations. We seek to understand how this interaction affects user-perceived metrics, such as file download time and Voice-over-IP (VoIP) call quality, as well as a traffic engineering metric. We consider several classes of TE strategies, and networks with varying degrees of replication of content. We conduct very large scale experiments simulating ISP traffic as a collection of TCP flows seeking to answer following questions:

- How do TE strategies compare in terms of user-perceived performance metrics?
- What is the effective capacity of different TE strategies to tolerate surges in traffic demand?
- How much benefit does TE give over a simple static routing?

1.3.2 Network CDN

A network CDN (NCDN) is an ISP that manages content delivery to users on its network. Unlike a traditional ISP (Section 1.3.1) that control only the routing, an NCDN has flexibility of choosing the placement strategy as well. We classify the possible NCDN strategies into *planned strategies* whose decisions are based on historic content demand patterns, and *unplanned strategies* that are oblivious to historic content demand patterns. We consider several strategies for an NCDN including a planned joint-optimization of placement and routing, an ideal planned strategy with perfect knowledge of future demand patterns, and a strategy that makes both placement and routing decisions in an unplanned manner. We conduct trace-driven evaluations based on real ISP topologies and content access traces from a leading commercial CDN, Akamai, to answer following questions:

- How do simple, unplanned strategies compare to the ideal planned strategy?
- How much benefit do joint optimization strategies yield over simpler strategies as practiced today, and does the benefit warrant the added complexity?
- What is the relative importance of optimizing placement vs optimizing routing?

1.3.3 Energy efficiency of CDN datacenters

The third problem considers the design of an energy-efficient CDN datacenter while ensuring a minimal impact on user-perceived performance either due to cache misses or highly-loaded servers. Our key insight is that jointly optimizing underlying network routing and application-level policies of load balancing and server shutdown can increase datacenter energy savings. We are also exploring novel load balancing and server shutdown policies so that the resulting placement of content on servers ensures high cache hit rates and minimizes impact on user-perceived performance. We plan to conduct trace-driven experiments with traces from a commercial CDN datacenter as well as experiments with a prototype to answer following questions:

- What is the impact of energy-saving strategies on user-perceived performance? How do we design server shutdown and load balancing strategies to minimize this impact?
- How much additional energy savings can be obtain by a joint optimization of network and server energy?
- How do different types of content workloads, e.g, video, downloads, mixture of all traffic types, affect these results?

1.3.4 Geo-distributed placement of dynamic content

Several applications today generate dynamic content such as weather, stocks, status updates posted on social media websites [14, 10]. Placement of dynamic content is a challenging problem due to a fundamental cost-performance tradeoff: dynamic content replication at multiple locations improves latency of content accesses but increases update propagation costs. State-of-art-systems either require manual placement configuration or use naive placement strategies, such as replicate-at-all-locations or replicate-at-k-random-locations, incur excessively high update cost or sub-optimal latency [36, 99].

Our research problem is to design and implement a system that meets the following requirements. (1) The placement of replicas should minimizes request latencies under pre-defined resource constraints across geo-distributed datacenters. (2) The design should not pose a fundamental limitation either on the number of datacenter locations or the number of objects stored in the system. (3) The system should provide flexible consistency semantics to meet requirements of wide-range of applications that need either strong or weak consistency semantics.

1.4 Research approach

We highlight the key elements of the approach we take to accomplish our research.

Experimental methodology: We take an empirical approach to evaluate our hypotheses, including experimentation with prototypes, and packet- and flow-level simulations. The choice of the experimental methodology is dictated by the research problem in most cases. For example, experimenting with an actual NCDN infrastructure is beyond our resources, therefore our evaluation for that problem uses flow-level simulations (Chapter 4). In evaluating TE strategies based on user-perceived metrics (Chapter 3), we perform packet-level simulations as they help us measure user-perceived metrics more accurately compared to flow-level simulations.

Data collection: We have performed extensive data collection from multiple ISPs including a Tier-1 US ISP, and from a leading commercial CDN, Akamai. From each ISP, we obtained point-of-presence-level (PoP-level) topology maps as well as PoP-to-PoP traffic demands represented in the form of traffic matrices. We have collected two sets of anonymized content access traces from Akamai CDN. The first set includes traffic for bit-intensive content such as video, and file downloads from users across the globe; these traces are used in evaluation of strategies for an NCDN. The second set of traces are collected at a single datacenter and include traffic of all content types; these traces will be used in evaluating the design of energy-efficient CDN datacenters. Our CDN traces contain logs of content accesses by millions of users, and reveal demand patterns at the level of individual objects, and therefore help us in evaluating placement strategies accurately.

Evaluation metrics: Our evaluation focuses on user-perceived performance metrics over system-level metrics, wherever possible. We find that a user-centric evaluation has potential to yield new insights. For example, Chapter 3 shows that common traffic engineering strategies, yield near-identical user-perceived performance, while the same set of strategies differed by up to 2$\times$ on a system-level metric of maximum link utilization. Along the same lines, we plan to conduct evaluation of end-to-end download performance in our ongoing work on designing energy-efficient CDN datacenters.

1.5 Contributions

This section presents a summary of contributions of this thesis.

1. We consider a placement strategy that replicas content at a small number of randomly chosen locations in an Internet service provider (ISP) network; end-users leverage the replicated content by downloading in parallel from all locations. Our experiments evaluate common traffic engineering strategies based on real topologies and traffic matrices. Our key contribution is to show that even a simple placement strategy that allows users to download content from just 2-4 locations enables all traffic engineering schemes to achieve near-optimal capacity to tolerate surges in traffic demand, and even simple static routing to be within 30% of optimal.

2. We model a single entity that controls both network and content delivery as in the case of a network CDN (NCDN). We classify the possible NCDN strategies into *planned strategies* whose decisions are based on historic content demand patterns, and *unplanned strategies* that are oblivious to historic content demand patterns. Our experiments on real network topologies and traces from a large CDN surprisingly show that unplanned placement strategies, e.g., LRU caching, combined with a simple static routing outperform (sometimes significantly) realistic (i.e., history-based) joint-optimization strategies and can achieve network cost and user-perceived latencies close to those of a joint-optimal strategy with future knowledge.

3. We design and implement Auspice, a system for dynamic content replication across geo-distributed datacenters. Our system infers pockets of high demand for a name and uses a heuristic placement strategy to provide low request latency, low update cost, and high availability. An application suited for Auspice is a global name service to store name-to-address mapping of mobile devices. Our extensive evaluation for an expected workload of such a global name service on multiple testbeds shows that Auspice significantly outperforms both commercial managed DNS services as well as DHT-based replication alternatives to DNS.

1.6 Organization

Chapter 2 presents the main concepts used in this thesis – traffic engineering techniques used by ISPs, algorithms used for content delivery including those for content placement and for request direction. This chapter also discusses prior work on the interaction between traffic engineering and content delivery.

Chapter 3, Chapter 4 and Chapter 6 present solutions and experimental results for the problems in Section 1.3.1, Section 1.3.2 and Section 1.3.4 respectively. These chapters describe the research we have already completed towards this thesis.

Chapter 5 surveys prior research on the design of energy efficient datacenters, and identifies key research questions that need to be answered in designing an energy-efficient CDN datacenter.

Chapter 7 presents a plan for future research to be accomplished towards the completion of this thesis.

1.7 Previous publications and collaboration

Chapter 3 revises a previous publication: A. Sharma, A. Mishra, V. Kumar, A. Venkataramani. Beyond MLU: An Application-Centric Comparison of Traffic Engineering Schemes. *Proc. IEEE INFOCOM, April 2011*. Aditya Mishra and Vikas Kumar provided invaluable support in performing experiments for this work.

Chapter 4 revises a previous publication: A. Sharma, A. Venkataramani, R. Sitaraman. Distributing Content Simplifies ISP Traffic Engineering. *Proc. ACM SIGMETRICS, June 2013*. Ramesh Sitaraman provided access to Akamai datasets for this work. A realistic experimental evaluation would not have been possible without these datasets.

Chapter 6 revises a previous publication: A. Sharma, X. Tie, H. Uppal, D. Westbrook, A. Venkataramani, A. Yadav. A Global Name Service for a Highly Mobile Internetwork. *Proc. ACM SIGCOMM, August 2014*. This work also appears in Xiaozheng Tie's thesis, which describes the same placement algorithm and a simulation-based evaluation of the algorithm. The new material in this chapter includes (1) mechanisms to provide consistency of data and (2) experiments with an implementation of the placement algorithm in an emulation testbed and a geo-distributed testbed. Both Xiaozheng Tie and Hardeep Uppal have contributed to the placement algorithm. Hardeep Uppal, David Westbrook and Arun Venkataramani have contributed in implementing the Auspice system.

CHAPTER 2

BACKGROUND

This thesis builds on prior research in three areas: traffic engineering, content delivery, and the interaction of the two. We first review the major classes of traffic engineering strategies (Section 2.1). Next, we discuss common techniques used by content delivery systems, including strategies for content placement and for request redirection (Section 2.2). Finally, we survey research on the interaction between network and content delivery, and find that the interaction between traffic engineering and either overlay routing or request redirection have been the focus of past research (Section 2.3).

2.1 Traffic engineering

The goal of traffic engineering (TE) is to avoid congestion hotspots in the network by optimizing routes based on network topology and expected traffic demand. In the context of large Internet service provider (ISP) networks, traffic engineering decides both intra-domain (within the ISP) and inter-domain routing (across ISPs). We focus here on intra-domain routing and refer the reader to [48, 104] for a survey of inter-domain traffic engineering.

We classify traffic engineering schemes based on the frequency at which they update routing. By this attribute, TE schemes can be grouped into three categories: (1) *Oblivious TE* uses static routes that are seldom updated [20, 22]. (2) *Offline TE* updates routes periodically, e.g., every few hours or every few days, based on recent history of traffic demand [50, 49]. (3) *Online TE* updates routes at timescales of hundreds of milliseconds, reacting instantaneously to traffic demand changes [70].

TE schemes are evaluated based on link utilization based metrics, e.g., a widely used metric is maximum link utilization [117]. A TE scheme is usually compared against the optimal solution that minimizes the given metric by solving a multi-commodity flow optimization [70, 50]. By this measure, oblivious routing schemes perform poorly and can be shown to be arbitrarily worse compared to the optimal strategy [50]. For many ISPs networks, simple oblivious routing schemes are sub-optimal by a small constant factor [117, 122, 70]. Offline TE schemes, while sub-optimal, perform superior to oblivious TE schemes. e.g., Fortz and Thorup show that offline TE delivers up to $2\times$ better on AT&T network backbone. Online schemes have been shown to achieve near-optimal performance, but they are rarely used in production networks.

In practice, offline TE based on Open Shortest Path First (OSPF) and Multi-protocol Label Switching (MPLS) are commonly used [117, 122, 50, 45]. Routes computed by OSPF traffic engineering must follow shortest-weight paths, therefore OSPF TE provides limited functionality to split traffic among multiple paths. MPLS TE overcomes this limitation by enabling traffic between two nodes to be split in arbitrary ratios among multiple paths. Therefore, MPLS TE gives better results than OSPF TE [117, 122].

Prior work on traffic engineering is based primarily on evaluation of link utilization based metrics, and has largely ignored the impact of traffic engineering on user-perceived performance. Further, the comparison of TE schemes has not taken into account the interaction with content delivery. This thesis contributes in answering these questions. In Chapter 3, we provide a comparison of traffic engineering schemes focusing on user-perceived metrics such as file download times, and VoIP call quality. In Chapter 3 and Chapter 4, we evaluate TE schemes while accounting for the interaction between TE and content delivery and show that this interaction helps simpler TE schemes, such as oblivious TE or OSPF TE, perform closer to the optimal TE strategy in terms of user-perceived metrics as well as TE metrics.

2.2 Content delivery

Content delivery systems seek to provide a high-quality experience to users accessing content in all regions at all times. A canonical example of a content delivery system is a content delivery network (CDN). State-of-the-art CDNs operate geo-distributed datacenters, and use a combination of edge caching, intelligent server selection, and path and protocol optimizations for delivery of several types of content, e.g., video, bulk downloads, and interactive websites [41, 86]. Given their geo-distributed deployment, the decisions of content placement, i.e., locations at which a content is placed, and request redirection, i.e., which location is best positioned to serve a user's request, are central to the functioning of a CDN.

2.2.1 Content placement

Placement strategies depend on whether content is static or dynamic. Dynamic content has limited cacheability therefore placement strategies for static content are not always applicable for dynamic content.

2.2.1.1 Static content placement

Static content, such as videos, audio, images and software updates, contribute to a vast majority of traffic in the Internet [85, 33]. The placement of static content is commonly handled by a caching strategy. A simple and widely used caching strategy is least recently used (LRU) cache replacement [119]. Caching strategies are effective because they exploit geographic and temporal locality of requests, resulting in high cache hit rates in many cases [38, 109, 57]. An alternative to caching is a planned placement approach, which prepositions content at a set of locations based on prior knowledge of demand. Planned placement is effective in scenarios where workloads are predictable over long intervals, e.g., hours, or days [19].

2.2.1.2 Dynamic content placement

Several applications today generate dynamic content such as stock prices, weather information, price catalogs. Such dynamic content is typically stored at a small number of fixed locations across the globe, mostly for fault tolerance objectives [7]. Due to a limited number of fixed replica locations, content accesses from regions away from the replica locations incurs high latency [7, 14]. Extensively replicating dynamic content is costly due to bandwidth and server resources used in propagating updates to all locations.

Placement strategies for dynamic content is an active research area. A naive placement strategy of replicating all data at all locations would incur high update costs. The alternatives provided by current systems either require manual configuration to decide placement or result in sub-optimal latency. For example, Spanner [36] provides configuration options to manually select which locations should a given subset of data be replicated. DHT-based systems automatically decide placement but result in high latency because replica are chosen randomly [99, 98]. Volley [14] uses a placement heuristic to select a single best location for each data item, but it would result in sub-optimal latencies when a data item is popular across many geographic regions.

This thesis addresses the problem of dynamic data placement across geo-distributed data centers while enhancing prior work in this area (Chapter 6). Our system, Auspice, automatically makes data placement decisions (unlike Spanner), places data replicas based on demand-locality (unlike DHT-based replication), and creates multiple replicas of each object (unlike Volley) and limits update propagation costs.

2.2.2 Request redirection

Request redirection strategies complement placement strategies by selecting the server location that is best suited to process a user's request. These strategies have been extensively studied and form the heart of CDN technology today. To quote from a report by Akamai, *"the system directs client requests to the nearest available server likely to have the requested content."* where the "nearest" server is one whose round trip latency as well as packet losses are small, and an "available" server is one that is lightly loaded considering all resources, i.e., network, CPU and disk [41].

Request redirection is implemented using three processes: (1) *Monitoring:* Probe messages sent intermittently help monitor network characteristics and server load and identify congested regions of network and overloaded server locations [56, 118]. (2) *Estimating distances:* The measured statistics are combined to compute a distance function that reflects the proximity of a server location to users in a geographic region [118]. (3) *Informing the user:* The user is informed of selected server/s either via DNS resolution or via HTTP redirection as described in [41] and [23].

2.3 Interaction between network and content delivery

Studying the interaction between network and content delivery has been a topic of much interest in both systems and theory communities. Several related questions have been put forth. Do these interactions negatively affect objectives of networks and content delivery systems? What is the sub-optimality caused due to these interactions in the worst case, and for typical topologies and traffic demands? How to leverage these interactions to improve traffic engineering and content delivery objectives?

Yet, we don't fully understand these interactions because prior research has studied the interaction of network routing with only a subset of content delivery decisions. Much prior research has focused on two aspects: the interaction of overlay routing and network routing [106, 95], and the interaction of request redirection and network routing [69, 42, 54, 121]. While placement decisions are critical to user-perceived performance, there has been little research on how content placement interacts with network routing.

2.3.1 Interaction between traffic engineering and overlay routing

Several results show the negative interaction between selfish overlay routing and network routing [106, 95], however it appears that selfish overlay routing is not used by most of the Internet traffic. For example, traffic from CDN edge server to the client always follows network routing. Further, overlay routing yields "marginal" benefits ($< 30\%$) over network routing for 79%-96% of paths depending on which geographic region is being considered [97], which suggests that traffic between CDN servers forming an overlay network follows network routing in most cases. For this reason, this thesis does not model the interaction between overlay and network routing.

2.3.2 Interaction between traffic engineering and request redirection

Recent research has investigated the interaction between request redirection and traffic engineering, without considering the role of placement strategies. This interaction is commonly studied in the context of Internet service providers (ISPs) and content providers (CPs) with geo-distributed datacenters. Both analytical results [69, 42] and system implementations [54, 121] have shown that there is value for joint optimization of request redirection and traffic engineering, and cooperative strategies can help traffic engineering metrics and also reduce user-perceived latencies. Commonly, these efforts assume that all content is available at all locations, ignoring the fact that content availability at a location depend on placement strategies. Therefore, in this thesis, we account for the effect of content placement along with request redirection, in studying the interaction between network and content delivery.

2.4 Summary

In this chapter, we reviewed prior research on traffic engineering and summarized different classes of traffic engineering schemes. We also reviewed common techniques used for content delivery, namely strategies for placement of static content and of dynamic content, and request redirection techniques used by CDNs. Finally, we reviewed research on interaction between network and content delivery, and found that much prior research in this area has focused on two aspects, the interaction of overlay routing and network routing, and the interaction of request redirection and network routing.

CHAPTER 3

TRAFFIC ENGINEERING: AN APPLICATION-CENTRIC COMPARISON

Traditionally, traffic engineering (TE) has been studied as an optimization problem that takes as input a traffic matrix (TM) and seeks to compute routes so as to minimize a network cost function. The cost function is intended to capture the severity of congestion hotpsots based on link utilization levels. For example, the most widely used cost function, MLU, is simply the utilization of the most utilized link in the network [117, 70, 122, 20]; others sum over all links a convex function of their utilization (so as to penalize highly utilized links more) [50, 49]. There are two implicit assumptions underlying this line of work. First, maintaining low link utilization improves user-perceived application performance under typical load conditions. Second, maintaining low link utilization increases the effective capacity of the network by enabling it to accommodate unexpected surges in the traffic demand.

Our work questions both of the above assumptions. The distinguishing aspect of our work is an application-centric approach to the problem: instead of posing TE as as optimization problem seeking to minimize link utilization, we focus on application performance metrics such as TCP throughput for elastic traffic and quality-of-service metrics (e.g., MOS score for VoIP quality [35]) for inelastic traffic. Accordingly, our evaluation methodology is empirical: instead of relying on mathematical simulations based on linear programming or heuristic techniques for NP-complete problems, our experiments carefully and at scale simulate end-to-end application behavior so as to compare TE schemes with respect to their impact on application performance.

Our application-centric and empirical approach reveals rather unexpected results. Our first finding is that metrics based on link utilization alone, and in particular MLU, are a poor proxy for application performance. For example, a TE scheme may incur twice the MLU of another TE scheme and yet achieve as good or better application performance. The key reason for this mismatch is that application performance is largely determined by end-to-end loss rate and delay, but link utilization does not capture them accurately. At typical Internet loads, and in fact until the utilization starts approaching the capacity, link loss rates remain negligibly small. This observation has also been confirmed by explicit measurements on Internet backbones [25], and is consistent with studies on ISP backbones showing that over 90% of all packet loss is caused by interdomain routing fluctuations as opposed to high utilization [64] and 90% of TCP flows experience no packet loss [53]. Furthermore, end-to-end Internet path delays are known to be largely determined by propagation delays as opposed to queueing delays [53, 89].

As a result, we find that all state-of-the-art TE schemes achieve nearly identical application performance at typical Internet load levels. In fact, even static shortest-path routing with link weights inversely proportional to the capacity (InvCap) (i.e., no engineering at all) achieves the same application performance as optimal TE. Ironically, TE schemes that engineer for unexpected traffic spikes (e.g., COPE [117]) consistently hurt TCP throughput despite achieving near-optimal MLU.

More surprisingly, we find that application adaptation to location diversity, i.e., the ability to download content from multiple potential locations, blurs differences even in the achieved capacities of different TE schemes enabling all of them to be near-optimal. With location diversity, we find that the inverse of the MLU is no longer a meaningful metric of capacity. Instead, we formalize a new metric of the capacity achieved by a TE scheme called the *surge protection factor* (SPF) that captures the factor of increase in demand that can be sustained while accounting for location diversity. TE schemes calculate routing based on a measured traffic matrix to achieve a desired network cost, but application adaptation to location diversity changes the traffic matrix itself in response to a change in routing resulting a different network cost than expected. As a result, the optimal TE scheme with perfect knowledge of traffic matrix and sub-optimal TE schemes like OSPF weight-tuning [50] end-up achieving the same SPF. Even the static routing scheme, InvCap, achieves an SPF at most 30% worse than optimal TE.

The rest of the chapter is as follows. Section 3.1 explains how location diversity changes the TE problem. Section 3.2 presents our simulation setup. Section 3.3 compares the application performance of TE schemes and Section 3.4 compares their achieved capacity under location diversity.

3.1 Engineering traffic with location diversity

In this section, we introduce location diversity, explain how it changes the traffic engineering problem, and introduce a new metric to quantify the capacity achieved by traffic engineering schemes with location diversity.

3.1.1 Location diversity: Prevalence

Location diversity, or the ability to download content from multiple potential locations, is widespread in the Internet today. Major commercial CDNs, e.g., Akamai [2], Level-3 [74], EdgeCast [30] etc., commonly replicate content at hundreds of locations and redirect users to the best server based on proximity or dynamic monitoring of server and network congestion [111]. Popular P2P applications such as BitTorrent [34], PPLive [79] download content simultaneously from many peers that are chosen based on a number of factors including network congestion. Other examples of location diversity include cloud computing infrastructure providers such as Google and Amazon with geographically distributed sites; content hosting services such as Carpathia [29], Rapidshare [17], etc.; mirrored websites such as SourceForge, Debian, etc.

Although quantifying the extent of location diversity in today's Internet is difficult, back-of-the-envelope calculations based on existing measurement studies suggests that it is significant. CDNs alone are estimated to account for 10% of Internet traffic [103]. Major cloud computing and content hosting companies with location diversity contribute to a significant fraction of Internet traffic, e.g., Google (6%), Comcast (3%), RapidShare (5%) and Carpathia (0.5%), a trend that is projected to increase in the near future [110, 103]. The fraction of P2P traffic in Internet was estimated to be between 18-60% by different measurement studies in 2009.

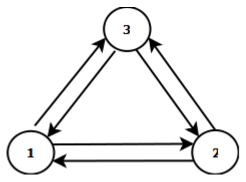
3.1.2 Location diversity: Impact on TE

Location diversity necessitates revisiting traffic engineering as it changes the assumptions underlying the traditional formulation of the problem, as described next.

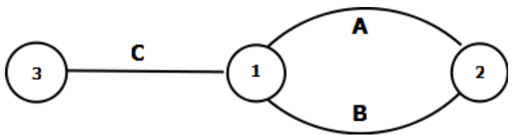
Location diversity can significantly increase the capacity of a network. For example, consider the triangular network in Figure 3.1(a). Suppose each link has 100 Mbps of capacity and each node seeks to download some content. Without location diversity, each node can download its content from exactly one location, say its counter-clockwise neighbor, i.e., 1 downloads from 2, 2 from 3, and 3 from 1. In this case, each node gets 150 Mbps of flow using both the direct and the 2-hop path to its source node. With location diversity. each node can download from both adjacent nodes. Now each node can receive a total of 200 Mbps. In this example, a diversity of two locations increases the capacity of the network by 200/150 = 1.33.

3.1.2.1 Location diversity changes the TE problem

A key assumption underlying the traditional formulation of the TE problem is that the input traffic matrix is fixed, i.e., computing routes by itself does not change the traffic matrix (although it may change over time due to inherent variation in user demand). However, when applications can leverage location diversity, the traffic matrix itself depends upon the TE scheme, i.e., the very act of computing routes can change the matrix.



(a) Triangle network



(b) Lasso network

Figure 3.1. Example networks

The three-node network in Figure 3.1(b) exemplifies the above phenomenon. All links are assumed to have a capacity of 100 units and a constant delay. The top link A has a very small delay compared to the other two links that both have equal delay. Node 1 has 100 Mbps of demand that it can obtain from 2 as well as 3. In addition, there is 20 Mbps of demand at node 1 which it can obtain only from 2. We assume that the aggregate demand at a node consists of a large number of user-initiated connections. When content can be downloaded from multiple locations, users initiate parallel TCP connections and the throughputs along paths in a parallel TCP connection are inversely proportional to the path delays. The TE scheme is assumed to be OSPF-based, i.e., shortest-path routing using configured link weights and traffic split equally among multiple paths with equal weights.

Suppose the weights of the links A and B are unequal and the link A has more weight. As a result, all of the traffic between 1 and 2 is routed using only link B. 1 splits its demand of 100 Mbps using parallel TCP equally between links B and C. Thus, the traffic on links A, B, and C is 0, 70, and 50 respectively. In the next step, seeking to balance load better for this resultant matrix, the TE scheme sets both the links A and B to the same weight (hoping to achieve link utilizations of 35, 35, and 50 respectively). Consider how parallel TCP connections respond to this change. Assuming each TCP connection between 1–2 is pinned to only one of the two paths—as is commonly done in practice to achieve equal-cost multi-path (ECMP) splitting—50 Mbps of demand at 1 gets routed using parallel TCP connections over the link A and link C, and an equal amount using parallel TCP connections along the link B and link C. In addition, the 20 Mbps of background traffic is split equally among link A and link B as per ECMP. Since link A has a much smaller delay than link C, the 50 Mbps of demand at 1 using parallel TCP along those two paths will flow entirely through link A. The remaining 50 Mbps using B and link C will get split equally across the two paths by parallel TCP. Thus, the traffic on the links A, B and C is 60, 35, and 25 respectively, which is different from what the TE scheme engineered for (namely, 35, 35, and 50). The resulting MLU of 0.6 is different compared to 0.5, the value that the TE scheme expected.

3.1.3 Location diversity: Quantifying capacity

How can we quantify the capacity achieved by a TE scheme in the presence of location diversity? In general, the capacity is a *region* that includes all of the traffic matrices that it can accommodate. However, quantifying the capacity of a TE scheme as a region may shed little light on its ability to tolerate typically encountered load spikes. Furthermore, it is cumbersome to compare TE schemes that achieve overlapping capacity regions. So, it is common to use a more concise metric such as the MLU to characterize the capacity with respect to a given traffic matrix. Intuitively, the inverse of the MLU serves as a metric of capacity, e.g., if a TE scheme achieves an MLU of 0.25 for a given matrix, then it can tolerate up to a $4\times$ surge in the load represented by the matrix. Unfortunately, as the example in Figure 3.1(b) shows, MLU is not a meaningful metric of capacity when application adaptation to location diversity determines the traffic matrix.

With location diversity, the demand is best represented as a "content matrix" that specifies for each node and each content the traffic for that content at that node and the set of source locations from where that content can be downloaded (e.g., 100 Mbps at node 1 downloadable from 2 and 3, and 20 Mbps at node 1 downloadable from node 2, in Figure 3.1(b)). The traffic matrix corresponding to this demand depends upon the underlying routes and application behavior (e.g., how parallel TCP splits traffic across the download locations). Furthermore, scaling the demand does not simply scale the traffic matrix entries by the same factor. In general, it is difficult to predict how application behavior might change the traffic matrix for a projected surge in demand, as that change depends upon the underlying routes that in turn depend upon the original traffic matrix. Indeed, as the example shows, even if the demand is unchanged, the mere act of engineering routes can change the traffic matrix yielding a different MLU than expected.

3.1.3.1 An empirical capacity measure

We propose a new metric, *surge protection factor* (SPF), to quantify the capacity achieved by a TE scheme with respect to a traffic matrix. Let E denote a TE scheme, M the demand specified as a content matrix. When there is no location diversity, M can be easily transformed to a unique traffic matrix $T(M)$. Let MLU$(E, T(M))$ denote the MLU achieved by E given the traffic matrix $T(M)$. In this case, SPF(E, M) is simply the inverse of MLU$(E, T(M))$, i.e., the factor of increase in the demand that can be satisfied. However, in the case when there is location diversity, SPF(E, M) is an *empirical* measure of the satisfiable increase in demand computed as follows. Let kM denote the demand that scales each entry in M by a factor $k > 1$. Then, SPF(E, M) is defined as the largest k such that the routing computed by E (for the matrix $T(M)$) can satisfy the demand kM.

Determining if an engineering scheme can satisfy a projected demand is difficult as it requires us to accurately model application adaptation to location diversity, so SPF is useful mainly as an empirically measured capacity metric. To this end, we describe our experimental setup next.

3.2 Experimental setup

In this section, we describe our experimental setup based on ns-2 used to compare TE schemes with respect to their impact on application performance. We chose ns-2 as it is well-suited for simulating thousands of flows in an ISP network at the packet level while also incorporating transport and application behavior in a fine-grained manner.

3.2.1 Simulating traffic matrices in ns-2

Figure 3.2.1 illustrates the experimental process. Each simulation has three inputs: (1) *ISP Topology* (2) a sequence of *File Arrivals* at each node based on the current *TM* (3) *Routing*, as computed using a *TE* scheme.

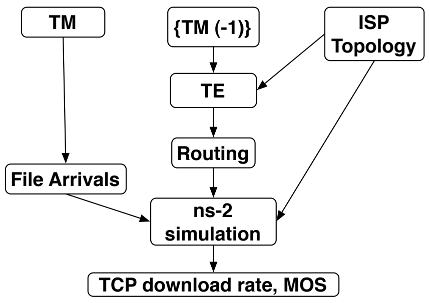


Figure 3.2. Block diagram of experiment process

We construct an ISP network topology from our dataset consisting of PoP-level ISP topology maps. PoPs are represented as nodes and links between these nodes are the backbone links of the ISP. Each PoP node has a number of users connected to it via separate access links. Each PoP node also has five server nodes connected to it via high capacity links that serve files to users. The number of user nodes in our simulation ranges from 300-6000 nodes and the capacity of backbone links varies from 50Mbps to 1Gbps.

We translate a *TM* to a sequence of *File Arrivals* as follows. Suppose the traffic matrix entry from A to B is 100 Mbps and the duration being simulated is 200 seconds. During the experiment interval, we generate a sequences of file arrivals from A to B whose total size is 100Mbps $\times$ 200 seconds and the sizes are chosen from a realistic distribution.

A traffic engineering scheme *TE* calculates routing for *TM* based on a set of matrices *TM(-1)* which consists of either the current traffic matrix (for Optimal) or a set of matrices from the previous traffic engineering *epoch* (for other TE schemes). The length of the epoch depends on *TE*, e.g., the epoch length for OptWt is 3 hours and for COPE is 1 day. When *TE* yields a routing that splits flows across multiple paths between two nodes, the number of files assigned to each path is proportional to the flow along that path. We use the source routing option in ns-2 to pin a file to a path. We note that the link utilization values obtained using this ns-2 methodology are consistent with those obtained using a simple linear program with the difference being at most 0.1.

In order to make the simulation complexity tractable, we scale down the topology and matrices. ISP backbone link capacities run into tens of Gbps. Simulating such a network at scale even for 100 seconds would require sending data on the order of terabytes (or equivalently, a million 100KB files). Experimentally, we find that simulating at a tenth of this scale, i.e., 100K files, is feasible given the computational and memory constraints of our machines. A typical scale in our simulation is 1/20, i.e., we simulate the backbone link with 1/20 the capacity and also scale down the traffic between each source-destination pair accordingly.

ISP	Nodes	Links	TM Duration
Abilene	12	30	5min
Geant	22	68	15min
US-ISP	-	-	1hr

Figure 3.3. ISP Data

BW (Mbps)	US users %	Europe users %
0.25	4.9	1.5
2.0	38.1	26.2
5.0	32.4	57.8
10.0	20.0	14.5
20.0	4.6	-

Figure 3.4. Bandwidth Distribution

3.2.1.1 ISP topologies and traffic matrices

We use datasets from the following three ISPs for our experiments:

(1) **Abilene**, from the publicly available Abilene ISP data [112]. (2) **Geant**, the un-anonymized version of the Geant topology obtained from the TotemData [94] project personnel. (3) **US-ISP**, a large Tier-1 ISP topology obtained from authors of [122]. TMs for all ISPs were logged in the period from 2004-2005. Figure 3.3 shows number of nodes, number of links, and the interval at which TMs are logged for each ISP. The number of nodes and links for US-ISP is proprietary information.

3.2.1.2 Simulation parameters

Unless otherwise stated, we choose the following parameters for all of our simulations. Our goal is to choose parameters that are close to realistic values for ISPs.
Scale: We experiment with Abilene, Geant and US-ISP datasets at scales 1/10, 1/20 and 1/100 respectively. These are the largest scales we can experiment with for each network given our computational constraints.
Duration: The simulation duration for most experiments is 300 seconds. We verified that running the simulations for longer durations did not qualitatively affect our results. Note that the duration here refers to the real time being simulated in ns-2, not the system time required to run the simulation.
Bandwidth of users: We use the bandwidth distribution of Internet users from the "State of the Internet Report" [87] released by Akamai, one of the largest commercial content distribution networks in operation today. Figure 3.4 tabulates this data for US and Europe.
File sizes: We simulate three file sizes of 100KB, 1MB and 10MB respectively contributing to 8%, 3% and 89% of the total traffic respectively. These values are the fractions of traffic due to small files (<200KB), medium size files (200KB to 2MB), and large files (>2MB) in the Internet. We obtained these numbers by collating data from multiple sources [110, 61, 59, 120].
Link delay: We calculate the propagation delay of backbone links from geographic distances between nodes for Geant and US-ISP. For Abilene, we measure the propagation delay of backbone links using *traceroute* and *ping* between PlanetLab [93] nodes in cities where the PoPs are located. All links use drop-tail queuing.

File inter-arrival time: We assume an exponential distribution of file inter-arrival times.

3.2.1.3 Computational resources

We use a shared cluster of 60 machines. Each machine has a 8-Core Intel Xeon processor and 16GB of memory. Each ns-2 simulation consists of 300–500s of simulated time and 10K to 200K file downloads, which results in a memory footprint of up to 10GB and takes between 1 to 48 hours to complete.

3.2.2 Traffic engineering schemes

We select a subset of TE schemes reflecting a variety of proposed approaches in the literature.

Optimal, the minimum MLU TE scheme for a TM. We consider it as being representative of *online* TE schemes.

InvCap, a simple routing scheme that does not "engineer" traffic, but instead simply relies on shortest-path routing using the inverse of the link capacity as the link weight. InvCap is a common default routing protocol supported by popular commercial router vendors [88].

OptWt, a shortest-path routing algorithm using link weights computed using a heuristic algorithm to optimize a cost function [50]. We use its implementation in the Totem Toolbox [94]. Typically, ISPs recompute routes a few times a day based on a set of measured TMs, so we simulate OptWt by computing a new routing every 3 hours based on the average of matrices in the past 3 hours.

MPLS, a TE scheme that minimizes the MLU in an *offline* manner. Similar to OptWt, MPLS recomputes a new routing once every 3 hours based on average of TMs in past 3 hours.

COPE, a TE scheme that minimizes the common-case MLU while limiting the worst-case MLU caused by unpredictable spikes in the traffic matrix. We use the authors' implementation and parameters settings, and recompute routes once a day based on the previous day's TMs as in [117].

MinDelay scheme minimizes average latency for all traffic, unlike the schemes above which optimize link utilization based metrics. Specifically, the objective function MinDelay optimizes is $\sum_{e \in E} d_e T_e$, where E is the set of all links, d_e is the propagation delay of link $e \in E$, and T_e is the total traffic (in bits/sec) for $e \in E$. We evaluate MinDelay to answer whether end-to-end application performance improves if we optimize latency instead of MLU. We constrain the linear program so that MLU does not exceed 0.6, thereby ensuring that high link utilization does not hurt application performance. Like Optimal, MinDelay has perfect knowledge of traffic matrix.

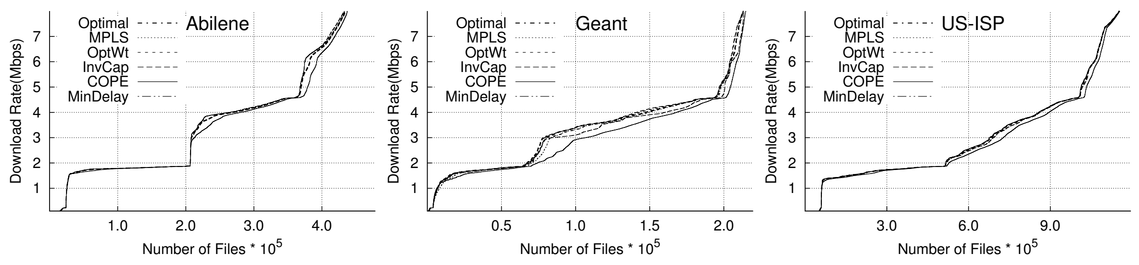


Figure 3.5. Download rate CDFs for all TE schemes are near identical except COPE which has slightly lower performance

3.3 Application performance

In this section, we present a comparative analysis of the impact of different TE schemes on end-to-end application performance. A summary of our findings is as follows. First, all TE schemes including InvCap show nearly identical application performance for TCP and UDP traffic. Second, different TE schemes do achieve different MLUs as expected, suggesting that MLU is a poor predictor of application performance. Third, COPE consistently performs slightly worse than all other schemes in TCP throughput, suggesting that accounting for unpredictable variations in traffic hurts the common case application performance.

3.3.1 TCP performance

We simulate TMs from 2 days of data for each ISP. For each day, we simulated 50 matrices measured at 5-minute intervals for Abilene, 25 matrices measured at 15-minute intervals for Geant, and 24 matrices measured hourly for US-ISP. We present results for the second day. The metric of application performance is the *download rate* of files using TCP, where the file arrival workload is generated using the traffic matrices as described in Section 3.2.1.

Figure 3.6 shows the mean download rate of files, where the average is across all files across all of the simulated matrices for each TE scheme. We make three observations from this graph. First, all schemes achieve nearly same mean download rates with the exception of COPE that is consistently worse by up to 10%. Next, Optimal (the leftmost bar in each group) is not always the best as minimizing MLU is not the same as optimizing TCP performance. Finally, MinDelay (the leftmost bar in each group) that optimizes latency performs the same as other TE schemes that optimize link utilization based metrics.

Figure 3.5 shows the corresponding CDFs for the mean download rates in Figure 3.6. The CDFs show that the near-identical TCP performance achieved by all TE schemes is not an artifact of presenting a specific statistic such as the mean, but is reflected by the entire distribution. All distributions show a stepwise increase which suggests that access links are a bottleneck for a significant fraction of file

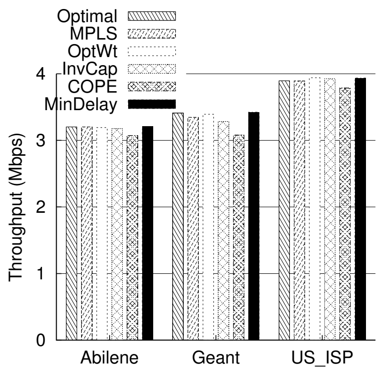


Figure 3.6. Mean download rates

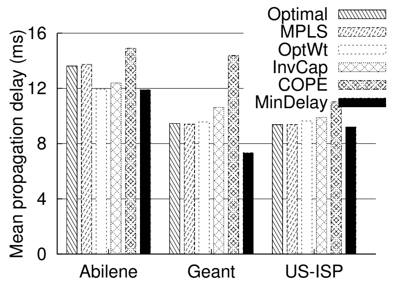


Figure 3.7. COPE has the highest propagation delay among TE schemes

transfers. This observation partly explains why MinDelay fails to improve TCP throughput over other schemes: TCP throughput cannot be improved for flows bottlenecked at access link even if MinDelay scheme reduced the RTT for these flows.

3.3.1.1 MLU vs. TCP performance

To further investigate the results in Figure 3.6 and Figure 3.5, we analyze the empirically observed MLU for all TE schemes in the experiments. Figure 3.8 plots the MLUs for all matrices considered. For US-ISP the MLU data is proprietary, so we present the ratio of MLU with respect to Optimal. As expected, different TE schemes do show substantially different MLUs. For example, the MLU for InvCap and OptWt is up to twice the MLU of Optimal in some cases; MinDelay has a MLU of 0.6 on Geant, which is more than twice of Optimal's MLU. These results suggest that MLU is a poor predictor of download rate performance: schemes with near-identical TCP throughput have very different MLUs, and COPE despite achieving near-optimal MLU consistently shows sub-optimal TCP throughput.

The main reason why MLU does not affect download rate is because queuing delay and loss rates are negligible until link utilization reaches a threshold. In our experiments, link utilization below 0.7 causes near negligible loss rates and queuing delays. Since the MLUs on most of the traffic matrices are below this value, loss rates on backbone links minimally impact the throughput of file downloads. These observations are consistent with a recent study on Level-3 ISP network [25] showing that loss rates on backbone links are zero even at 95% link utilization. This threshold is expected to be higher for actual backbone traffic as our experiments are at scale 1/10 or smaller. At larger scales, there would be more concurrent flows resulting in less bursty traffic and lower loss rates.

The second reason why MLU hardly impacts the average download rate as well as the distribution is because it is largely determined by the traffic of only one link. Even under high MLU, the rest of the network may not be congested.

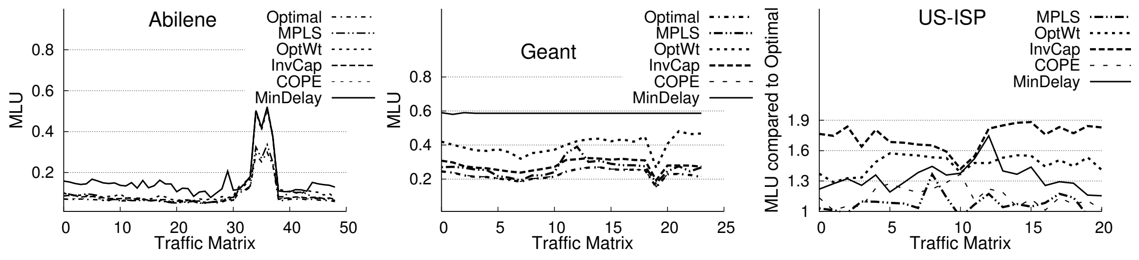


Figure 3.8. TE schemes differ as much as 2× in MLU

File download rates are affected only for flows on this link, which may be a tiny fraction of the total traffic.

3.3.1.2 The price of predictability

Why is COPE's performance consistently worse than the other schemes? To investigate this, we analyzed the propagation delays of routes computed by COPE. Given uniformly low loss rates and queueing delays, propagation delays primarily determine TCP performance.

Figure 3.7 shows the path delay averaged across all files and across all matrices for the different TE schemes. COPE has a significantly higher delay compared to all other schemes. We attribute this phenomenon to COPE's optimization approach, which engineers for unpredictable spikes in traffic demands. Specifically, COPE attempts to bound the worst-case MLU for any traffic matrix similar to oblivious routing like schemes [20]. COPE intentionally routes some traffic along longer paths so as to leave room for occasional traffic spikes along shorter paths. While this approach makes COPE robust with respect to MLU under rare spikes in traffic, it comes at the cost of hurting common-case application performance. Although we have not experimented with other oblivious routing schemes, these results suggest that any oblivious routing scheme that attempts to optimize MLU, e.g., [20], is likely to incur a similar penalty in application performance in the common case.

3.3.2 UDP performance

3.3.2.1 Measuring UDP performance

We assume that the loss rate and the queuing delay on each link for UDP traffic is the same as that measured during experiments with TCP traffic. This assumption is reasonable as TCP accounts for over 90% of Internet traffic [53]. We calculate the loss rate and delay for a path by combining the loss rates of links along the path; we compute the delay by summing the propagation and queuing delay of links along the path.

We compare performance of VoIP traffic (which uses UDP) using Mean Opinion Score (MOS). MOS is an industry standard VoIP call quality metric for which a score of above 4 is considered good and below 3 is considered bad. We calculate MOS using the formula in [35] which calculates MOS given the loss rate and delay for a path.

We calculate MOS for VoIP calls between all pairs of source and destination PoP nodes in an ISP. First, we measure loss rates and queuing delay on backbone links for each 10-second interval. For each interval, we calculate the MOS for a path based on its end-to-end loss rate and delay. The mean MOS for a path is the average value of MOS over all intervals. For TE schemes that split traffic across multiple paths between a source-destination pair, the mean MOS for a source-destination node pair is calculated as the weighted average of mean MOS weighted by the fraction of the traffic split along each path between the node pair. We similarly calculate the 5th percentile MOS for a source-destination pair by taking the weighted average of 5th percentile MOS values for all its paths.

3.3.2.2 Results

We obtain a distribution of mean MOS values for a TE scheme by combining mean MOS values for all pairs of source and destination nodes for all traffic matrices. We find that the minimum and the maximum values of mean MOS for all TE schemes are in the range (4.08, 4.14) for Abilene, (4.07, 4.14) for Geant and (4.08, 4.14) for US-ISP. The range of values for 5th percentile MOS are (4.07, 4.13) for Abilene, (4.08, 4.14) for Geant and (4.05, 4.14) for US-ISP. MOS scores for all schemes are always above 4.0 and the differences between different TE schemes is at most 0.1. These results are not surprising since loss rates and queuing delay are near-negligible for most links in the network. Furthermore, MOS is not very sensitive to few milliseconds difference in propagation delay among TE schemes.

3.4 Capacity and location diversity

The results in the previous section may seem unsurprising—different TE schemes yield nearly identical application performance simply because today's low traffic demand levels obviate the need to engineer traffic. However, in this section, we show that similar conclusions hold when we compare TE schemes with respect to their potential capacity, i.e., their ability to accommodate surges in traffic demand in the future.

The key factor that explains our unexpected findings is location diversity, i.e., the ability to download content from multiple locations. Our main findings are that (1) location diversity can significantly increase the capacity (by up to 2×) achieved by all engineering schemes; (2) even a modest amount of location diversity (e.g., the ability to download content from two locations) enables all engineering schemes to achieve near-Optimal capacity; (3) with location diversity even simple routing scheme of InvCap has at most 30% less capacity compared to Optimal.

3.4.1 Empirically measuring capacity

Our metric of capacity is the SPF, i.e., the maximum surge in demand that can be satisfied (as defined formally in Section 3.1.3.1). Analytically determining whether an engineering scheme can satisfy a projected demand is difficult as it requires us to accurately model application adaptation to location diversity, so the SPF must be determined empirically. In our experiments, we use a metric called *maximum input output difference* (or Max-IO-Diff) to determine whether a given demand can be satisfied. For each node, the *input* is the total traffic (bits/sec) requested by that node, while the *output* is the total traffic received by that node. Max-IO-Diff is defined as the maximum across all nodes of the relative difference between the input and output, i.e., *(input - output)/input*. If Max-IO-Diff is measured to be less than 0.1, then the demand is considered as satisfiable. We allow for a small difference in order to account for measurement error as well as to account for bursts in demand over the measurement duration.

Max-IO-Diff helps clearly distinguish workloads that can be satisfied. For example, in Figure 3.9, we show a Max-IO-Diff profile for five experiments at surge factors of 1, 2, 3, 4 and 5 for a Geant TM with InvCap routing. The graph shows the Max-IO-Diff measured at intervals of 10 seconds throughout the simulation. We ignore the first 50 seconds of simulation as the input significantly exceeds output at the start of simulation. We observe that beyond the initial period of fluctuation, Max-IO-Diff is relatively stable and below 0.1 for surge factors 1–3 that can be satisfied, but significantly higher for surge factors 4 and 5 that can not be satisfied.

3.4.2 Simulating location diversity

Figure 3.10 illustrates the experimental process with location diversity. The lower half is similar to Figure 3.2.1 with two differences. First, to incorporate location diversity, we modify the procedure to transform *TM* to *File Arrivals* as follows. As in Section 3.2, we first transform PoP-to-PoP entries in *TM* to a sequence of file download requests. However, instead of downloading each file from just that one location, k-1 additional randomly chosen source locations are introduced so as to emulate a location diversity of k. The file is downloaded in parallel from all k locations using parallel TCPs. The download is considered complete when the total bytes downloaded across all k locations equals the size of the file.

Second, application adaptation to location diversity changes the input to *TE* as indicated by the block *Transformed TM(-1)* that is obtained as follows. Let *TM(-1)* and *TM(-2)* respectively denote the (set of) matrix(ces) in the last and last-to-last epochs. Recall that *TE* determines the length of the epoch (0 for Optimal, 3 hours for OptWt and MPLS and a day for COPE). *Transformed TM(-1)* is generated by the top simulation that takes as input the file arrivals obtained from *TM(-1)* and *Routing (-1)*. The latter is obtained by applying *TE* to *TM(-2)* in the previous epoch.

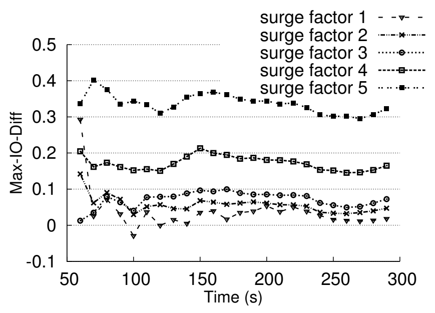


Figure 3.9. Profile of Max-IO-Diff at increasing surge factors for a Geant TM

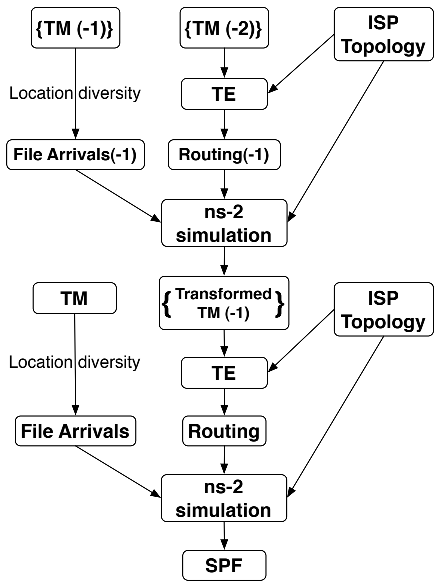


Figure 3.10. Block diagram of experiment process with location diversity

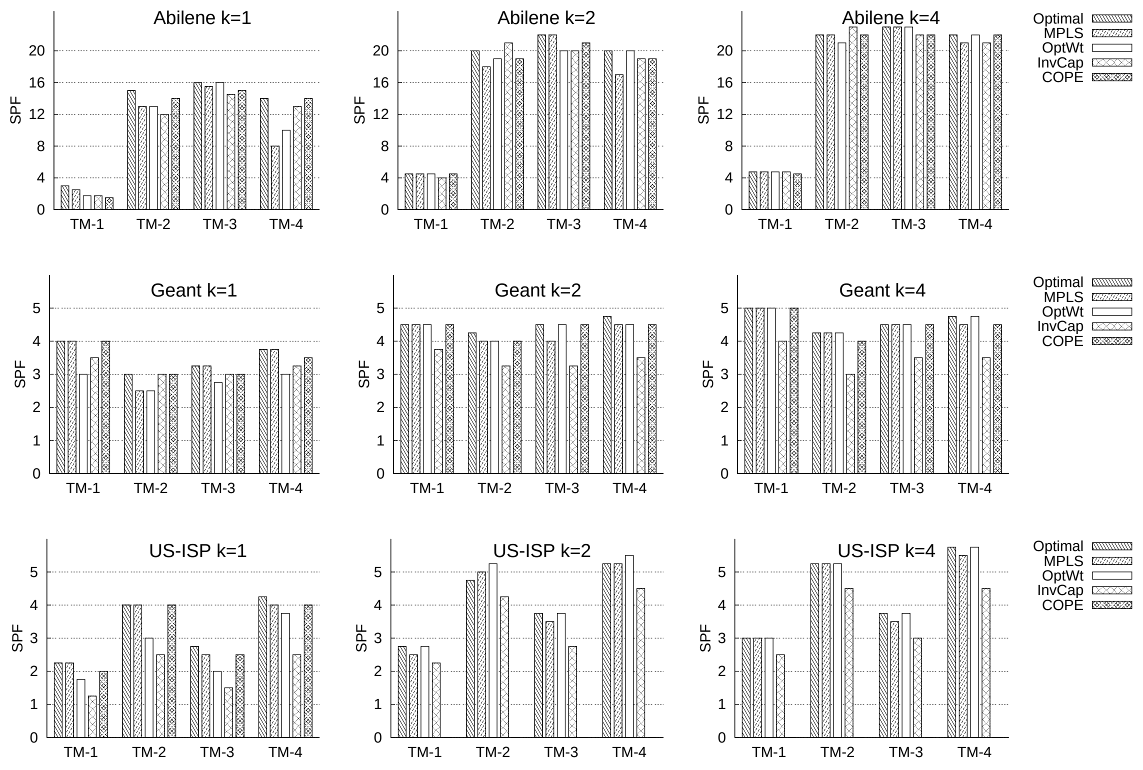


Figure 3.11. Comparison of SPF among TE schemes for different levels of location diversity; SPF values are obtained using ns-2 simulations

This two-step simulation is intended to approximate the interaction of TE and application adaptation to location diversity that changes the TM.

3.4.3 Experimental procedure

The experiments to determine SPF involve a computationally intensive search across many different surge factors for each matrix. Furthermore, at high surge factors, the number of ns-2 data structures required to simulate ongoing parallel TCP connections becomes prohibitively high. So for computational tractability, we selected 4 matrices each from one day of data of each ISP. The matrices were selected randomly, one from each 6-hour duration during the day. For each matrix and each engineering scheme, we conduct an experiment at each value of the surge factor starting from 1 in increments of 0.25 until the capacity point is reached, i.e., the Max-IO-Diff value exceeds 0.1. Each experiment is run until the Max-IO-Diff value stabilizes or 300 seconds, whichever is greater.

3.4.4 Capacity increase with location diversity

In Figure 3.11, we present the SPF values obtained using ns-2 simulations for the selected TMs. We compared all TE schemes for three levels of location diver-

TE/Optimal	k=1	k=2	k=4
Optimal/Optimal	1	1	1
MPLS/ Optimal	0.89	0.98	0.99
OptWt/Optimal	0.73	0.99	0.99
InvCap/Optimal	0.91	0.86	0.85
COPE/Optimal	0.91	0.99	0.98

Figure 3.12. Comparison of SPF values

sity: $k = 1, 2$ and 4. Note that we do not present the results for COPE for US-ISP (k =2 and k = 4), since the implementation of COPE's algorithm failed to compute a feasible set of routes even after 12 hours of simulation time (1 million iterations) after which we aborted the simulation. We have used authors' implementation of the algorithm and communication with them confirmed that indeed in some cases COPE's implementation can take a long time to terminate. This happens in cases where barrier-crossover method to solve a linear program fails and COPE instead uses simplex method which is much slower.

The average capacity increase for Optimal from $k = 1$ to $k = 4$ is 1.41× and from $k = 1$ to $k = 2$ is 1.31×. Optimal is the maximum SPF for a network with no location diversity ($k = 1$). This shows that a network with location diversity of $k = 4$ has 40% greater capacity than a network with no location diversity. Even location diversity of $k = 2$ gives 75% of capacity increase obtained from location diversity of $k = 4$.

Location diversity enables all TE schemes to achieve near-optimal capacity. In Figure 3.12 we compare the SPF of Optimal to that of other TE schemes. The statistic presented is ratio of SPF of TE scheme to SPF of Optimal for the same level of location diversity averaged over all TMs. Except InvCap, all TE schemes have SPF within 2% of Optimal for $k = 4$ as well as $k = 2$. Figure 3.11 shows that with location diversity any TE scheme has at most 10% capacity difference compared to Optimal. On average InvCap has 15% less capacity compared to Optimal for a location diversity of $k = 4$. In the worst case InvCap achieves a capacity that is 30% less than Optimal (Figure 3.11, Geant k = 2).

The above result calls into question the usefulness of online TE schemes. In today's Internet, offline TE schemes such as OptWt or MPLS are commonly used. It is believed that these schemes are sub-optimal and online TE schemes (e.g., TeXCP, MATE etc.) can achieve near-optimal capacity. However, our results suggest that application adaptation to location diversity results in near-optimal SPF for all TE schemes. Even the shortest-path routing scheme, OptWt, achieves the same SPF as TE schemes employing MPLS for flow splitting.

3.4.4.1 Other results

We briefly summarize other experimental results deferred to a technical report [11]. First, SPF increases in a concave manner with the fraction of traffic that can leverage location diversity. Even if only half of the traffic has location diversity, it suffices to capture over 90% of the potential increase in SPF for each TE scheme, and the SPFs achieved by different TE schemes continues to be less that 5%. Second, the "near-optimality" of capacity achieved by all TE schemes is reflected not only in their SPFs but in application performance metrics as well, i.e., TCP download rates and MOS scores (in the mean as well as across various percentiles) degrade similarly for all TE schemes as the demand approaches the SPF capacity point. As expected, application performance starts to dip earlier under InvCap as its SPF is somewhat lower than TE schemes. Thus, these results also suggest that, unlike link utilization metrics, SPF is a sound empirical metric to measure how effectively a TE scheme can accommodate load surges under location diversity.

3.5 Conclusion

Our application-centric focus and empirical evaluation methodology reveal unexpected results that challenge conventional wisdom in traffic engineering. We find that link utilization, the most widely used metric to evaluate TE, is a poor predictor of application performance. Under typical Internet load conditions, all TE schemes and even static routing achieve nearly identical application performance despite achieving vastly different MLUs. In fact, engineering link utilization in order to accommodate unexpected traffic spikes can actually hurt common-case application performance. More intriguingly, we find that application adaptation to location diversity, or the ability to download content from multiple locations, eliminates differences in the achieved capacity of all TE schemes including "optimal" TE. With location diversity, even static routing achieves a capacity that is at most 30% (and typically significantly less) worse than the optimal. Taken together, our findings suggest that it matters little which TE scheme is used (or whether TE is used at all) at today's traffic levels as well as under reasonable projections of increased demand. A provocative message to network operators is to stop engineering traffic and let end-users do the work for them.

CHAPTER 4

NETWORK CDNS

Content delivery networks (CDNs) today provide a core service that enterprises use to deliver web content, downloads, streaming media, etc. to a global audience of their end-users. The traditional and somewhat simplified, tripartite view of content delivery involves three sets of entities as shown in Figure 4.1. The *content providers* (e.g., media companies, news channels, e-commerce sites, software distributors, enterprise portals, etc.) produce the content and wish to provide a high-quality experience to their end-users. The *networks* (e.g., telcos such as AT&T, multi-system operators such as Comcast, and ISPs) own the underlying network infrastructure and are responsible for provisioning capacity and routing traffic demand. Finally, the *CDNs* (e.g., Akamai, Limelight) are responsible for optimizing content delivery to end-users on behalf of the content providers, residing as a global, distributed overlay service.

Recent powerful trends are reshaping the simplified tripartite view of content delivery. A primary driver is the torrid growth of video [85, 33] and downloads traffic on the Internet. For example, a single, popular TV show with 50 million viewers, with each viewer watching an HD-quality stream of 10 Mbps, generates 500 Tbps of network traffic! The increasing migration of traditional media content to the Internet and the consequent challenges of scaling the network backbone to accommodate that traffic has necessitated the evolution of *network CDNs* (or NCDNs)[1] that vertically integrate CDN functionality such as content caching and redirection with traditional network operations [65, 83, 75, 21, 115] (refer Figure 4.1). A second economic driver of NCDNs is the desire of networks to further monetize the "bits" that flow on their infrastructure by contracting directly with content providers, or to offer value-added service packages to their own end-user

[1]NCDNs are sometimes referred to as Telco CDNs, or Carrier CDNs. Further, they are referred to as a Licensed CDN when a pure-play CDN such as Edgecast[30] licenses the CDN software to a network to create an NCDN.

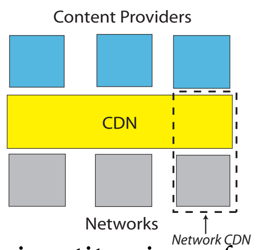


Figure 4.1. A tripartite view of content delivery.

subscribers (e.g., Verizon's recent offering that delivers HBO's content to FIOS subscribers [114].

As NCDNs control both the content distribution and network infrastructure, the costs and objectives of their interest are different both from a traditional CDN and a traditional ISP. In particular, an NCDN is in a powerful position to place content in a manner that "shapes" the traffic demand so as to optimize both network cost and user-perceived latency. Indeed, several recent works have alluded to the benefits of such joint optimization strategies in the context of cooperative or competitive interaction between ISPs and content providers [121, 42, 69, 55]. On the surface, an NCDN would appear to be the perfect setting for fielding joint optimization strategies as it eliminates potentially conflicting competitive interests. Nevertheless, NCDNs today continue to treat content distribution and traffic engineering concerns separately, operating the former simply as an overlay.

This disparity raises several research questions that form the focus of this chapter. How should an NCDN determine content placement, network routing, and request redirection decisions so as to optimize network cost and user-perceived latency? How much benefit do joint optimization strategies yield over simpler strategies as practiced today, and does the benefit warrant the added complexity? How do content demand patterns and placement strategies impact network cost? How do planned strategies (i.e., using knowledge of recently observed demand patterns or hints about anticipated future demands) for placement and routing compare against simpler, unplanned strategies?

Our primary contribution is to empirically analyze the above questions for realistic content demand workloads and ISP topologies. To this end, we collect content request traces from Akamai, the world's largest CDN today. We focus specifically on on-demand video and large-file downloads traffic as they are two categories that dominate overall CDN traffic and are significantly influenced by content placement strategies. Our combined traces consist of a total of 28.2 million requests from 7.79 million unique users who downloaded a total of 1455 Terabytes of content across the US over multiple days. Our main finding based on trace-driven experiments using these logs and realistic ISP topologies is that *simple, unplanned strategies for placement, routing, and redirection of NCDN content are better than sophisticated joint-optimization approaches*. Specifically,

- For NCDN traffic, simple unplanned schemes for placement and routing (such as least-recently-used and InverseCap) yield significantly lower (2.2–$17\times$) network cost and user-perceived latency than a joint-optimal scheme with knowledge of the previous day's demand[2].

- NCDN traffic demand can be "shaped" by simple placement strategies so that traffic engineering, i.e., optimizing routes with knowledge of recent traffic matrices, hardly improves network cost or user-perceived latency over unplanned routing (InverseCap).

[2]We use the term "optimal" when placement or routing is the solution of an optimization problem, but the solution may not have the lowest cost (for reasons detailed in Section 4.4.3.1)

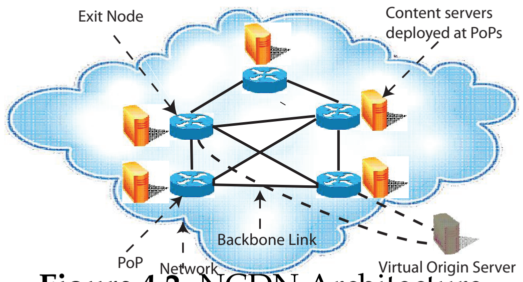


Figure 4.2. NCDN Architecture

- For NCDN traffic, unplanned placement and routing is just 1%-18% sub-optimal compared to a joint-optimal placement and routing with perfect knowledge of the next day's demand at modest storage ratios (≈ 4).
- With a mix of NCDN and transit traffic, traffic engineering does lower network cost (consistent with previous studies), but the value of traffic engineering substantially diminishes as the relative volume of NCDN traffic begins to dominate that of transit traffic.

In the rest of this chapter, we first overview the NCDN architecture highlighting why it changes traditional ISP and CDN concerns (Section 4.1). Next, we formalize algorithms that jointly optimize content placement and routing in an NCDN (Section 4.2). We then describe how we collected real CDN traces (Section 4.3) and evaluate our algorithms using these traces and real ISP topologies (Section 4.4).

4.1 Background and motivation

A typical NCDN architecture, as shown in Figure 4.2, resembles the architecture of a global CDN but with some important differences. First, the content servers are deployed at points-of-presence (PoPs) within the network rather than globally across the Internet as the NCDN is primarily interested in optimizing content delivery for its own customers and end-users. Second, and more importantly, the NCDN owns and manages the content servers as well as the underlying network. Content providers that purchase content delivery service from the NCDN publish their content to origin servers that they maintain external to the NCDN itself.

Each PoP is associated with a distinct set of end-users who request content such as web, video, downloads etc. An end-user's request is first routed to the content servers at the PoP to which the end-user is connected. If a content server at that PoP has the requested content in their cache, it serves that to the end-user. Otherwise, if the requested content is cached at other PoPs, the content is downloaded from a nearby PoP and served to the end-user. If the content is not cached in any PoP, it is downloaded directly from the content provider's origin servers.

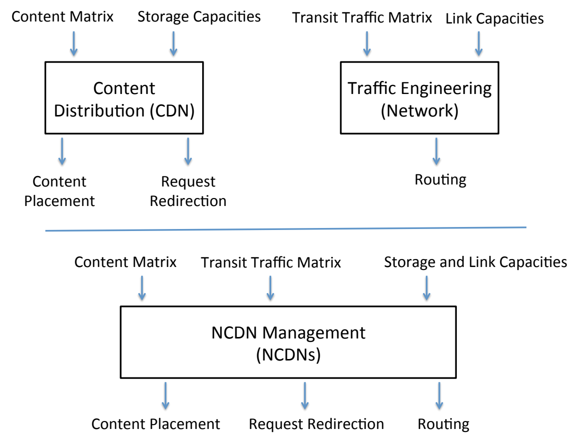


Figure 4.3. (Top) Traditional formulation with content distribution and traffic engineering optimized separately. (Bottom) Our new formulation of NCDN magamenent as a joint optimization.

4.1.1 Why NCDNs change the game

Managing content distribution as well as the underlying network makes the costs and objectives of interest to an NCDN different from that of a traditional CDN or a traditional ISP. Figure 4.3 (top) shows the traditional concerns of content distribution and traffic engineering as addressed by a traditional CDN and a traditional ISP respectively, while Figure 4.3 (bottom) shows the combined concerns that an NCDN must address. We explain these in detail below.

4.1.1.1 Content distribution

A traditional CDN has two key decision components—*content placement* and *request redirection*—that seek to optimize the response time perceived by end-users and balance the load across its content servers. Content placement decides which objects should be cached at which nodes. An object may be stored at multiple nodes in the network or not stored in the network at all and be served from the origin server instead. Request redirection determines which server storing a replica of the object is best positioned to serve it.

Content placement schemes can either be *planned* or *unplanned*. A planned scheme calculates placement using a *content matrix* that specifies the demand for each content at each location. The content matrix is learned by monitoring a recent history of system-wide requests and possibly including hints, if any, from content providers about anticipated demand for some objects. A planned scheme uses a recent content matrix to decide on a placement periodically (e.g., daily) but does not alter its placement in between. In contrast, an unplanned scheme can continually alter its placement potentially even after every single request. A simple and widely used example of an unplanned placement scheme is LRU, where each server evicts the least-recently-used object from its cache to make room for new ones.

4.1.1.2 Traffic engineering

A key component of ISP network operations is traffic engineering, which seeks to route the traffic demands through the backbone network so as to balance the load and mitigate hotspots. Traffic engineering is commonly viewed as a routing problem that takes as input a *traffic matrix*, i.e., the aggregate flow demand between every pair of PoPs observed over a recent history, and computes routes so as to minimize a network-wide cost objective. The cost seeks to capture the severity of load imbalance in the network and common objective functions include the maximum link utilization (MLU) or a convex function (so as to penalize higher utilization more) of the link utilization aggregated across all links in the network [50]. ISPs commonly achieve the computed routing either by using shortest-path routing (e.g., the widely deployed OSPF protocol [50]) or by explicitly establishing virtual circuits (e.g., using MPLS [45]). ISPs perform traffic engineering at most a few times each day, e.g., morning and evening each day [51].

Routing can also be classified as *planned* or *unplanned* similar in spirit to content placement. Traffic engineering schemes as explained above are implicitly planned as they optimize routing for recently observed demand. To keep the terminology simple, we also classify online traffic engineering schemes [70, 45] (that are rarely deployed today) as planned. In contrast, unplanned routing schemes are simpler and rely upon statically configured routes [20, 22], e.g., InverseCap is a static shortest-path routing scheme that sets link weights to the inverse of their capacities; this is a common default weight setting for OSPF in commercial routers [51].

4.1.1.3 NCDN management

As NCDNs own and manage the infrastructure for content distribution as well as the underlying network, they are in a powerful position to control all three of placement, routing, and redirection (Figure 4.3). In particular, an NCDN can place content in a manner that "shapes" the traffic demands so as to jointly optimize both user-perceived latency as well as network cost.

The prospect of jointly optimizing placement, routing, and redirection raises several natural questions. How much additional benefit does such joint optimization offer compared to treating CDN and ISP concerns independently as practiced today? Is the added complexity of joint optimization strategies worth the benefit? Which of the three—placement, routing, and redirection—is the most critical to reducing network cost and user-perceived latency? How sensitive are these findings to characteristics of the content workload (e.g., video vs. download traffic)?

4.2 NCDN management strategies

To answer the above questions, we develop an optimization model for NCDNs to jointly optimize placement, routing, and redirection so as to optimize network cost or user-perceived latency. We formulate the optimization problem as a mixed-integer program (MIP), present hardness and inapproximability results, and discuss approximation heuristics to solve MIPs for realistic problem sizes.

Input variables and descriptions	
V	Set of nodes where each node represents a PoP
E	Set of edges where each link represents a communication link
o	Virtual origin node that hosts all the content in K
X	Set of exit nodes in V
D_i	Disk capacity at node $i \in V$ (in bytes)
C_e	Capacity of link $e \in E$ (in bits/sec)
K	the set of all content accessed by end-users
S_k	Size of content $k \in K$.
T_{ik}	Demand (in bits/sec) at node $i \in V$ for content $k \in K$
Decision variables and descriptions	
α	MLU of the network
z_k	Binary variable indicating whether one or more copies of content k is placed in the network
x_{jk}	Binary variable indicating whether content k is placed at node $j \in V \cup \{o\}$
f_{ij}	Total traffic from node j to node i
f_{ije}	Traffic from node j to node i crossing link e.
t_{ijk}	Traffic demand at node $i \in V$ for content $k \in K$ served from node $j \in V \cup \{o\}$

Table 4.1. List of input and decision variables for the NCDN problem formulation.

4.2.1 NCDN model

Table 1 lists all the model parameters. An NCDN consists of a set of nodes V where each node represents a PoP in the network. The nodes are connected by a set of directed edges E that represent the backbone links in the network. The set of content requested by end-users is represented by the set K and the sizes of content are denoted by $S_k, k \in K$. The primary resource constraints are the link capacities $C_e, e \in E$, and the storage at the nodes $D_i, i \in V$. We implicitly assume that the content servers at the PoPs have adequate compute resources to serve locally stored content.

A *content matrix* (CM) specifies the demand for each content at each node. An entry in this matrix, $T_{ik}, i \in V, k \in K$, denotes the demand (in bits/second) for content k at node i. CM is assumed to be measured by the NCDN a priori over a coarse-grained interval, e.g., the previous day. The infrastructure required for this measurement is comparable to what ISPs have in place to monitor traffic matrices today.

Origin servers, owned and maintained by the NCDN's content providers, initially store all content published by content providers. We model origin servers using a single virtual origin node o external to the NCDN that can be reached via a set of exit nodes $X \subset V$ in the NCDN (Figure 4.2). Since we are not concerned with traffic engineering links outside the NCDN, we model the edges (x, o), for all $x \in X$, as having infinite capacity. The virtual origin node o always maintains a copy of all the requested content. However, a request for a content is served from the virtual origin node only if no copy of the content is stored at any node $i \in V$. In this case, the request is assumed to be routed to the virtual origin via the exit node closest to the node where the request was made (in keeping with the commonly practiced *early-exit* or *hot potato* routing policy).

ISP networks carry transit traffic in addition to NCDN traffic, which can be represented as a transit traffic matrix (TTM). Each entry in the TTM contains the volume of transit traffic between two PoPs in the network.

4.2.2 Cost functions

We evaluate NCDN-management strategies based on two cost functions. The first cost function is maximum link utilization (or MLU) which measures the effectiveness of traffic engineering in an NCDN. MLU is a widely used network cost function for traditional TE.

The second cost function models user-perceived latency and is defined as $\sum_{e \in E} X_e$, where X_e is the product of traffic on link e and its link latency $L(e)$. The latency of a link $L(e)$ is the sum of a fixed propagation delay and a variable utilization dependent delay. For a unit flow, link latency is defined as $L_e(u_e) = p_e(1 + f(u_e))$, where p_e is the propagation delay of edge e, u_e is its link utilization, and $f(u)$ is a piecewise-linear convex function. This cost function is similar to that used by Fortz and Thorup [50]. At small link utilizations (< 0.6), link latency is determined largely by propagation delay hence f is zero. At higher link utilizations (0.9 and above) an increase in queuing delay and delay caused by retransmissions significantly increase the effective link latency. The utilization-dependent delay is modeled as proportional to propagation delay as the impact of (TCP-like) retransmissions is more on paths with longer links. Since L_e is convex, a set of linear constraints can be written to constraint the value of X_e (as in [50]).

4.2.3 Optimal strategy as MIP

We present here a joint optimization strategy for NCDN-management formulated as a MIP. This formulation takes as input a content matrix, i.e., the demand for each content at each network point-of-presence (PoP), and computes content placement, request redirection and routing that minimizes an NCDN cost function while respecting link capacity and storage constraints. The decision variables for this problem are listed in Table 1. The MIP to minimize an NCDN cost function C (either MLU or latency) is as follows:

$$\min C \tag{4.1}$$

subject to

$$\begin{aligned}
\sum_{j \in V} t_{ijk} + t_{iok} &= T_{ik}, \quad \forall k \in K, i \in V && (4.2)\\
\sum_{k \in K} t_{ijk} &= f_{ij}, \; \forall j \in V - X, i \in V && (4.3)\\
\sum_{k \in K} t_{ijk} + \sum_{k \in K} \delta_{ij} t_{iok} &= f_{ij}, \; \forall j \in X, i \in V && (4.4)
\end{aligned}$$

where δ_{ij} is 1 if j is the closest exit node to i and 0 otherwise. Note that δ_{ij} is not a variable but a constant that is determined by the topology of the network, and hence constraint (4) is linear.

$$\sum_{p \in P(l)} f_{ijp} - \sum_{q \in Q(l)} f_{ijq} = \begin{cases} f_{ij} & \text{if } l = i, \\ -f_{ij} & \text{if } l = j, \\ 0 & \text{otherwise,} \end{cases} \quad \forall i, j, l \in V \qquad (4.5)$$

where $P(l)$ and $Q(l)$ respectively denote the set of outgoing and incoming links at node l.

$$\begin{aligned}
\sum_{i \in V, j \in V} f_{ije} &\leq \alpha \times C_e, \quad \forall e \in E && (4.6)\\
\sum_{k \in K} x_{ik} S_k &\leq D_i, \quad \forall i \in V && (4.7)\\
x_{ok} &= 1, \quad \forall k \in K && (4.8)\\
\sum_{i \in V} x_{ik} &\geq z_k, \quad \forall k \in K && (4.9)
\end{aligned}$$

$$x_{ik} \leq z_k, \quad \forall k \in K, i \in V \qquad (4.10)$$

$$\begin{aligned}
t_{ijk} &\leq x_{jk} T_{ik}, \quad \forall k \in K, i \in V, j \in V \cup \{o\} && (4.11)\\
t_{iok} &\leq T_{ik}(1 - z_k), \quad \forall k \in K && (4.12)
\end{aligned}$$

$$\begin{aligned}
x_{jk},\, z_k &\in \{0, 1\}, \quad \forall j \in V, k \in K\\
f_{ije},\, t_{ijk},\, t_{iok} &\geq 0, \quad \forall i, j \in V, e \in E, k \in K
\end{aligned}$$

The constraints have the following rationale. Constraint (2) specifies that the total traffic demand at each node for each content must be satisfied. Constraints (3) and (4) specify that the total traffic from source j to sink i is the sum over all content k of the traffic from j to i for k. Constraint (5) specifies that the volume of a flow coming in must equal that going out at each node other than the source or the sink. Constraint (6) specifies that the total flow on a link is at most α times capacity. Constraint (7) specifies that the total size of all content stored at a node must be less than its disk capacity. Constraint (8) specifies that all content is placed at the virtual origin node o. Constraints (9) and (10) specify that at least one copy of content k is placed within the network if $z_k = 1$, otherwise $z_k = 0$ and no copies of k are placed at any node. Constraint (11) specifies that the flow from a source to a sink for some content should be zero if the content is not placed at the source (i.e., when $x_{jk} = 0$), and the flow should be at most the demand if the content is placed at the source (i.e., when $x_{jk} = 1$). Constraint (12) specifies that if some content is placed within the network, the traffic from the origin for that content must be zero.

Updating the content placement itself generates traffic and impacts the link utilization in the network. For ease of exposition, we have deferred a formal description of the corresponding constraints to a techreport [12]. Finally, a simple extension to this MIP presented in a techreport [12] jointly optimizes routing given a TTM as well a CM. We have presented a CM-only formulation here as our findings (in Section 6.2) show that a joint optimization of the CM and TTM is not useful for NCDNs.

4.2.4 Computational hardness

Opt-NCDN is the decision version of the NCDN problem. The proofs for these theorems are presented in a techreport [12].

THEOREM 1. *Opt-NCDN is NP-Complete even in the special case where all objects have unit size, and all demands, link capacities, and storage capacities have binary values.*

COROLLARY 1. *Opt-NCDN is inapproximable to within a constant factor unless P = NP.*

4.2.5 Approximation techniques for MIP

As solving the MIP for very large problem scenarios is computationally infeasible, we use two approximation techniques to tackle such scenarios.

The first is a two-step local search technique. In the first step, we "relax" the MIP by allowing the integral variables x_{jk} and z_k to take fractional values between 0 and 1. This converts an MIP into an LP that is more easily solvable. Note also that the optimal solution of the relaxed LP is a lower bound on the optimal solution of the MIP. However, the LP solution may contain fractional placement of some of the content with the corresponding x_{jk} variables set to fractional values between 0 and 1. However, in our experiments only about 20% of the variables in the optimal LP solution were set to fractional values between 0 or 1, and the rest took integral values of 0 or 1. In the second step, we search for a valid solution for the MIP in the local vicinity of the LP solution by substituting the values for variables that were set to 0 or 1 in the LP solution, and re-solving the MIP for the remaining variables. Since the number of integer variables in the second MIP is much smaller, it can be solved more efficiently than the original MIP.

The second approximation technique reduces the number of unique content in the optimization problem using two strategies. First, we discard the tail of unpopular content prior to optimization. The discarded portion accounts for only 1% of

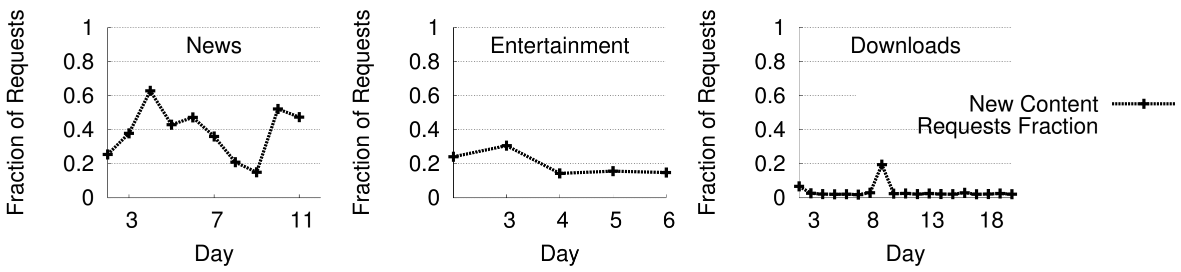


Figure 4.4. News and entertainment have a significant fraction of requests for new content on all days. Downloads has a small fraction of requests for new content on all days, except one.

all requests, but reduces the number of content by 50% or more in our traces. Second, we sample 25% of the content from the trace and, in our experiments, select trace entries corresponding only to the sampled content. These approximations reduce the number of content from tens of thousands to under 5000. An MIP of this size can be solved using local search in an hour by a standard LP solver [67] for the ISP topologies in our experiments.To check for any untoward bias introduced by the sampling, we also performed a small number of experiments with the complete trace and verified that our findings remain qualitatively unchanged.

4.3 Akamai CDN traces

To conduct a realistic simulation of end-users accessing content on an NCDN, we collected extensive traces of video and download traffic from Akamai as described below.

Video traces. Videos are the primary source of traffic on a CDN and are growing at a rapid rate [85, 33]. Our video trace consists of actual end-users accessing on-demand videos on the Akamai network over multiple days. To make the traces as representative as possible, we chose content providers with a whole range of business models, including major television networks, news outlets, and movie portals. The videos in our traces include a range of video types from short-duration video (less than 10 mins) such as news clips to longer duration (30 min to 120 min) entertainment videos representing TV shows and movies. In all, our traces represent a nontrivial fraction of the overall traffic on Akamai's media network and accounted for a total of 27 million playbacks of over 85000 videos, 738 TBytes of traffic, served to 6.59 million unique end-users around the US. Since we only had US-based network topologies with accurate link capacity information, we restricted ourselves to US-based traffic.

We collect two sets of video traces called *news trace* and *entertainment trace* respectively. The news trace was collected from a leading news outlet for an 11-day period in Sept 2011, and consists mostly of news video clips, but also includes a small fraction of news TV shows. The entertainment trace was collected for a 6 day period in January 2012, and includes a variety of videos including TV shows, clips of TV shows, movies and movie trailers from three major content providers.

The trace collection mechanism utilized a plugin embedded in the media player that is capable of reporting (anonymized) video playback information. Our traces include a single log entry for each playback and provides time of access, user id, the location of the user (unique id, city, state, country, latitude, and longitude), the url of the content, the content provider, the total length of the video (in time and bytes), the number of bytes actually downloaded, the playback duration, and the average bitrate over the playback session.

Downloads traces. The second largest traffic contributor in a CDN is downloads of large files over HTTP. These include software and security updates, e.g., Microsoft's Windows or Symantec's security updates, as well as music, books, movies, etc.. The large file downloads at Akamai typically use a client-side software called the download manager [86]. We collect extensive and anonymized access data reported from the download manager using Akamai's NetSession interface [68] for a large fraction of content providers for a period of a month (December 2010). Our traces represent a nontrivial fraction of the overall US-based traffic on Akamai's downloads network and accounted for a total of 1.2 million downloads, 717 TBytes of traffic, served to 0.62 million unique end-users around the US. Our traces provide a single log entry for each download and provide time of access, user id, location of the user (city, state, country, latitude, and longitude), the url identifier of the content, content provider, bytes downloaded, and file size.

Figure 4.4 shows the fraction of requests for new content published each day relative to the previous day for news, entertainment, and downloads traces. The news trace has up to 63% requests due to new content because the latest news clips generated each day are the most popular videos on the website. The entertainment trace also has up to 31% requests each day due to new content such as new episodes of TV shows, and the previews of upcoming TV shows. The downloads trace has only 2-3% requests due to new content on a typical day. However, on the 9th day of the trace major software updates were released, which were downloaded on the same day by a large number of users. Hence, nearly 20% requests on that day were for new content. The fraction of requests for new content impacts the performance of planned placement strategies as we show Section 4.4.

4.4 Experimental evaluation

We conduct trace-driven experiments to compare different NCDN-management strategies. Our high-level goal is to identify a simple strategy that performs well for a variety of workloads. In addition, we seek to assess the relative value of optimizing content placement versus routing; the value of being planned versus being unplanned and the value of future knowledge about demand.

4.4.1 Trace-driven experimental methodology

To realistically simulate end-users accessing content on an NCDN, we combine the CDN traces (in Section 4.3) with ISP topologies as follows. We map each content request entry in the Akamai trace to the geographically closest PoP in the ISP topology in the experiment (irrespective of the real ISP that originated the request). Each PoP has a content server as shown in Figure 4.2, and the request is served locally, redirected to the nearest (by hop-count) PoP with a copy, or to the origin as needed.

ISP topologies. We experimented with network topology maps from two US-based ISPs. First is the actual ISP topology obtained from a large tier-1 ISP in the US (referred to as US-ISP). Second is the Abilene ISP's topology [112].

MLU computation. We compute the traffic that flow through each link periodically. To serve a requested piece of content from a PoP s to t, we update the traffic induced along all edges on the path(s) from s to t as determined by the routing protocol using the bytes-downloaded information in the trace. To compute the MLU, we partition simulation time into 5-minute intervals and compute the average utilization of each link in each 5-minute interval. We discard the values of the first day of the trace in order to warm up the caches, as we are interested in steady-state behavior. We then compute our primary metric, which is the 99-percentile MLU, as the 99^{th} percentile of the link utilization over all links and all 5-minute time periods. We use 99-percentile instead of the maximum as the former is good proxy for the latter but with less experimental noise. Finally, for ease of visualization, we scale the 99-percentile MLU values in all graphs so that the maximum 99-percentile MLU across all schemes in each graph is equal to 1. We call this scaled MLU the *normalized MLU*. Note that only the relative ratios of the MLUs for the different schemes matter and scaling up the MLU uniformly across all schemes is equivalent to uniformly scaling down the network resources or uniformly scaling up the traffic in the CDN traces.

Latency cost computation. Our latency cost metric, which models user-perceived latencies, is a sum of the latency on ISP backbone links and the the latency from user to its nearest PoP. Traffic served from origin incurs an additional latency from origin to the exit locations in the network. We assume origin servers to be located close to exit locations so that latency from exit locations to origin servers is a small fraction of the overall end user latency. The latency cost of a link e for a interval of a second when traffic (in bits/sec) on link e is V_e and link utilization is u_e, is calculated as $V_e \times L_e(u_e)$, where L_e is the latency function defined in Section 4.2. The aggregate latency cost of a link is calculated by summing the latency costs for all 1 sec intervals during the experiment (excluding the first day). The user-to-nearest PoP latency cost is calculated by summing the traffic (in bits) requested by a user times the propagation delay to its nearest PoP for all users.

Storage. We assume that storage is provisioned uniformly across PoPs except in Section 4.4.6 where we analyze heterogenous storage distributions. We repeat each simulation with different levels of provisioned storage. Since the appropriate amount of storage depends on the size of the working set of the content being served, we use as a metric of storage the *storage ratio*, or the ratio of total storage at all PoPs in the network to the average storage footprint of all content accessed in a day for the trace. The total storage across all nodes for a storage ratio of 1 is 228 GB, 250 GB, and 895 GB for news, entertainment and downloads respectively.

4.4.2 Schemes Evaluated

Each evaluated scheme has a content placement component and a routing component.

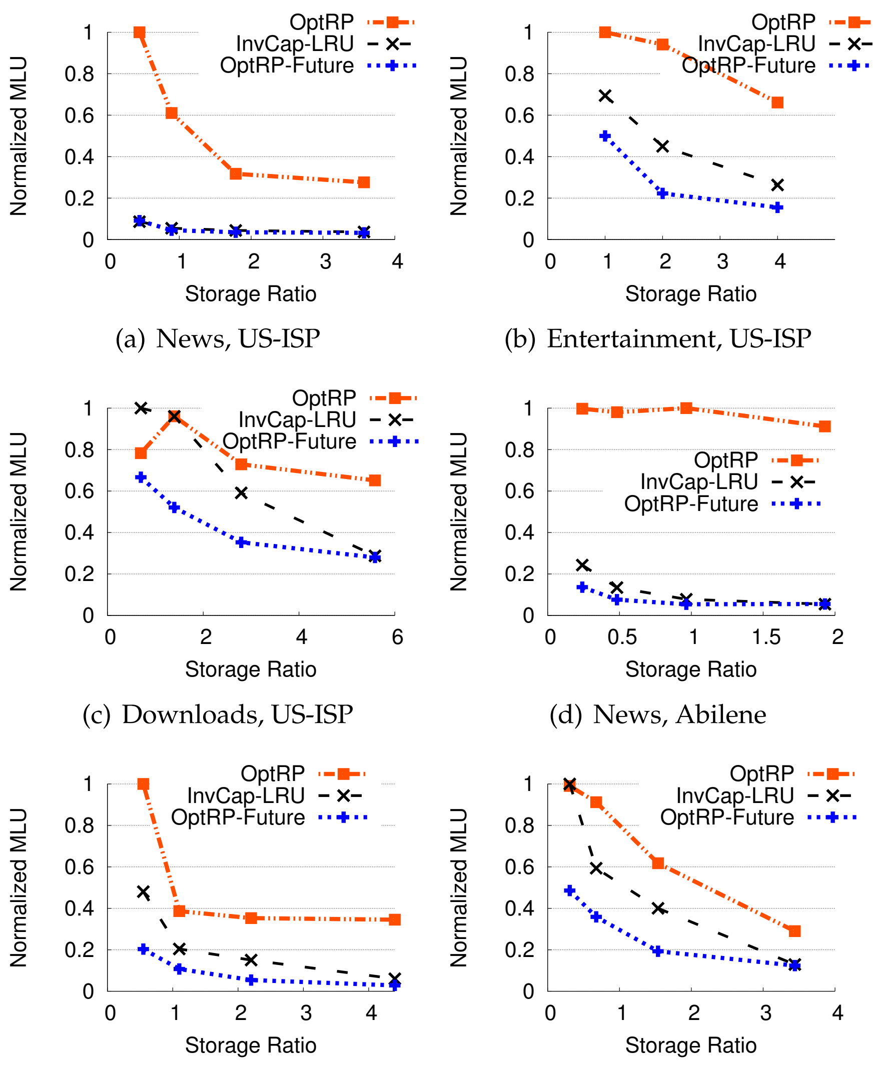
InvCap-LRU uses LRU as the cache replacement strategy and InverseCap (with ECMP) as the routing strategy. InverseCap is a static, shortest-path routing scheme where link weights are set to the inverse of the link capacity. This scheme requires no information of either the content demand or the traffic matrix. If content is available at multiple PoPs, we choose the closest PoP based on hop count distance. We break ties randomly among PoPs with equal hop count distance.

We added a straightforward optimization to LRU where if a user terminates the request before 10% of the video (file) is viewed (downloaded), the content is not cached (and the rest of the file is not fetched); otherwise the entire file is downloaded and cached. This optimization is used since we observe in our traces that a user watching a video very often stops watching it after watching the initial period. A similar phenomenon is observed for large file downloads, but less frequently than video.

OptR-LRU uses an unplanned placement, LRU, but it uses an planned, optimized routing that is updated every three hours. The routing is computed by solving a multi-commodity flow problem identical to the traditional traffic engineering problem [50]. We assume that the NCDN measures the traffic matrix over the preceding three hours and computes routes that optimize the MLU for that matrix. The matrix incorporates the effect of the unplanned placement and the implicit assumption is that the content demand and unplanned placement result in a traffic matrix that does not change dramatically from one monitoring interval to the next—an assumption that also underlies traffic engineering as practiced by ISPs today.

OptRP computes a joint optimization of placement and routing once a day based on the previous day's content matrix using the MIP formulation of Section 4.2.3. **OptRP-Future** has oracular knowledge of the content matrix for the next day and uses it to calculate a joint optimization of placement, redirection and routing. OptRP and OptRP-Future are identical in all respects except that the former uses the content matrix of the past day while the latter has perfect future knowledge. These two schemes help us understand the value of future knowledge. In practice, it may be possible for an NCDN to obtain partial future knowledge placing it somewhere between the two extremes. For instance, an NCDN is likely to be informed beforehand of a major software release the next day (e.g., new version of the Windows) but may not be able to anticipate a viral video that suddenly gets "hot".

To determine the value of optimizing routing alone, we study the **InvCap-OptP-Future** scheme. This is a variant of OptRP-Future where InverseCap routing is used and content placement is optimized, rather than jointly optimizing both. This scheme is computed using the MIP formulation in Section 4.2.3 but with an additional constraint modification that ensures that InvCap routing is implemented.



(a) News, US-ISP (b) Entertainment, US-ISP

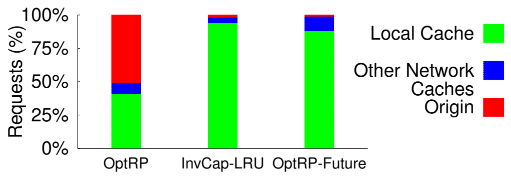
(c) Downloads, US-ISP (d) News, Abilene

(e) Entertainment, Abilene (f) Downloads, Abilene

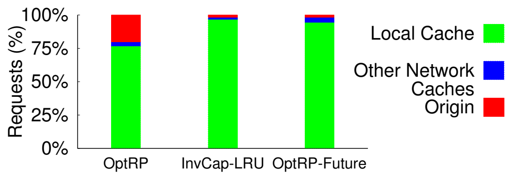
Figure 4.5. Planned OptRP performs much worse than unplanned InvCap-LRU. OptRP-Future performs moderately better than InvCap-LRU primarily at small storage ratios.

We add a suffix **-L** to the names of a scheme if it is optimizing for latency cost instead of MLU, e.g. **OptRP-L**.

For all schemes that generate a new placement each day, we implement the new placement during the low-traffic period from 4 AM to 7 AM EST. This ensures that the traffic generated due to changing the content placement occurs when the links are underutilized. For these schemes, the routing is updated each day at 7 AM EST once the placement update is finished.



(a) News, Abilene



(b) Entertainment, Abilene

Figure 4.6. [Videos, Abilene] OptRP serves 50% and 21% of news and entertainment requests respectively from the origin. InvCap-LRU and OptRP-Future serve at most 2% from the origin.

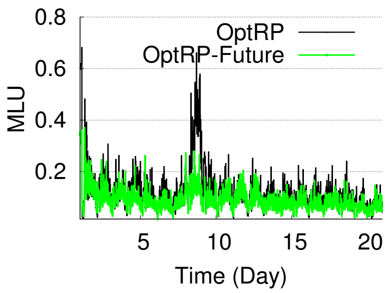


Figure 4.7. [Downloads, US-ISP] OptRP incurs a very high MLU on one "peak load" day.

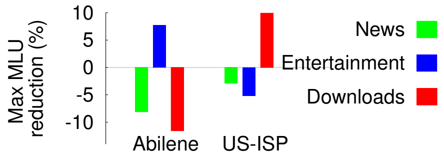
4.4.3 Comparison of network cost

4.4.3.1 Analysis of video & downloads traffic

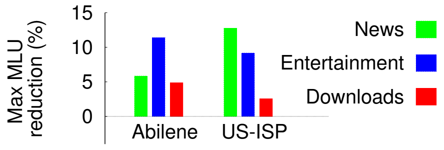
Figure 4.5 shows the results for the news, entertainment and downloads traces on Abilene and US-ISP. Our first observation is that a realistic planned placement and routing scheme, OptRP, performs significantly worse than a completely unplanned scheme, InvCap-LRU. OptRP has 2.2× to 17× higher MLU than InvCap-LRU even at the maximum storage ratio in each graph. OptRP has a high MLU because it optimizes routing and placement based on the previous day's content demand while a significant fraction of requests are for new content not accessed the previous day (see Figure 4.4). Due to new content, the incoming traffic from origin servers is significant, so the utilization of links near the exit nodes connecting to the origin servers is extremely high.

The fraction of requests served from the origin is much higher for OptRP compared to InvCap-LRU and OptRP-Future on the news and the entertainment traces. Figure 4.6 shows that OptRP serves 50% and 21% of requests from the origin for news and entertainment respectively. In comparison, InvCap-LRU and OptRP-Future serve less than 2% of requests from the origin. Therefore, OptRP has a much higher MLU than both InvCap-LRU and OptRP-Future on the two traces.

The downloads trace differs from other traces in that, except for one day, the traffic is quite predictable based on the previous day's history. This is reflected



(a) MLU reduction with OptR-LRU compared to InvCap-LRU



(b) MLU reduction with OptRP-Future compared to InvCap-OptP-Future

Figure 4.8. [All traces] Optimizing routing yields little improvement to MLU of either InvCap-LRU or InvCap-OptP-Future

in the performance of OptRP that performs nearly the same as OptRP-Future on all days except the ninth day of the trace (see Figure 4.7). The surge in MLU for OptRP on the ninth day is because nearly 20% of requests on this day is for new content consisting of highly popular software update releases (see Figure 4.4). The surge in MLU on this one day is mainly responsible for the poor performance of OptRP on the downloads trace.

Next, we observe that InvCap-LRU does underperform compared to OptRP-Future that has knowledge of future content demand. However, InvCap-LRU improves with respect to OptRP-Future as the storage ratio increases. The maximum difference between the two schemes is for the experiment with entertainment trace on US-ISP topology. In this case, at a storage ratio of 1, InvCap-LRU has twice the MLU of the OptRP-Future scheme; the difference reduces to 1.6$\times$ at a storage ratio of 4. This shows that when storage is scarce, planned placement with future knowledge can significantly help by using knowledge of the global demand to maximize the utility of the storage. However, if storage is plentiful, the relative advantage of OptRP-Future is smaller. An important implication of our results is that an NCDN should attempt to do planned placement only if the future demand can be accurately known or estimated. Otherwise, a simpler unplanned scheme such as LRU suffices.

How are the above conclusions impacted if InvCap-LRU were to optimize routing or OptRP-Future were to use InverseCap routing? To answer this question, we analyze the maximum reduction in MLU by using OptR-LRU over InvCap-LRU across all storage ratios in Figure 4.8. We similarly compare OptRP-Future and InvCap-OptP-Future. We find that OptR-LRU improves the MLU over InvCap-LRU by at most 10% across all traces suggesting that optimizing routing is of little value for an unplanned placement scheme. OptRP-Future reduces the network cost by at most 13% compared to InvCap-OptP-Future. As we consider OptRP-Future to be the "ideal" scheme with full future knowledge, these results show that the best MLU can be achieved by optimizing content placement alone; optimizing routing adds little additional value.

Somewhat counterintuitively, the MLU sometimes increases with a higher storage ratio for the OptRP scheme. There are three reasons that explain this. First, the optimization formulation optimizes for the content matrix assuming that the demand is uniformly spread across the entire day, however the requests may actually arrive in a bursty manner. So it may be sub-optimal compared to a scheme that is explicitly optimized for a known sequence of requests. Second, the optimization formulation optimizes the MLU for the "smoothed" matrix, but the set of objects placed by the optimal strategy with more storage may not necessarily be a superset of the objects placed by the strategy with lesser storage at any given PoP. Third, and most importantly, the actual content matrix for the next day may differ significantly from that of the previous day. All of these reasons make the so-called "optimal" OptRP strategy suboptimal and in combination are responsible for the nonmonotonicity observed in the experiments.

4.4.3.2 Content chunking

Content chunking is widely used today to improve content delivery and common protocols such as HTTP [105] and Apple HLS [18] support content chunking. This experiment analyzes the effect of content chunking on our findings. In these experiments, we split videos into chunks of 5 minute duration. The size of a video chunk depends on the video bitrate. For the downloads trace, we split content into chunks of size 50 MB.

Our results show that although chunking improves performance of both InvCap-LRU and OptRP-Future, it significantly improves the performance of InvCap-LRU relative to OptRP-Future (see Figure 4.9). Due to chunking, the maximum difference between the MLU of InvCap-LRU and OptRP-Future reduces from 2.5× to 1.4×. At the maximum storage ratio, InvCap-LRU is at most 18% worse compared to OptRP-Future. Our experiments on other traces and topologies (omitted for brevity) show that InvCap-LRU has at most 4% higher network cost than OptRP-Future at the maximum storage ratio. An exception is the news trace, where chunking makes a small difference as more than 95% content is of duration less than our chunk size. Hence, chunking strengthens our conclusion that InvCap-LRU achieves close to the best possible network cost for an NCDN. Even with chunking, OptRP has up to 7× higher MLU compared to InvCap-LRU (not shown in Figure 4.9). This is because chunking does not help OptRP's primary problem of not being able to adapt effectively to new content, so it continues to incur a high cost.

4.4.3.3 Alternative planned schemes

The experiments so far suggest that a planned scheme that engineers placement and routing once a day based on the previous day's demand performs poorly compared to an unplanned scheme, InvCap-LRU. Therefore, in this section, we evaluate the performance of two alternative planned schemes.

First, we evaluate a hybrid placement scheme, which splits the storage at each node into two parts - one for a planned placement based on the previous day's content demand (80% of storage) and the other for placing the content in a unplanned LRU manner (20% of storage). This hybrid strategy is similar to that used in [19]. We find that InvCap-LRU performs either as well or better than the hybrid scheme. We also experimented with assigning a greater fraction of storage to

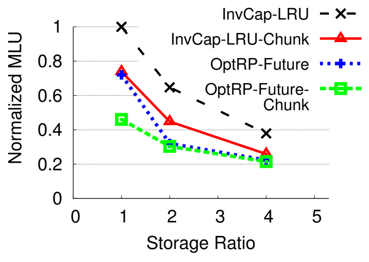


Figure 4.9. [Entertainment, US-ISP] Content chunking helps bridge the gap between InvCap-LRU and OptRP-Future.

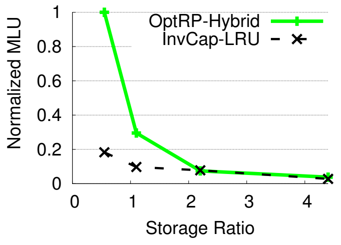


Figure 4.10. [Entertainment, Abilene] Hybrid placement schemes perform at best as well as InvCap-LRU.

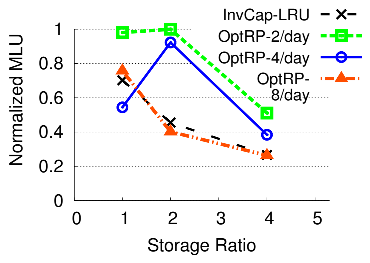


Figure 4.11. [Entertainment, US-ISP] OptRP does not outperform InvCap-LRU despite engineering 8 times a day.

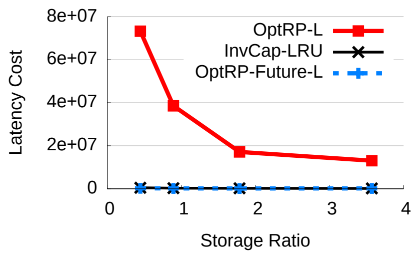
unplanned placement (omitted for brevity), but the above conclusions remain unchanged in those experiments. Of course, a carefully designed hybrid scheme by definition should perform at least as well as the unplanned and planned schemes, both of which are extreme cases of a hybrid strategy. However, we were unable to design simple hybrid strategies that consistently outperformed fully unplanned placement and routing.

Next, we analyze the performance of planned schemes that engineer placement and routing multiple times each day at equal intervals - twice/day, 4 times/day, and 8 times/day. In all cases, we engineer using the content demand in the past 24 hours. As Figure 4.11 shows, OptRP needs to engineer 8 times/day to match the performance of the InvCap-LRU scheme. In all other cases, InvCap-LRU performs better. In fact, the experiment shown here represents the best case for OptRP. Typically, OptRP performs worse even when engineering is done 8 times/day, e.g., on the news trace, we find OptRP incurs up to 4.5$\times$ higher MLU compared to InvCap-LRU even on engineering 8 times/day.

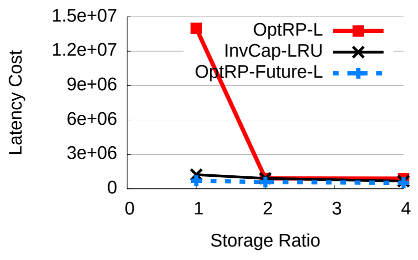
Executing a planned placement requires considerable effort—measuring content matrix, solving a computationally intensive optimization, and moving content to new locations. Further, a planned placement needs to be executed 8 times a day (or possibly more) even to match the cost achieved by an unplanned strategy. Our position is that NCDNs are better served by opting for a much simpler unplanned strategy and provisioning more storage, in which case, an unplanned strategy already obtains a network cost close to the best a planned strategy can possibly achieve.

4.4.4 Comparison of latency cost

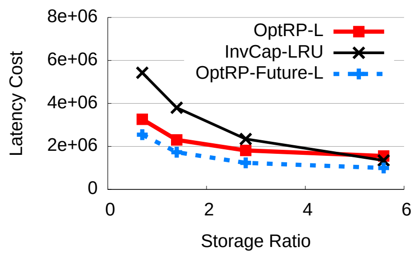
Latency cost metric models user-perceived latency in the network. In our evaluation, we compare InvCap-LRU scheme, which is a completely unplanned scheme, against OptRP-L and OptRP-Future-L that optimize latency cost based on previous day's content matrix and based on next day's content matrix respectively.



(a) News, US-ISP



(b) Entertainment, US-ISP



(c) Downloads, US-ISP

Figure 4.12. A realistic planned scheme, OptRP-L causes excessively high latency costs in some cases. InvCap-LRU achieves latency costs close to ideal planned scheme, OptRP-Future-L, at higher storage ratios.

We experiment with ISP topologies in which links are scaled down uniformly. We needed to scale down the links as our traces did not generate enough traffic to fill even 5% of the capacity of the links during the experiment; ISP networks are unlikely to operate at such small link utilizations. The network topology is scaled such that the 99-percentile MLU for results is 75% link utilization for the InvCap-LRU scheme. This ensures that network has sufficient capacity to support content demand at all storage ratios and network links are not heavily under-utilized.

We present the results of our comparison on the US-ISP topology in Figure 4.12. Experiments on the Abilene topology show qualitatively similar conclusions (graph omitted for brevity). We find that on the news and entertainment traces, OptRP-L scheme results in an order of magnitude higher latency costs. OptRP-L scheme is similar to OptRP scheme except it optimizes latency instead of network cost. Like the OptRP scheme, OptRP-L is unable to predict the popularity of new content resulting in high volume of traffic from origin servers and high link utilization values. OptRP-L either exceeds link capacities or operates close to link capacity for some links which results in very high latencies.

The latency cost of InvCap-LRU relative to OptRP-Future-L improves with an increase in storage ratio. At the smallest storage ratio, InvCap-LRU has 70-110% higher latency cost than OptRP-Future-L. The difference reduces to 14-34% at the maximum storage ratio. Higher storage ratio translate to higher cache hit rates, which reduces propagation delay of transfers and lowers link utilizations. Both these factors contribute to a smaller latency cost for InvCap-LRU. This finding shows that NCDNs can achieve close to best latency costs with an unplanned scheme InvCap-LRU and provisioning moderate amounts of storage.

The performance of OptRP-L on the downloads trace is much closer to OptRP-Future-L than on the other two traces. Unlike other traces, content popularity is highly predictable on the downloads trace based on yesterday's demand, except for a day on which multiple new software releases were done. On all days except one, OptRP-L has nearly optimal latency cost and it incurs a higher latency cost on one day of the trace. As a result, OptRP-L's aggregate latency cost summed over all days is only moderately higher than that of OptRP-Future-L.

4.4.5 Effect of NCDN traffic on network cost

This experiment, unlike previous experiments, considers a network consisting of both ISP and NCDN traffic. Our goal is to evaluate how network costs change as the fraction of NCDN traffic increases in the network. Second, we seek to examine the benefit of optimizing routing over an unplanned routing scheme, InverseCap. To this end, we compare the performance of InvCap-LRU and OptR-LRU schemes. The latter scheme optimizes routing for the combined traffic matrix due to NCDN traffic and ISP transit traffic. In order to estimate the best gains achievable with an optimized routing, we provide to the OptR-LRU scheme knowledge of future ISP traffic matrices. OptR-LRU cannot be provided the knowledge of future NCDN traffic matrices because NCDN traffic matrices can only be measured from experiment itself and we do not know them beforehand. We optimize routing once a day in this experiment. Varying the frequency of routing update did not improve OptR-LRU's performance.

We experiment with hourly transit traffic matrices spanning 7 days from the same Tier-1 ISP — US-ISP. These matrices were collected in February, 2005. Since ISP traffic volumes are much higher than NCDN traffic volumes, at first, we performed this experiment by scaling down the ISP traffic matrices, so that ISP and NCDN traffic have comparable volumes. Of the total NCDN traffic, less than 10% reaches the backbone links, rest is served locally by PoPs. For equal volumes of NCDN and ISP traffic we expected the MLU of a network with ISP traffic only to be much higher than MLU for the network with only NCDN traffic. Our experiment showed that MLU for ISP traffic and NCDN traffic are nearly the same.

We found that this was because the NCDN traffic showed highly variable link utilization even over the course of a few minutes: the maximum link utilization differed by up to 3$\times$ in the course of 15 minutes. The hourly ISP traffic matrix that we experimented with retained the same, smoothed utilization level for an hour. As a result, 99-percentile MLU's for NCDN traffic are the same as that for ISP even though its aggregate backbone traffic was much lesser.

To make the variability of NCDN traffic comparable to ISP traffic, we scaled up the volume of NCDN traffic. The scaling is done by introducing new content similar to a randomly chosen content in the original trace. Each new content is of the same size, and same video bit rate as the original content. All requests for the new content are made from the same locations, at approximately the same times

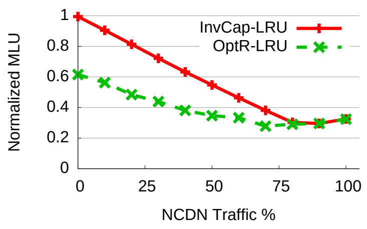


Figure 4.13. [News, US-ISP]Network costs at varying fractions of NCDN traffic in an ISP network.

(within an 1-hour window of the request of the original content), and are of the same durations as the requests for the original content. Our scaling preserves the popularity distribution of objects and the geographic and temporal distribution of requests. We scaled our trace to the maximum level so as to not exceed the memory available (8 GB) in our machine.

We present the results of our experiments on the news trace in Figure 4.13. We vary the fraction of NCDN to ISP traffic, and report MLUs normalized by the total volume of ISP and NCDN traffic. Our results are not independent of the scale of simulations: a larger or a smaller scaling of CDN trace may give quantitatively different conclusions. Hence, we only make qualitative conclusions from this experiment. First, we find that as the fraction of NCDN traffic increases, MLU decreases for both schemes. This is intuitive since a large fraction of NCDN traffic is served from caches located at PoPs. Second, as NCDN traffic increases optimizing routing (OptR-LRU) gives lesser benefits compared to InverseCap routing. In a network dominated by NCDN traffic, optimizing routing gives almost no benefits over InvCap-LRU. We find these results to be consistent with our earlier experiments with NCDN traffic only.

4.4.6 Other Results and Implications

We summarize our findings from other experiments with NCDN traffic here due to space constraints.

Link-utilization aware redirection: We evaluate a request redirection strategy for InvCap-LRU that periodically measures link utilizations in the network and prefers less loaded paths while redirecting requests. Our evaluation shows that such a redirection gives small benefits in terms of network cost ($7\% - 13\%$) and gives almost no benefits on latency costs. This implies that sophisticated network-aware redirection strategies may be of little value for an NCDN.

Request redirection to neighbors: If each PoP redirects requests only to its one-hop neighbor PoPs before redirecting to the origin, InvCap-LRU incurs only a moderate (6%-27%) increase in the MLU. However, if a PoP redirects to no other PoPs but redirects only to the origin, the MLU for InvCap-LRU increases significantly (25%-100%). Thus, request redirection to other PoPs helps reduce network cost, but most of this reduction can be had by redirecting only to neighboring PoPs.

Heterogenous storage: Heterogenous storage at PoPs (storage proportional to the number of requests at a PoP in a trace, and other simple heuristics) increases the MLU compared to homogenous storage for both InvCap-LRU and OptRP-Future, and makes InvCap-LRU more sub-optimal compared to OptRP-Future. This leads us to conclude that our results above with homogeneous storage are more relevant to practical settings.

OptR-LRU parameters: Whether OptR-LRU updates routing every 3 (default), 6, or 24 hours, makes little difference to its performance. Further, whether OptR-LRU optimizes routing using traffic matrix measured over the immediately preceding three hours (default) or using traffic matrices measured the previous day, its network cost remains nearly unchanged. These experiments reinforce our finding that optimizing routing gives minimal improvement over InvCap-LRU.

Number of exit nodes: When the number of network exit nodes is increased to five or decreased to one, our findings in Section 4.4.3.1 remain qualitatively unchanged.

Link failures: The worst-case network cost across all single link failures for InvCap-LRU as well as OptRP-Future is approximately twice compared to their network costs during a failure-free scenario. Comparing the failure-free scenario and link failure scenarios, the relative sub-optimality of InvCap-LRU with respect to OptRP-Future remains the same at small storage ratios but reduces at higher ratios.

4.4.7 Limitations

Although we have evaluated several NCDN management strategies, our experimental methodology suffers from some shortcomings. First, we assume that servers deployed at each PoP have enough resources to serve users requests for locally cached content. In cases when server resources are inadequate, e.g., due to flash crowds, a simple redirection strategy, e.g., redirection to the closest hop-count server used by the InvCap-LRU scheme, may result in poor user-perceived performance. In practice, NCDNs should adopt a redirection strategy that takes server load into account to handle variability of user demands. Second, we model user-perceived latency using a latency cost function that considers propagation delays and link utilization levels. Our latency cost function is a crude approximation of user-perceived latency. A better metric would be based on the TCP throughput experienced by a user. However, an accurate estimation of TCP throughputs of users at the scale of an ISP network with dynamic workloads is extremely challenging. We defer addressing these concerns to future work.

4.5 Conclusions

We posed and studied the NCDN-mangament problem where content distribution and traffic engineering decisions can be optimized jointly by a single entity. Our trace-driven experiments using extensive access logs from the world's largest CDN and real ISP topologies resulted in the following key conclusions. First, simple unplanned schemes for routing and placement of NCDN content, such as InverseCap and LRU, outperform sophisticated, joint-optimal placement and routing schemes based on recent historic demand. Second, NCDN traffic demand can be "shaped" by effective content placement to the extent that the value of engineering routes for NCDN traffic is small. Third, we studied the value of the future knowledge of demand for placement and routing decisions. While future knowledge helps, what is perhaps surprising is that a small amount of additional storage allows simple, unplanned schemes to perform as well as planned ones with future knowledge. Finally, with a mix of NCDN and transit traffic, the benefit of traditional traffic engineering is commensurate to the fraction of traffic that is transit traffic, i.e., ISPs dominated by NCDN traffic can simply make do with static routing schemes. Overall, our findings suggest that content placement is a powerful degree of freedom that NCDNs can leverage to simplify and enhance traditional traffic engineering.

CHAPTER 5

SHRINK: GREENING CDN DATACENTERS

Our growing consumption of digital content is increasing the energy use of datacenters used for content delivery, and datacenters today consume more than 1.3% of world's electricity consumption [102]. This growth is a result of increased availability of several forms of content such as video [33, 85], and social media content, as well new platforms for content consumption, such as smartphones and tablets [33]. Due to our growing dependence on digital content, mechanisms for reducing energy use of content delivery are an important societal need. Further, such mechanisms have potential to be an important cost-cutting tool for content delivery networks (CDNs). Energy is a major factor of operational costs [24] of a datacenter; reducing energy cost of their datacenters would help CDNs stay competitive in an commoditized marketplace.

Recent research has shown that techniques that consolidate demand in a datacenter on a subset of servers can enable turning off up to 50% servers and bring significant energy savings [82, 60, 76, 80]. However, the impact of this energy minimization on user-perceived performance in a CDN datacenter has not been evaluated before. These studies do not evaluate how cache hit rates in a CDN datacenter would be impacted due to server shutdown policies. Note that reduced hit rates adversely affect user-perceived performance. Further, their analyses are based only in terms of system-level metrics, e.g., aggregate load at a datacenter, and not based on user-perceived metrics such as file download times.

The network of a datacenter, which consumes about 10-20% of datacenter energy, can be made energy efficient by traffic engineering techniques as shown by prior work [113, 63, 123, 32, 16]. The energy savings that a scheme achieves depends on traffic patterns in the datacenter, traffic patterns that traffic engineering schemes assume to be fixed. This assumption isn't true for content delivery data centers, where traffic patterns could be influenced by load balancing decisions and server shutdown policies. We hypothesize that there exists potential for more network energy savings in datacenters than shown previously, provided techniques for saving server energy work in coordination with those for saving network energy.

Our goal is to quantify how much energy savings are achievable in a CDN datacenter with minimal or no impact on user-perceived performance. To this end, we seek to design server and network shutdown policies that coordinate with each other and increase datacenter network energy savings, and design server shutdown policies and load balancing algorithms that minimize the reduction in cache hit rates. To conduct a realistic evaluation of proposed techniques, we plan to collect datasets of content access traces of various types from a CDN, and evaluate proposed strategies in terms of user-perceived metrics using a combination of trace-driven experiments and testbed experiments with a prototype.

5.1 Related work

To our knowledge, this would be first effort to evaluate energy minimizing strategies in a CDN datacenter in terms of user-perceived performance, and to explore coordinated server and network shutdown policies. Prior work has explored energy-minimizing strategies for datacenter and ISP networks, and has evaluated potential energy savings of datacenters, without evaluating user-perceived performance metrics.

Energy-minimizing routing: Several papers [113, 63, 123, 32, 16] have proposed network energy minimizing routing algorithms for ISP networks and data center networks. These approaches save energy by concentrating the traffic, which is input in the form of a traffic matrix, on a subset of links and switches, and shutting off remaining switches and links. In [113] and [63], authors demonstrate using Click and OpenFlow based prototypes respectively, the feasibility of implementing these routing protocols in today's switches. In comparison to approaches that optimize routing for a given traffic matrix, this work explores how server shutdown policies can coordinate with energy-minimizing routing strategies to increase energy savings in a datacenter.

Energy-proportional datacenters: Analysis of data center traces have shown that servers in datacenters are lightly loaded in many cases. Motivated by this observation, several papers [82, 60, 76, 80], based on trace-driven experiments, have explored how much energy savings can be obtained by using only a fraction of the servers at a given time, and by shutting-off remaining servers or switching them to a low power state. While these studies show that there is significant potential for energy savings, they implicitly assume the total load will be evenly distributed among the active set of servers. However, server shutdown policies could cause additional load imbalance in datacenters, thereby degrading user-perceived performance. Further, in case of a CDN datacenter, they do not evaluate the impact of server shutdown policies on cache hit rates, which are a key determinant of user-perceived performance. In comparison, a key goal of this work is to design load-balancing and server shutdown strategies that ensure cache hit rates are minimally affected, and load imbalance is small enough to not cause a degradation in user-perceived performance.

Other work: There are a number of recent efforts whose goals are complementary to this work. Work on device-level power management for switches [84] and servers [27] could complement our strategies. A CDN could use geographic load balancing to globally optimize energy use across datacenters while using our strategies in a single datacenter [96, 78, 58, 100]. Using recently developed techniques for designing highly scalable load-balancers such as ETTM [43] and Ananta

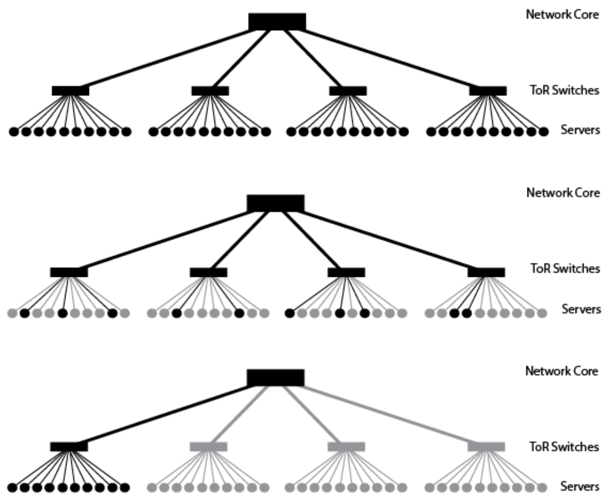


Figure 5.1. A datacenter topology. Black components are turned on and grey components are turned off. (Top) All servers and switches are on as is the current practice. (Middle) Demand is consolidated on randomly selected 10 servers and remaining servers are shutoff. All ToR switches must be kept on to provide connectivity to servers. (Bottom) Demand is consolidated on servers in one rack, which allows servers as well as ToR switches in other racks to be turned off.

[92], a CDN can extend the strategies we propose in this work to develop a scalable solution for use in datacenters with tens of thousands of servers.

5.2 Research outline

Our goal is to design a comprehensive solution for CDN datacenters which makes load balancing, server shutdown, and traffic engineering decisions to reduce energy use with a minimal impact on user perceived performance.

5.2.1 Algorithm design

There are three design questions which are key to achieving the performance and energy-efficiency goals of a CDN datacenter.

(1) How many servers to keep active? Reducing the number of active serves is necessary for energy-efficiency because servers consume more than 80% of power usage in datacenters [13], and today's servers consume 50-70% energy even in idle state [24]. We will decide the number of active servers to ensure the following:
Availability: The datacenter will have sufficient resources, e.g, compute, network, disk bandwidth, to handle all incoming requests. ensure both high availability and high performance.
Performance: Cache misses at datacenter require objects to be fetched from a remote datacenter and results in a noticeable increase in user-perceived performance. The

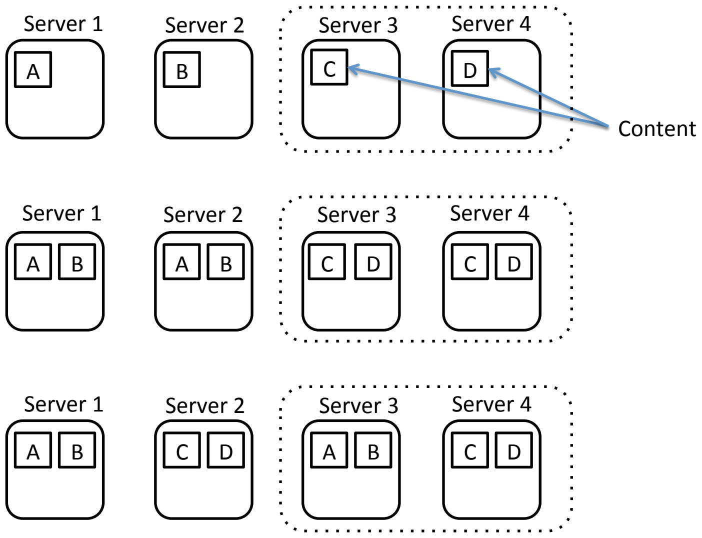


Figure 5.2. Replication strategy impacts content availability after server shutdown. All four servers are active during normal utilization periods, but the two servers on the right are turned off during low utilization periods. Squares with same letters represent replicas of the same content. (Top) One replica of each content is maintained as shown. When servers 3 and 4 are shutdown, two of the four content become unavailable. (Middle) Two replicas of each content are maintained, but still shutting down servers 3 and 4 makes two of the four content unavailable. (Bottom) Two replicas of each content are maintained, but servers 1 and 2 have one copy of all four content. In this case, all content is available despite shutting down servers 3 and 4.

aggregate storage available across servers will be sufficient to enough a cache hit rate that is comparable to the cache hit rates if all servers were active.

(2) Which servers to keep active? The set of active servers determine the potential energy savingsfrom network switches. As shown in Figure 5.2, randomly selecting the set of active servers requires entire datacenter network to be turned off. Therefore, we will select the active servers in a manner that enables more network switches to be turned off, e.g., selecting servers that are within a small sub-tree in a datacenter topology.

(3) Which servers to replicate each content? As Figure 5.2 shows, if insufficient number of replicas are maintained or the set of servers is not chosen carefully, then server shutdown could temporarily make content unavailable in the cluster. Such content unavailability decreases cache hits in datacenter, thereby hurting user perceived performance. To maximize cache hits, we will replicate content across servers so that despite ongoing server shutdown events, one copy of content is kept available in most cases.

5.2.2 System overview

Our algorithms would be implemented as a datacenter manager software, called **Shrink**, running at the front-end load balancer. All servers in the datacenter would be running a caching proxy software such as Squid [40]. All caching proxies would be configured as peers of one another, enabling them to request content from one another. Shrink would make its decisions based on content demand statistics from each server. To this end, Shrink would require support from daemon processes running at each server for reporting statistics. Periodically, Shrink would compute load balancing and traffic engineering decisions, and also decide which set of servers and switches to keep active in the next interval. The computed load balancing decisions would be updated locally. To implement routing decisions, Shrink would require OpenFlow [52] support at switches to communicate the computed routing. To turn servers on/off as necessary, Shrink would again depend on the daemon process running at each server.

5.2.3 Data collection

To evaluate our strategies for a realistic content workload, we have collected content access traces from a datacenter of a leading commercial CDN, Akamai. The traces include all requests received at a datacenter with 18 servers over a week in December 2013. Our anonymized traces traces include several major types of traffic observed in a CDN including video, social media, and other web traffic. Each anonymized log entry includes among other fields, the request timestamp, content URL, size of requested content, actual number of bytes sent, and IP address of the user. Overall, the traces contain more than 2 billion requests generating nearly 200 TB of network traffic.

5.2.4 Experimental evaluation

Our experimental evaluation would seek to find answers to following questions:

- How much energy savings do our strategies achieve compared to the current practice of leaving entire datacenters in “always on” state? How does it compare with respect to a lower bound on energy savings?
- How do the user-perceived performance metrics such as file download times impacted compared to an “always on” strategy? What, if any, is the increase in average and 99-percentile latency? Does a specific subset of content, such as that of a single content provider, see a performance degradation?
- How much additional energy saving does a coordinated server and network shutdown save over uncoordinated approaches for different datacenter network topologies?

A timeline of the proposed research for this project is given in Chapter 7.

CHAPTER 6

DEMAND-AWARE GEO-DISTRIBUTED PLACEMENT FOR LOW LATENCY

Geo-distributed computing clouds such as Amazon EC2, Google offer a scalable, fault tolerant and flexible solution for hosting web services that cater to users world-wide. The easy availability of these platforms provides opportunities to several applications to leverage the performance benefits of geo-distribution.

A challenge for a web service in switching to a geo-distributed cloud is the cost of replication of the "data-tier". Several web services today generate dynamic content such as weather information, stock prices, and status updates posted by users on a social networking website. Data replication is necessary to reduce latency of content accesses but is a costly operation for dynamic content. The reason is that the cost of propagating updates to dynamic content increases linearly with the number of locations. State-of-art replication alternatives either provide poor cost-vs-performance tradeoffs or leave data placement to be done manually by web-services which increases human cost and effort. For example, DHT-based designs make a constant number of replicas but result in high latency of content accesses.

Our key insight is that replication of dynamic data should be done in a demand-aware manner such that a limited number of data replicas placed close to pockets of demand are sufficient to reduce latency of content accesses. A demand-aware placement implicitly assumes geo-locality of workloads, an assumption we believe is justified based on recent studies of workload characteristics. We have developed a heuristic placement strategy that decides number of replicas based on read-to-write ratio of a name and selects replica locations based on geo-distribution of demand to provide low lookup latency, low update cost, and high availability.

Our system, Auspice, is implemented as a geo-distributed key-value store that stores arbitrary JSON objects as records. Like several other key-value stores [8, 47, 5], Auspice exposes a simple GET/PUT interface to clients. Auspice provides flexible consistency semantics for accesses to a single object. However, Auspice is not a general-purpose database and lacks several features that are common in a database, e.g., support for running SELECT queries efficiently, or support for transactions. Auspice is scalable to a large number of locations and data items due to a fully decentralized design both in the data plane and and the control plane that makes data placement decisions.

An application suited for Auspice is a global name service that provide name-to-address mapping for mobile devices. We have evaluated Auspice extensively for an expected workload of such a global name service. Our experiments show that Auspice provides 5.4×-11.2× lower lookup latency than a DHT-based design for DNS. In a comparison to commercial managed DNS providers, we find that Auspice provides a median update latency that is 1.1 sec to 24.7 sec lower than three top-tier providers.

6.1 Auspice design & implementation

6.1.1 Design goals

Our goal is to design a distributed system that meets the following requirements:

(1) Low lookup latency: Replicas of a key should be placed close to end-users accessing it so as to minimize user-perceived response times.

(2) Resource cost: The design must ensure low replication cost. A naive way to minimize lookup latencies is to replicate every record at every possible location, however high mobility means high update rates, so the cost of pushing each update to every replica would be prohibitive. Worse, load hotspots can actually degrade lookup latencies.

(3) High availability: The design must ensure resilience to node failures including outages of entire datacenters; by consequence, it should also prevent crippling load hotspots.

(4) Consistency: The design must provide flexible consistency semantics as desired by an application.

6.1.2 Auspice's geo-distributed design

To address the above goals, the Auspice is designed as a massively geo-distributed key-value store. The geo-distribution is essential to the latency and availability goals while the key-value API is chosen for its popularity among today's web services. Each *record* in Auspice is associated with a globally unique identifier (GUID) that is the record's primary key. A record contains an associative array of key-value pairs, wherein each key K_i is a string and the value V_i may be a string, a primitive type, or recursively a key-value pair, as shown below.

GUID $\mid K_1, V_1 \mid K_2, V_2 \mid \cdots$

At the core of Auspice is a placement engine that achieves the latency, cost, and availability goals by adapting the number and locations of replicas of each record in accordance with (1) the lookup and update request rates for the record, (2) the geo-distribution of requests for the record, and (3) the aggregate request load across all records.

Figure 6.1 illustrates the placement engine. Each record is associated with a fixed number, F, of *replica-controllers* and a variable number of *active replicas* of the corresponding record. The record's replica-controllers are computed using consistent hashing to select F consecutive or otherwise deterministic nodes along the ring *onto* which the hash function maps records and nodes. The replica-controllers

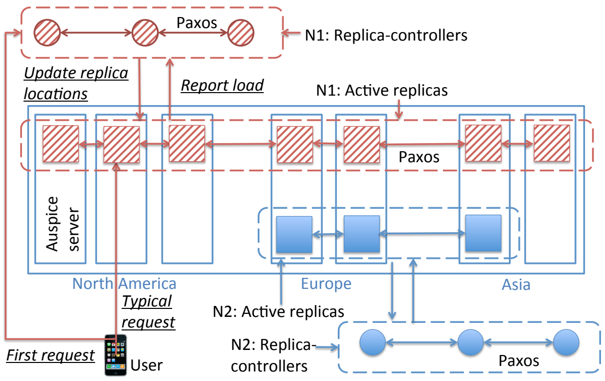


Figure 6.1. Geo-distributed servers in Auspice. Replica-controllers (logically separate from active replicas) decide placement of active replicas and active replicas handle requests from end-users. N1 is a globally popular record and is replicated globally; record N2 is popular in select regions and is replicated in those regions.

are responsible only for determining the number and locations of the active replicas, and the actives replicas are responsible for maintaining the actual record and processing client requests. The replica-controllers implement a replicated state machine using Paxos [72] in order to maintain a consistent view of the current set of active replicas.

A record's replica-controllers compute its active replica locations in a *demand-aware* manner. This computation proceeds in epochs as follows. At creation time, the active replicas are chosen to be physically at the same locations as the corresponding replica-controllers. In each epoch, the replica-controllers obtain from each active replica a summarized load report that contains the request rates for that record from different regions as seen by that replica. Here, *regions* partition users into non-overlapping groups that capture locality, e.g., IP prefixes or a geographic partitioning based on cities; and the *load report* is a spatial vector of request rates as seen by the replica. The replica-controllers aggregate these load reports to obtain a concise spatial distribution of all requests for the record.

6.1.2.1 Demand-aware replica placement

In each epoch, the replica-controllers use a placement algorithm that takes as input the aggregated load reports and capacity constraints at servers to determine the number and locations of active replicas for each record so as to minimize client-perceived latency. We have formalized this *global* optimization problem as a mixed-integer program and shown it to be computationally hard. As our focus is on simple, practical algorithms, we defer the details of the optimization approach [1], using it only as a benchmark in small-scale experiments with Auspice's heuristic algorithm.

Auspice's placement algorithm is a simple heuristic and can be run *locally* by each replica-controller. The placement algorithm computes the number of replicas using the lookup-to-update ratio of a record in order to limit the update cost to within a small factor of the lookup cost. The number of replicas is always kept more than the minimum number needed to meet the availability objective under failures. The location of these replicas are decided to minimize lookup latency by placing a fraction of replicas close to pockets of high demand for that record while placing the rest randomly so as to balance the potentially conflicting goals of reducing latency and balancing load among servers.

Specifically, the placement algorithm computes the number of replicas for a record as $(F + \beta r_i/w_i)$, where r_i and w_i are the lookup and update rates of record i; F is the minimum number of replicas needed to meet the availability goal (§6.1.1); and β is a replication control parameter that is automatically determined by the system so as to trade off latency benefits of replication against update costs given capacity constraints as follows. In each epoch, the replica-controllers recompute β so that the aggregate load in the system corresponds to a preset threshold utilization fraction μ. For simplicity of exposition, suppose read and write operations impose the same load, and the total capacity across all servers (in reads/sec) is C. Then, β is set so that

$$\mu C = \sum_i r_i + \sum_i (F + \beta \frac{r_i}{w_i}) w_i \tag{6.1}$$

where the right hand side represents the total load summed across all records. The first term in the summation above is the total read load and the second is the total write load.

Having computed β as above, replica-controllers compute the locations of active replicas for record i as follows. Out of the $F + \beta r_i/w_i$ total replicas, a fraction ν of replicas are chosen based on locality, i.e., replica-controllers use the spatial vector of load reports to select $\nu(F + \beta r_i/w_i)$ servers that are respectively the closest to the top $\nu(F + \beta r_i/w_i)$ regions sorted by demand for record i. The remaining $(1 - \nu)(F + \beta r_i/w_i)$ are chosen randomly without repetition. The locality-based replicas above are chosen as the *closest* with respect to round-trip latency plus load-induced latency measured locally at each server. An earlier design chose them based on round-trip latency alone, but we found that adding load-induced latencies in this step (in addition to choosing the remaining replicas randomly) ensures better load balance and lowers overall client-perceived latency. Our current prototype and system experiments fix the random perturbation knob ν to 0.5. We have since developed a slightly modified placement scheme that relieves the administrator from setting ν manually, automatically balancing locality-awareness and load to ensure low latencies [1]. Thus, an administrator need only specify F and μ based on fault tolerance and aggressiveness of capacity utilization.

Auspice's replica placement scheme (Eq. 6.1) is designed to use a fraction μ of system-wide resources so as to make at least F and at most M replicas of each record, where M is the total number of server locations. Thus, at light load, Auspice may replicate every record at every location, while under heavy load, it may create exactly F replicas for all but the most popular records.

6.1.2.2 Client request routing

A client request is routed from an end-host to a suitable server as follows. The set of all servers in an Auspice instance is known to each member server and can be obtained from a well-known location. When a client encounters a request for a record for the first time, it uses the known set of all servers and consistent hashing to determine the replica-controllers for that record and sends the request to the closest replica-controller. The replica-controller returns the set of active replicas for the record and the client resends the request to the closest active replica. In practice, we expect replica-controllers to be contacted infrequently as the set of active replicas can be cached and reused until they change in some future epoch.

Network latency as well as server-load-induced latency help determine the closest replica at a client. Each client maintains an estimate of the round-trip latency to all servers using infrequent pings; an (as yet unimplemented) optimization to reduce the overhead of all-to-all pings is to use coordinate embedding, geo-IP, or measurement-driven techniques [81]. To incorporate load-induced latency, the latency estimate to a server is passively measured as a moving average over lookups sent to that server. The client also maintains a timeout value based on the moving average and variance of the estimates. If a lookup request sent to a server times out, the client infers that either the server or network route is congested, and it multiplicatively increases its latency estimate to that server by a fixed factor. Thus, if multiple lookups sent to a server time out, the estimated latency shoots up and the client stops sending requests to that server, which effectively acts as a more agile load-balancing policy in the request routing plane (complementing the replica placement plane above that operates in coarser-grained epochs).

6.1.2.3 Consistency with static replication

As a geo-distributed key-value store, Auspice must at least ensure this eventual consistency property: *all active replicas must eventually return the same value of the record and, in a single-writer scenario, this value must be the last update made by the (only) client*; "eventually" means that there are no updates to a record and no replica failures for sufficiently long. Violating this property means that a client may be indefinitely unable to obtain the the up-to-date value of the record even though the record is no longer being updated.

With a static set of replicas, it is straightforward to support this property. A replica receiving a client update need only record the write in a persistent manner locally, return a commit to the client, and lazily propagate the update to other active replicas for that record. Lazy propagation is sufficient to ensure that all replicas eventually receive every update committed at any replica, and a deterministic reconciliation policy, e.g., as in Dynamo [39], suffices to ensure that concurrent updates are consistently applied across all replicas. Temporary divergence across replicas under failures can be shortened by increasing durability, i.e., by recording the update persistently at more replicas before returning a commit to the client. The additional "single-writer" clause is satisfied simply by incorporating a client-local timestamp in the deterministic reconciliation policy.

Total ordering. To be useful to a broader set of applications that demand more sophisticated query patterns, it may be useful in some scenarios to ensure that update operations (like appending to or deleting from a list) to a record are applied in the same order by all active replicas. Ensuring a total ordering of all updates to a record is a stronger property than eventual consistency, calling for a state-machine approach, which Auspice supports as an option. In fact, Auspice supports an option to perform total ordering of all updates and lookups as well, which is a even stronger property than total write ordering alone.

To this end, active replicas for a record participate in a Paxos instance maintained separately for each record (distinct from Paxos used by replica-controllers to compute active replicas for that record). Each update is forwarded to the active replica that is elected as the Paxos coordinator that, under graceful execution, first gets a majority of replicas to accept the update number and then broadcasts a commit. Total write ordering of course implies that updates can make progress only when a majority of active replicas can communicate with each other while maintaining safety (consistent with the so-called CAP dilemma).

6.1.2.4 Consistency with replica reconfiguration

With a dynamic set of replicas as in Auspice, achieving eventual consistency is straightforward, as it suffices if a replica recovering from a crash lazily propagates all pending writes to a record to its *current* set of active replicas as obtained from any of the consistently-hashed replica-controllers for the record. However, satisfying the (optional) total write order property above is nontrivial.

To this end, we have designed a two-tier reconfigurable Paxos system that involves explicit coordination between the consensus engines of the replica-controllers and active replicas. Reconfiguration is accomplished by a replica-controller issuing and committing a stop request that gets committed as the last update of the current active replica group. The replica-controller subsequently initiates the next group of active replicas that can obtain the current record value from any member of the previous group. This design shares similarities with Vertical Paxos [73], however we were unable to find existing implementations or even reference systems using similar schemes, so we had to develop it from scratch. The details of the reconfiguration protocol are here [1].

6.1.2.5 Replica reconfiguration policies

The policy decision of when to reconfigure the replica group for a specific record is orthogonal to the consistency mechanisms above. The frequency of reconfiguration presents a tradeoff between agility to demand and the reconfiguration messaging overhead. This overhead includes (1) load reports per epoch from active replicas to replica-controllers; (2) replica-controllers computing the new set of active replicas by consensus; (3) replica-controllers notifying the old active replicas to stop and the new ones to start; and (4) newly added replicas obtaining the record state from any old replica, and (5) replica-controllers agreeing that the group change is complete. Each of these steps entails one or two small messages per record at each active replica or replica controller, except for step 4 where the overhead depends on the size of the record. The first overhead is incurred per epoch and can be further reduced by having an active replica issue a load report for a record only if the geo-locality of demand for the record has changed beyond a threshold at that replica. The subsequent steps are needed only if the policy deems reconfiguration as warranted based on the record's aggregate load report (noting that only a subset of records may need reconfiguration in an epoch).

Thus, any reconfiguration policy that (1) limits load reports at an active replica to at most once per m client requests for the record at that replica, and (2) limits reconfiguration at replica-controllers to at most once per km total client requests across the k currently active replicas, suffices to ensure that the reconfiguration overhead is at most a fraction $\approx 1/m$ of the incoming client request load. As a consequence, it may take at least m requests for a record at a new location before an active replica for it is created there.

6.1.2.6 Implementation status

We have implemented Auspice as described in Java with 28K lines of code. We have been maintaining an alpha deployment for research use for many months across eight EC2 regions. We have implemented support for two pluggable NoSQL data stores, MongoDB (default) and Cassandra, as persistent local stores at servers. We do not rely on any distributed deployment features therein as the coordination middleware is what Auspice provides.

6.2 Evaluation

Our evaluation is centered around an representative application: a global name service (GNS) to provide name-to-address mapping for mobile devices. We expect a GNS to receive requests from clients spread in a large geographic area and the records in GNS to show a non-trivial update rate due to end-host mobility. These traits make a GNS a representative application for Auspice. In the following discussion, we refer an Auspice server as a *name server*, an Auspice client as a *local name server*, and a record as a *name record* or a *name*.

Our evaluation seeks to answer the following questions: (1) How well does Auspice's design meet its performance, cost, and availability goals compared to state-of-the-art alternatives under high mobility? (2) How does Auspice's cost-performance tradeoff compare to best-of-breed managed DNS services for today's (hardly mobile) domain name workloads?

6.2.1 Experimental setup

Testbeds: We use geo-distributed testbeds (Amazon EC2 or Planetlab) or local emulation clusters (EC2 or a departmental cluster) depending upon the experiment's goals.

Workload: There is no real workload today of clients querying a name service in order to communicate with mobile devices frequently moving across different network addresses, both because such a name service does not exist and mobile

Workload parameter	**Value**
Fraction of (highly mobile) device names	90%
Fraction of (mostly static) service names	10%
% of device name lookups	33.33%
% of device name updates	33.33%
% of service name lookups	33.33%
% of service name updates	0.01%
Geo-locality: [devices, services]	[0.75, 0.8]

Table 6.1. Default workload parameters.

devices do not have publicly visible IP addresses. So we conduct an evaluation using synthetic workloads for device names (§6.2.2), but to avoid second-guessing future workload patterns, we conduct a comprehensive sensitivity analysis against all of the relevant parameters such as the read rate, write rate, popularity, and geo-locality of demand [1].

The following are default experimental parameters for *device names*. The ratio of the total number of lookups across all devices to the total number of updates is 1:1, i.e., devices are queried for on average as often as they change addresses. The lookup rate of any single device name is uniformly distributed between 0.5–1.5× the average lookup rate; the update rate is similarly distributed and drawn independently.

How requests are geographically distributed is clearly important for evaluating a replica placement scheme. We define the *geo-locality* of a name as the fraction of requests from the top-10% of regions where the name is most popular. This parameter ranges from 0.1 (least locality) to 1 (high locality). For a device name with geo-locality of g, a fraction g of the requests are assumed to originate from 10% of the local name servers, the first of which is picked randomly and the rest are the ones geographically closest to it. We pick the geo-locality $g = 0.75$ for device names, i.e., the top 10% of regions in the world will account for 75% of requests, an assumption that is consistent with the finding that communication and content access exhibits a high country-level locality [71], and is consistent with the measured geo-locality (below) of service names today.

In addition to device names, *service names* constitute a small fraction (10%) of names and are intended to capture domain names like today with low mobility. Their lookup rate (or popularity) distribution and geo-distribution are used directly from the Alexa dataset [3]. Using this dataset, we calculated the geo-locality exhibited by the top 100K websites to be 0.8. Updates for service names are a tiny fraction (0.01%) of lookups as web services can be expected to be queried much more often than they are moved around. The lookup rate of service names is a third of the total number of requests (same as the lookup or update rates of devices).

Table 6.1 summarizes the default workload parameters.

Replication schemes compared: Auspice uses the replica placement strategy as described in §6.1 with the default parameter values $F = 3, \mu = 0.7, \nu = 0.5$.

We compare Auspice against the following: (1) **Random-M** replicates each name at three random locations; (2) **Replicate-All** replicates all names at all locations; (3) **DHT+Popularity** replicates names using consistent hashing with replication similar to Codons[98]. The number of replicas is chosen based on the popularity ranking of a name and the location of replicas is decided by consistent hashing. The average hop count in Codons's underlying Beehive algorithm is set so that it creates the same average number of replicas as Auspice for a fair comparison. All schemes direct a lookup to the closest available replica after the first request.

6.2.2 Evaluating Auspice's replica placement

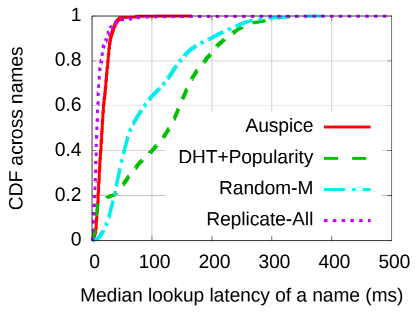
We conduct experiments in this subsection on a 16-node (each with Xeon 5140, 4-cores, 8 GB RAM) departmental cluster, wherein each machine hosts 10 instances of either name servers or local name servers so as to emulate an 80-name server Auspice deployment. We instrument the instances so as to emulate wide-area latencies between any two instances that correspond to 160 randomly chosen Planetlab nodes. We choose emulation instead of a geo-distributed testbed in this experiment in order to obtain reproducible results while stress-testing the load-vs.-response time scaling behavior of various schemes given identical resources.

6.2.2.1 Lookup latency and update cost

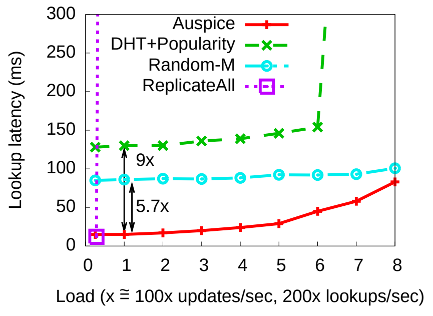
How well does Auspice use available resources for replicating name records? To evaluate this, we compare the lookup latency of schemes across varying load levels. A machine running 10 name servers receives on average 2000 lookups/sec and 1000 updates/sec at a load = 1. For each scheme, load is increased until 2% of requests fail, where a failed request means no response is received within 10 sec. The experiment runs for 10 mins for each scheme and load level. To measure steady-state behavior, both Auspice and DHT+Popularity pre-compute the placement at the start of the experiment based on prior knowledge of the workload.

Figure 6.2(a) shows the distribution of median lookup latency across names at the smallest load level (load = 0.3). Figure 6.2(b) shows load-vs-lookup latency curve for schemes, where "lookup latency" refers to the mean of the median lookup latencies of names. Figure 6.2(c) shows the corresponding mean of the distribution of update cost across names at varying loads; the update cost for a name is the number of replicas times the update rate of that name.

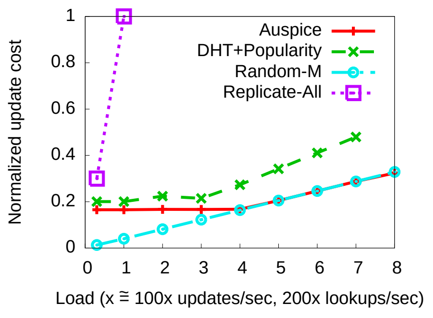
Replicate-All gives low lookup latencies at the smallest load level, but generates a very high update cost and can sustain a request load of at most 0.3. This is further supported by Figure 6.2(c) that shows that the update cost for Replicate-All at load = 0.4 is more than the update cost of Auspice at load = 8. In theory, Auspice can have a capacity advantage of up to N/M over Replicate-All, where N is the total number of name servers and M is the minimum of replicas Auspice must make for ensuring fault tolerance (resp. 80 and 3 here). *Random-M* can sustain a high request load (Fig. 6.2(b)) due to its low update costs, but its lookup latencies are higher as it only creates 3 replicas randomly.



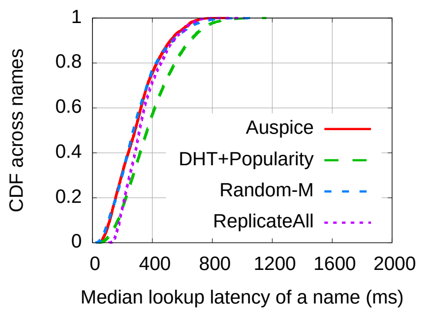
(a) Lookup latencies (load = 0.3)



(b) Lookup latency vs. load



(c) Update cost vs. load



(d) Update latency

Figure 6.2. Auspice has up to 5.7× to 9× lower latencies than Random-M and DHT+Popularity reps. (6.2(b)). A load of 1 means 200 lookups/sec and 100 updates/sec per name server. Replicate-All peaks out at a load of 0.3 while Auspice can sustain a request load of up to 8 as it carefully chooses between 3 and 80 replicas per name. In Figure 6.2(d), median update latency of Auspice with total write ordering per name is 284ms and is comparable to other schemes.

Auspice has $5.7\times-9\times$ lower latencies over Random-M and DHT+Popularity respectively (Figure 6.2(b), load=1). This is because it places a fraction of the replicas close to pockets of high demand unlike the other two. At low to moderate loads, servers have excess capacity than the minimum needed for fault tolerance, so Auspice creates as many replicas as it can without exceeding the threshold utilization level (Eq. 6.1), thereby achieving low latencies for loads$\leq$4. At loads $\geq$ 4, servers exceed the threshold utilization level even if Auspice creates the minimum number of replicas needed for fault tolerance. This explains why Auspice and Random-M have equal update costs for loads $\geq$ 4 (Figure 6.2(c)). Reducing the number of replicas at higher loads allows Auspice to limit the update cost and sustain a maximum request load that is equal to Random-M.

DHT+Popularity has higher lookup latencies as it replicates based on lookup popularity alone and places replicas using consistent hashing without considering the geo-distribution of demand. Further, it answers lookups from a replica selected enroute the DHT route. Typically, the latency to the selected replica is higher than the latency to the closest replica for a name, which results in high latencies.

DHT+Popularity replicates 22.3% most popular names at all locations. Lookups for these names go to the closest replica and achieve low latencies; lookups for remaining 77.7% of names incur high latencies.

DHT+Popularity incurs higher update costs than Auspice even though both schemes create nearly equal numbers of replicas at every load level. This is because DHT+Popularity decides the number of replicas of a name only based on its popularity, i.e., lookup rates, while Auspice decides the number of replicas based on lookup-to-update ratio of names. Due to its higher update costs, DHT+Popularity can not sustain as high a request load as Auspice.

6.2.2.2 Update latency, update propagation delay

The *client-perceived update latency*, i.e., the time from when when a client sends an update to when it receives a confirmation. These numbers are measured from the experiment in §6.2.2.1 for load=0.3. The median and 90th percentile update latency for Auspice with total write ordering is 284ms and is comparable to other schemes. A request, after arriving an active replica, takes four one-way network delays (two rounds) to be committed by Paxos. The median update latency is a few hundred milliseconds for all schemes as it is dominated by update propagation delays.

The *update propagation delay*, i.e., the time from when a client issued a write till the last replica executes the write, is a key determiner of the time-to-connect. With eventual consistency, update propagation takes one round, while with total write ordering, update propagation takes two rounds and 50% more messages.

The measured update propagation delay is consistent with expectations. With eventual consistency, this delay is 154 ms, while with total write ordering, it is 292ms. Thus. the cost of the stronger consistency provided by total write ordering compared to eventual is that it can increase the time-to-connect latency by up to 2×. Note that the 2× inflation is a worst-case estimate, i.e., it will impact the time-to-connect latency only if a read request arrives at a replica while a write is under propagation to that replica, as we show below.

6.2.2.3 Time-to-connect to "moving" endpoints

We evaluate the time-to-connect to a moving destination as a function of the mobility (or update) rate. This experiment is performed with the help of msocket [9], a user-level socket library that interoperates with Auspice. The *end-to-end time-to-connect* here is measured as the latency to look up an up-to-date address of the destination (or the time-to-connect as defined in §6.1.2) plus the time for msocket to successfully establish a TCP connection between the client and the mobile destination. This e2e-time-to-connect also incorporates the impact of timeouts and retried lookups if the client happens to have obtained a stale value.

The experiment is conducted on PlanetLab and consists of a single msocket client and a single mobile msocket server that is "moving" by changing its listening port number on a remote machine, and updating the name record replicated on three Auspice name servers accordingly. A successful connection setup delay using msocket is takes 2 RTTs (2 $\times$ 105 ms) [9]. The values of the update propagation latency d_i and the lookup latency l_i are 250 ms and 20 ms respectively, and the update rate w_i varies from 1/1024/s to 1/s. The timeout value (T) in our experiment is dependent on the RTT between the client and the server. If the client attempts to connect to the server on a port which the server is not listening on, the server immediately returns an error response to the client. Specifically, the timeout value is either 1 or 2 RTTs with equal probability depending on whether the connection failed during the first or the second round-trip of msocket's connection setup. The client sends lookups at a rate of 10/s (but this rate does not affect the time-to-connect), and both lookups and updates inter-arrival times are exponentially distributed.

Figure 6.3 shows the distribution of the time-to-connect with update propagation delays entailed by eventual consistency. For low-to-moderate mobility rates ($< \frac{1}{64s}$), we find that all time-to-connect values are close to 230 ms, of which 20ms is the lookup latency, and 210ms is msocket's connection setup latency. The reason the client is able to obtain the correct value upon first lookup in all cases is that the update propagation latency of 250ms is much smaller than the average inter-update interval (64s). The update propagation delay becomes a non-trivial fraction of the inter-update interval at high mobility rates of $\approx$1/sec that results in 26% of lookups returning stale values. The mean e2e-time-to-connect increases to 302 ms for an update rate of 1/sec, which suggests that Auspice's time-to-connect is limited by network propagation delays in this regime. Nevertheless, once a connection is successfully established, individual migration can quickly resynchronize the connection in $\approx$two round-trips between the client and the mobile without relying on Auspice (not shown here).

We have also developed an analytical model to predict time-to-connect values [1]. Figure 6.3 also shows that the time-to-connect as predicted by our analytical model are close to those observed in the experiment, thereby re-affirming our design.

6.2.2.4 Reconfiguration overhead vs. responsiveness

In the next experiment, we show how Auspice can choose the epoch length and reconfiguration trigger so as to limit the overhead of reconfiguration (as in §6.1.2.5) while being responsive to changes in demand geo-locality. The workload changes the geo-distribution of demand for each name every 300 sec. For each name, we change the regions from which most requests arise so that changing the placement of replicas becomes necessary to minimize lookup latencies. The experiment is performed on a local cluster with a workload of 1000 names with characteristics as described in Table 6.1. The epoch length of group changes is chosen to be 75 sec, which ensures that group changes result in less than 10% overhead to the system (§6.1.2.5).

We show the lookup latencies of Auspice and the message overhead of reconfiguration in the experiment in Figue 6.4. We find that Auspice takes two epochs

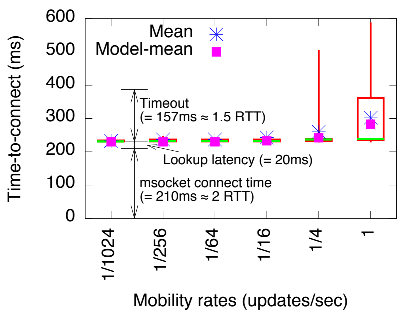


Figure 6.3. Time-to-connect ≈ lookup latency for moderate mobility rates (<1/100s) as Auspice returns up-to-date responses w.h.p., but sharply rises thereafter.

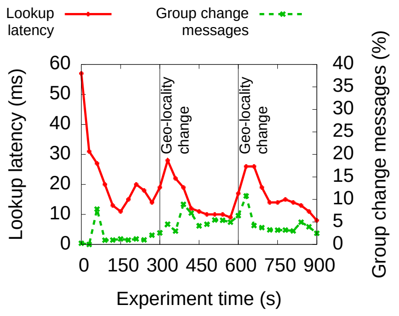


Figure 6.4. Auspice adapts to varying demand geo-locality in two epochs each 75s long with a (tunable) reconfiguration message overhead of 3.6%.

to infer a change in the geo-distribution of demand and to adapt to it. This result suggests that Auspice can optimize lookup latencies provided the geo-distribution of demand for a name remains stable for a few epochs. Further, we measured the overhead of reconfiguration messages in this experiment to be 3.6%. The overhead is less than the expected overhead of 10% because not all names are reconfigured in every epoch.

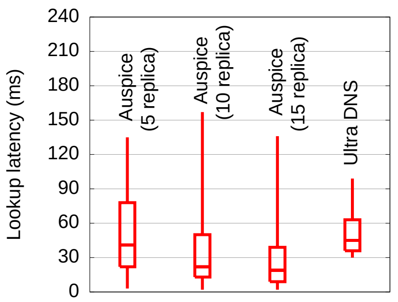


Figure 6.5. Lookup latency: Auspice with 5 replicas is comparable to UltraDNS (16 replicas); Auspice with 15 replicas has 60% lower latency than UltraDNS.

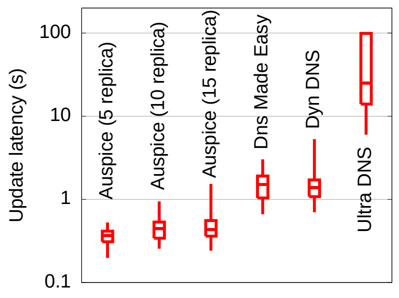


Figure 6.6. Update propagation delay: Auspice with 5 replicas is 1.0 to 24.7 secs lower than three top-tier managed DNS service providers.

6.2.3 Auspice vs. managed DNS providers

Can demand-aware replication benefit commercial managed DNS providers that largely rely on statically replicating today's (hardly mobile) domain names? To investigate this, we compare Auspice against three top-tier providers, UltraDNS, DynDNS, and DNSMadeEasy that offer geo-replicated authoritative DNS services widely used by enterprises (e.g., Dyn provides DNS service for Twitter).

6.2.3.1 Lookup latency

We compare Auspice to UltraDNS for a workload of lookups for domain names serviced by the provider. We identify 316 domain names among the top 10K Alexa websites serviced by this provider, and determine the geo-distribution of lookups for each name from their data [3]. For each name, we measure the latency for 1000 lookups from across 100 PlanetLab nodes. We ensure that lookups are served from the name servers maintained by the provider by requesting the address for a new random sub-domain name each time, e.g, `xqf4p.google.com` instead of `google.com`, that is unlikely to exist in a cache and requires an authoritative lookup. Auspice name servers are deployed across a total of 80 PlanetLab locations while UltraDNS has 16 known server locations [101]. We evaluate Auspice for three configurations with 5, 10, and 15 replicas of a name respectively.

Figure 6.5 shows the lookup latencies of names for Auspice and for UltraDNS. UltraDNS incurs a median latency of 45 ms with 16 replicas, while Auspice incurs 41 ms, 22 ms, and 18 ms respectively with 5, 10, and 15 replicas. With 5 replicas, Auspice's performance is comparable to UltraDNS with one-third the replication cost. With 15 replicas, Auspice incurs 60% lower latency for a comparable cost. The comparison against the other two, Dyn and DNSMadeEasy, is qualitatively similar [1]. Thus, Auspice's demand-aware replication achieves a better cost–performance tradeoff compared to static replication.

6.2.3.2 Update propagation delay

To measure update propagation delays, we purchase DNS service from three providers for separate domain names. All providers replicate a name at 5 locations across US and Europe for the services we purchased. We issue address updates for the domain name serviced by that provider and then immediately start lookups to the authoritative name servers for our domain name. These authoritative name servers can be queried only via an anycast IP address, i.e., servers at different locations advertise the same externally visible IP address. Therefore, to maximize the number of provider locations queried, we send queries from 50 random PlanetLab nodes. From each location, we periodically send queries until all authoritative name server replicas return the updated address. The update propagation latency at a node is the time between when the node starts sending lookup to when it receives the updated address. The latency of an update is the the maximum update latency measured at any of the nodes. We measure latency of 100 updates for each provider.

To measure update latencies for Auspice, we replicate 1000 names at a fixed number of PlanetLab nodes across US and Europe. The number of nodes is chosen to be 5, 10, and 20 across three experiments. A client sends an update to the nearest node and waits for update confirmation messages from all replicas. The latency of an update is the time difference between when the client sent an update and when it received the update confirmation message from all replicas (an upper bound on the update propagation delay). We show the distribution of measured update latencies for Auspice and for three managed DNS providers in Figure 6.6.

Auspice incurs lower update propagation latencies than all three providers for an equal or greater number of replica locations for names. We were unable to ascertain from UltraDNS why their update latencies are an order of magnitude higher than network propagation delays, but this finding is consistent with a recent study [101] that has shown latencies of up to tens of seconds for these providers. Indeed, some providers even distinguish themselves by advertising shorter update propagation delays than competitors [101].

6.2.4 Sensitivity analyses and other results.

We have conducted a comprehensive evaluation of the sensitivity of Auspice's performance-cost trade-offs to workload and system parameters across scales varying by several orders of magnitude. These include workload parameters such as geo-locality, read-to-write rate ratio, ratio of device-to-service names, etc. and system parameters such as the fault-tolerance threshold, capacity utilization, perturbation knob, the tunable overhead of replica reconfiguration, etc. using a combination of simulation and system experiments. These results do not qualitatively change the above findings, and are deferred to the technical report [1].

6.3 Related work

Our work draws on lessons learned from an enormous body of prior work on distributed systems as discussed below.

DNS. Many have studied issues related to performance, scalability, load balancing, or denial-of-service vulnerabilities in DNS's resolution infrastructure [91, 98, 28, 44]. Several DHT-based alternatives have been put forward [98, 37, 90] and we compare against one representative proposal, Codons [98]. In general, DHT-based designs are ideal for balancing load across servers, but are less well-suited to scenarios with a large number of service replicas that have to coordinate upon updates, and are at odds with scenarios requiring placement of replicas close to pocket of demand. In comparison, Auspice uses a planned placement approach.

Vu et al. describe DMap [116], an in-network DHT scheme that is similar in spirit to Random-M as evaluated in our experiments (§6.2) (with a more direct comparison in [1]), showing that demand-aware placement can dramatically outperform randomized placement.

Server selection. Many prior systems have addressed the server selection problem with data or services replicated across a wide-area network. Examples include anycast services [56, 26, 118] to map users to the best server based on server load or network path characteristics. These systems as well as CDNs and cloud hosting providers share our goals of proximate server selection and load balance given a fixed placement of server replicas. Auspice differs in that it additionally considers replica placement itself as a degree of freedom in achieving latency or load balance.

Dynamic placement. We were unable to find prior systems that *automatically* reconfigure the *geo-distributed* replica locations of frequently *mutable objects* while preserving *consistency* (i.e., those satisfying all four italicized properties). However, reconfigurable placement has been studied for static or slow changing content [62] or within a single datacenter, or without replication. For example, Volley [14] optimizes the placement of mutable data objects based on the geo-distribution of accesses and is similar in spirit to Auspice in this respect, however it implicitly assumes a single replica for each object, so it does not have to worry about high update rates or replica coordination overhead.

Auspice is related to many distributed key-value stores [8, 47, 5], most of which are optimized for distribution within, not across, data centers. Some (e.g., Cassandra) support a geo-distributed deployment using a fixed number of replica sites. Spanner [36] is a geo-distributed data store that synchronously replicates data ("directories") across datacenters with a semi-relational database abstraction. Compared to Spanner, Auspice does not provide any guarantees on operations spanning multiple records, but unlike Spanner's geographic placement of replicas that "administrators control" by creating a "menu of named options", Auspice automatically reconfigures the number and placement of replicas so as to reduce lookup latency and update cost. Furthermore, Spanner assigns a large number of directory objects to a much smaller number of fixed Paxos groups; Auspice supports an arbitrarily reconfigurable Paxos group per object based on principles in recent theoretical work on reconfigurable consensus, e.g., Vertical Paxos [73] and the more recent report on Viewstamped Replication Revisited [77].

6.4 Conclusions

We presented the design, implementation, and evaluation of Auspice, a scalable, geo-distributed, key-value store. At the core of Auspice is a placement engine for replicating records to achieve low lookup latency, low update cost, and high availability. We have extensively evaluated Auspice for a representative application: a global name service to provide name-to-address mapping for mobile devices. Our evaluation shows that Auspice's placement strategy can significantly improve the performance-cost tradeoffs struck both by commercial managed DNS services employing simplistic replication strategies today as well as previously proposed DHT-based replication alternatives with or without high mobility.

CHAPTER 7

PROPOSED RESEARCH PLAN

The research towards the completion of this thesis would be conducted in following four phases as outlined in Figure 7.1.

Phase	Project	Duration	Brief description
1	Shrink (Ch. 5)	2 months	Algorithm design and trace-based simulation
2	Shrink (Ch. 5)	2 months	Prototype development and evaluation
3	Auspice (Ch. 6)	2 months	Evaluation in a wide-area testbed
4	Dissertation	2 months	Documentation of research in Phase 1-3

Figure 7.1. Proposed work and timeline

7.1 Shrink algorithm design and trace-driven evaluation

Our first goal would be to design algorithms for problems in Section 5.2.1 and iteratively refine them by conducting trace-driven evaluation with our datasets. Trace-driven evaluation can be accomplished faster than actual experimentations, and will help us iterate quickly over our algorithms.

7.2 Shrink prototype development and evaluation

The second phase would involve a developing research prototype, whose main components are following: **(1) Shrink core:** This module would implement the algorithms designed in the first phase, namely those for deciding the set of active elements in a datacenter, and determining network routing and load balancing decisions. **(2) Server and switch daemons:** The role of server and switch daemons is discussed in Section 5.2.2. These daemons could be integrated within a process already running at that server, e.g., the proxy software [40], or may be implemented as a stand-alone processes. **(3) Client:** The client module will send HTTP requests to simulate a datacenter workload and measure end-to-end performance.

We plan to conduct an experimental evaluation with the above prototype on a cloud platform, e.g, Amazon EC2 [4], as well on the 12-node Skuld cluster within our department. We expect the experiment evaluation to be an iterative process as well in which experimental findings lead us to reconsider both the designed algorithms and the system implementation.

7.3 Auspice evaluation in a wide-area testbed

The research in this phase would be an extension of the evaluation presented in Chapter 6. Although emulation-based results show that Auspice's placement strategy outperforms state-of-the-art systems, the results therein do not fully validate if the Auspice system can deliver similar gains in a wide-area setting or at a workload with a greater number of records. In particular, the variable latency and loss in a wide-area setting as well as the overhead of replicating a greater number of records in a demand-aware manner may reduce the latency improvements that we have demonstrated. We plan to evaluate Auspice on the PlanetLab testbed for a larger-sized workload to provide a more realistic estimate of Auspice's performance and cost improvements.

7.4 Dissertation

In the concluding phase, we will incorporate research findings in the above phases in the thesis draft to be presented to the committee.

BIBLIOGRAPHY

[1] A Global Name Service for a Highly Mobile Internetwork. UMass SCS Technical Report, 2013 and 2014. `https://web.cs.umass.edu/publication`.

[2] Akamai. `http://www.akamai.com/`.

[3] Alexa web information service. `http://www.alexa.com`.

[4] Amazon EC2. `http://aws.amazon.com/ec2/`.

[5] Cassandra. `http://cassandra.apache.org`.

[6] Cisco. Best Practices in Network Planning and Traffic Engineering. `http://www.nanog.org/meetings/nanog52/presentations/Sunday/maghbouleh-bestpractices-tutorial.pdf`.

[7] Geo-Replication Performance Gains with Microsoft SQL Server 2008 Running on Windows Server 2008. `http://technet.microsoft.com/en-us/library/dd263442(v=sql.100).aspx`.

[8] Mongo DB. `http://www.mongodb.org/`.

[9] msocket: System Support for Developing Seamlessly Mobile, Multipath, and Middlebox-Agnostic Applications. `http://sites.google.com/site/msockettech/home`.

[10] Twitter. `https://twitter.com/`.

[11] Abhigyan, Mishra, Aditya, Kumar, Vikas, and Venkataramani, Arun. Beyond MLU : An Application Centric Comparison of Traffic Engineering Schemes. In *UMASS Computer Science Technical Report* (UM-CS-2010-012).

[12] Abhigyan, Venkataramani, A, and Sitaraman, R. Distributing Content Simplifies ISP Traffic Engineering. *Technical Report*. `http://peoplecsumassedu/~abhigyan/NCDNpdf`.

[13] Abts, Dennis, Marty, Michael R., Wells, Philip M., Klausler, Peter, and Liu, Hong. Energy proportional datacenter networks. In *Proceedings of the 37th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2010), ISCA '10, ACM, pp. 338–347.

[14] Agarwal, Sharad, Dunagan, John, Jain, Navendu, Saroiu, Stefan, Wolman, Alec, and Bhogan, Harbinder. Volley: automated data placement for geo-distributed cloud services. In *NSDI* (2010).

[15] Andersen, David, Balakrishnan, Hari, Kaashoek, Frans, and Morris, Robert. *Resilient overlay networks*. ACM, 2001.

[16] Andrews, M., Anta, A.F., Zhang, L., and Zhao, Wenbo. Routing for energy minimization in the speed scaling model. In *INFOCOM, 2010 Proceedings IEEE* (2010), pp. 1–9.

[17] Antoniades, D, Markatos, EP, and Dovrolis, C. One-Click Hosting Services: A File-Sharing Hideout. In *IMC* (2009).

[18] Apple. HTTP Live Streaming. `http://bitly/MgoUED`.

[19] Applegate, D, Archer, A, Gopalakrishnan, V, Lee, S, and Ramakrishnan, K K. Optimal content placement for a large-scale VoD system. In *Co-NEXT* (2010).

[20] Applegate, D, and Cohen, E. Making routing robust to changing traffic demands: algorithms and evaluation. *IEEE/ACM Trans Netw 14* (December 2006), 1193–1206.

[21] AT&T. Content Distribution, 2011. `http://bitly/Lefgj2`.

[22] Azar, Y, Cohen, E, Fiat, A, Kaplan, H, and Racke, H. Optimal oblivious routing in polynomial time. In *STOC* (2003).

[23] Barbir, A, Cain, Brad, Nair, Raj, and Spatscheck, Oliver. Known content network (cn) request-routing mechanisms. *Internet Engineering Task Force RFC 3568* (2003).

[24] Barroso, Luiz André, and Hölzle, Urs. The case for energy-proportional computing. *IEEE computer 40*, 12 (2007), 33–37.

[25] Beheshti, N, Ganjali, Y, Ghobadi, M, McKeown, N, and Salmon, G. Experimental study of router buffer sizing. In *IMC* (2007).

[26] Bhattacharjee, Samrat, and et al. Application-Layer Anycasting. In *IEEE INFOCOM* (1997).

[27] Bohrer, Pat, Elnozahy, ElmootazbellahN., Keller, Tom, Kistler, Michael, Lefurgy, Charles, McDowell, Chandler, and Rajamony, Ram. The case for power management in web servers. In *Power Aware Computing*, Robert Graybill and Rami Melhem, Eds., Series in Computer Science. Springer US, 2002, pp. 261–289.

[28] Brownlee, N., Claffy, K.C., and Nemeth, E. Dns measurements at a root server. In *GLOBECOM '01. IEEE* (2001).

[29] Carpathia. `http://www.carpathia.com/assets/files/carpathialoadbalancesheet.pdf`.

[30] Cast, Edge. `http://wwwedgecastcom/solutions/licensed-cdn/`.

[31] Cha, Meeyoung, Kwak, Haewoon, Rodriguez, Pablo, Ahn, Yong-Yeol, and Moon, Sue. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2007), IMC '07, ACM, pp. 1–14.

[32] Chiaraviglio, Luca, Mellia, Marco, and Neri, Fabio. Minimizing isp network energy cost: formulation and solutions. *IEEE/ACM Trans. Netw. 20*, 2 (Apr. 2012), 463–476.

[33] Cisco. Visual Networking Index, 2011. `http://bitly/KXDUaX`.

[34] Cohen, B. BitTorrent Protocol. `http://bitly/KDhQV1`.

[35] Cole, RG, and Rosenbluth, JH. Voice over IP performance monitoring. In *SIGCOMM CCR* (2001).

[36] Corbett, James C., Dean, Jeffrey, Epstein, Michael, and et al. Spanner: Google's globally distributed database. *ACM Trans. Comput. Syst.* (2013).

[37] Cox, Russ, Muthitacharoen, Athicha, and Morris, Robert. Serving dns using a peer-to-peer lookup service. In *IPTPS* (2002).

[38] Davison, Brian D. A web caching primer. *Internet Computing, IEEE 5*, 4 (2001), 38–45.

[39] DeCandia, Giuseppe, Hastorun, Deniz, Jampani, Madan, Kakulapati, Gunavardhan, Lakshman, Avinash, Pilchin, Alex, Sivasubramanian, Swaminathan, Vosshall, Peter, and Vogels, Werner. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev. 41*, 6 (Oct. 2007).

[40] Delivery, Squid: Optimising Web. Squid: Optimising Web Delivery. `http://www.squid-cache.org/`.

[41] Dilley, John, Maggs, Bruce M, Parikh, Jay, Prokop, Harald, Sitaraman, Ramesh K, and Weihl, William E. Globally Distributed Content Delivery. *IEEE Internet Computing 6*, 5 (2002), 50–58.

[42] DiPalantino, D, and Johari, R. Traffic Engineering vs Content Distribution: A Game Theoretic Perspective. In *INFOCOM* (2009).

[43] Dixon, Colin, Uppal, Hardeep, Brajkovic, Vjekoslav, Brandon, Dane, Anderson, Thomas, and Krishnamurthy, Arvind. Ettm: a scalable fault tolerant network manager. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation* (Berkeley, CA, USA, 2011), NSDI'11, USENIX Association, pp. 7–7.

[44] DNSSEC. DNS Threats & Weaknesses of the Domain Name System, 2012. `http://www.dnssec.net/dns-threats.php`.

[45] Elwalid, A, Jin, C, Low, S, and Widjaja, I. MATE: MPLS adaptive traffic engineering. In *INFOCOM* (2001).

[46] Erman, Jeffrey, Gerber, Alexandre, Ramadrishnan, K. K., Sen, Subhabrata, and Spatscheck, Oliver. Over the top video: the gorilla in cellular networks. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference* (New York, NY, USA, 2011), IMC '11, ACM, pp. 127–136.

[47] Escriva, Robert, Wong, Bernard, and Sirer, Emin Gün. Hyperdex: a distributed, searchable key-value store. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication* (New York, NY, USA, 2012), SIGCOMM '12, ACM, pp. 25–36.

[48] Feamster, Nick, Borkenhagen, Jay, and Rexford, Jennifer. Guidelines for interdomain traffic engineering. *SIGCOMM Comput. Commun. Rev. 33*, 5 (Oct. 2003), 19–30.

[49] Fortz, B, Rexford, J, and Thorup, M. Traffic engineering with traditional IP routing protocols. *Communications Magazine, IEEE 40*, 10 (2002), 118–124.

[50] Fortz, B, and Thorup, M. Internet traffic engineering by optimizing OSPF weights. In *INFOCOM* (2000).

[51] Fortz, B, and Thorup, M. Optimizing ospf/is-is weights in a changing world. *JSAC* (May 2002).

[52] Foundation, Open Networking. Open Networking Foundation. `https://www.opennetworking.org/`.

[53] Fraleigh, C, Moon, S, Lyles, B, Cotton, C, Khan, M, Moll, D, Rockell, R, Seely, T, and Diot, C. Packet-Level Traffic Measurements from the Sprint IP Backbone. In *IEEE Network* (2003).

[54] Frank, B, Poese, I, Smaragdakis, G, Uhlig, S, and Feldmann, A. Content-aware Traffic Engineering. *ArXiv e-prints* (2012).

[55] Frank, B, Poese, I, Smaragdakis, G, Uhlig, S, and Feldmann, A. Content-aware traffic engineering. In *SIGMETRICS* (2012).

[56] Freedman, Michael J., Lakshminarayanan, Karthik, and Mazires, David. Oasis: Anycast for any service, 2006.

[57] Gadde, Syam, Chase, Jeff, and Rabinovich, Michael. Web caching and content distribution: A view from the interior. *Computer Communications 24*, 2 (2001), 222–231.

[58] Gao, Peter Xiang, Curtis, Andrew R., Wong, Bernard, and Keshav, Srinivasan. It's not easy being green. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication* (New York, NY, USA, 2012), SIGCOMM '12, ACM, pp. 211–222.

[59] Gill, P, Arlitt, M, Li, Z, and Mahanti, A. Youtube traffic characterization: a view from the edge. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2007), IMC '07, ACM, pp. 15–28.

[60] Guenter, B., Jain, N., and Williams, C. Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. In *INFOCOM, 2011 Proceedings IEEE* (2011), pp. 1332–1340.

[61] Gummadi, KP, Dunn, RJ., Saroiu, S, Gribble, SD, Levy, HM, and Zahorjan, J. Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. In (2003).

[62] Gwertzman, James, and Seltzer, Margo. The case for geographical push caching. In *IEEE HotOS Workshop* (May 1995).

[63] Heller, Brandon, Seetharaman, Srini, Mahadevan, Priya, Yiakoumis, Yiannis, Sharma, Puneet, Banerjee, Sujata, and McKeown, Nick. Elastictree: saving energy in data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation* (Berkeley, CA, USA, 2010), NSDI'10, USENIX Association, pp. 17–17.

[64] Hengartner, Moon, S, Mortier, R, and Diot, C. Detection and analysis of routing loops in packet traces. In *IMW* (2002).

[65] HP. The edge of bandwidth growth. `http://bitly/HwXtUO`.

[66] Huang, Cheng, Li, Jin, and Ross, Keith W. Can internet video-on-demand be profitable? In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2007), SIGCOMM '07, ACM, pp. 133–144.

[67] IBM. ILOG CPLEX. `http://ibmco/KRuqhB`.

[68] Interface, Akamai NetSession. `http://www.akamai.com/client`.

[69] Jiang, W, Zhang-Shen, R, Rexford, J, and Chiang, M. Cooperative content distribution and traffic engineering in an ISP network. In *SIGMETRICS* (2009).

[70] Kandula, S, Katabi, D, Davie, B, and Charny, A. Walking the tightrope: responsive yet stable traffic engineering. In *SIGCOMM* (2005).

[71] Kwak, Haewoon, Lee, Changhyun, Park, Hosung, and Moon, Sue. What is twitter, a social network or a news media? In *WWW* (2010).

[72] Lamport, Leslie. The part-time parliament. *ACM Trans. Comput. Syst. 16*, 2 (May 1998), 133–169.

[73] Lamport, Leslie, Malkhi, Dahlia, and Zhou, Lidong. Vertical paxos and primary-backup replication. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing* (2009), PODC '09.

[74] Level-3, 2012. `http://www.level3.com/`.

[75] Level3. Content Delivery Network, 2011. `http://bitly/LvsIDm`.

[76] Lin, M., Wierman, A., Andrew, L. L. H., and Thereska, E. Dynamic right-sizing for power-proportional data centers. *Networking, IEEE/ACM Transactions on* (2012).

[77] Liskov, Barbara, and Cowling, James. Viewstamped replication revisited. Tech. Rep. MIT CSAIL-TR-2012-021, 2012.

[78] Liu, Zhenhua, Lin, Minghong, Wierman, Adam, Low, Steven H., and Andrew, Lachlan L.H. Greening geographical load balancing. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems* (New York, NY, USA, 2011), SIGMETRICS '11, ACM, pp. 233–244.

[79] Live, PP. `http://wwwpplivecom/`.

[80] Lu, Tan, Chen, Minghua, and Andrew, L.L.H. Simple and effective dynamic provisioning for power-proportional data centers. *Parallel and Distributed Systems, IEEE Transactions on 24*, 6 (2013), 1161–1171.

[81] Madhyastha, Harsha V., Isdal, Tomas, Piatek, Michael, Dixon, Colin, Anderson, Thomas, Krishnamurthy, Arvind, and Venkataramani, Arun. iplane: an information plane for distributed services. In *OSDI* (Berkeley, CA, USA, 2006), USENIX Association, pp. 367–380.

[82] Mathew, V., Sitaraman, R.K., and Shenoy, P. Energy-aware load balancing in content delivery networks. In *INFOCOM, 2012 Proceedings IEEE* (2012), pp. 954–962.

[83] Media, Streaming. Telco-CDN Whitepapers. `http://bitly/GUDrUZ`.

[84] Nedevschi, Sergiu, Popa, Lucian, Iannaccone, Gianluca, Ratnasamy, Sylvia, and Wetherall, David. Reducing network energy consumption via sleeping and rate-adaptation. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2008), NSDI'08, USENIX Association, pp. 323–336.

[85] Nielsen. Online Video Usage Up 45%. `http://bitly/MiXiPU`.

[86] Nygren, E, Sitaraman, R K, and Sun, J. The Akamai network: a platform for high-performance internet applications. *SIGOPS Oper Syst Rev* (August 2010).

[87] of the Internet Reports, Akamai State. `http://www.akamai.com/stateoftheinternet/`.

[88] OSPF, Cisco Configuring. `http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/`.

[89] Papagiannaki, K, Moon, S, Fraleigh, C, Thiran, P, and Diot, C. Measurement and Analysis of Single-Hop Delay on an IP Backbone Network. In *IEEE JSAC* (2006).

[90] Pappas, V., Massey, D., Terzis, A., and Zhang, L. A comparative study of the dns design with dht-based alternatives. In *INFOCOM* (2006).

[91] Pappas, Vasileios, Xu, Zhiguo, Lu, Songwu, Massey, Daniel, Terzis, Andreas, and Zhang, Lixia. Impact of configuration errors on dns robustness. In *SIGCOMM* (2004).

[92] Patel, Parveen, Bansal, Deepak, Yuan, Lihua, Murthy, Ashwin, Greenberg, Albert, Maltz, David A., Kern, Randy, Kumar, Hemant, Zikos, Marios, Wu, Hongyu, Kim, Changhoon, and Karri, Naveen. Ananta: cloud scale load balancing. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM* (New York, NY, USA, 2013), SIGCOMM '13, ACM, pp. 207–218.

[93] Peterson, L, Bavier, A, Fiuczynski, M, and MuirLy, S. Experiences building planetlab. In *OSDI* (2006).

[94] Project, TOTEM. `http://totem.info.ucl.ac.be/`.

[95] Qiu, Lili, Yang, Yang Richard, Zhang, Yin, and Shenker, Scott. On selfish routing in internet-like environments. *IEEE/ACM Trans. Netw. 14*, 4 (Aug. 2006), 725–738.

[96] Qureshi, Asfandyar, Weber, Rick, Balakrishnan, Hari, Guttag, John, and Maggs, Bruce. Cutting the Electric Bill for Internet-Scale Systems. In *ACM SIGCOMM* (Barcelona, Spain, August 2009).

[97] Rahul, Hariharan, Kasbekar, Mangesh, Sitaraman, Ramesh, and Berger, Arthur. Towards Realizing the Performance and Availability Benefits of a Global Overlay Network, 2006. `http://hdl.handle.net/1721.1/30580`.

[98] Ramasubramanian, Venugopalan, and Sirer, Emin G. The design and implementation of a next generation name service for the internet. In *SIGCOMM* (2004).

[99] Ramasubramanian, Venugopalan, and Sirer, Emin Gun. Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In *NSDI* (2004).

[100] Rao, Lei, Liu, Xue, Xie, Le, and Liu, Wenyu. Minimizing electricity cost: Optimization of distributed internet data centers in a multi-electricity-market environment. In *INFOCOM, 2010 Proceedings IEEE* (2010), pp. 1–9.

[101] Read, Jason. Comparison and analysis of managed dns providers, Aug 2012. Cloud Harmony Inc.

[102] Renzenbrink, Tessel. Data Centers Use 1.3% of World's Total Electricity. A Decline in growth, 2011. `http://www.techthefuture.com/energy/data-centers-use-1-3-of-worlds-total-electricity-a-decline-in-growth/`.

[103] Report, ATLAS Internet Observatory 2009 Annual. `http://www.nanog.org/meetings/nanog47/presentations/-Monday/Labovitz_ObserveReport_N47_Mon.pdf`.

[104] Rexford, J. Route optimization in IP networks. In *Handbook of Optimization in Telecommunications, Springer Science + Business Media* (February,2006).

[105] RFC. 2616. `http://wwwietforg/rfc/rfc2616txt`.

[106] Roughgarden, Tim, and Tardos, Éva. How bad is selfish routing? *J. ACM 49*, 2 (Mar. 2002), 236–259.

[107] Saroiu, Stefan, Gummadi, Krishna P., Dunn, Richard J., Gribble, Steven D., and Levy, Henry M. An analysis of internet content delivery systems. *SIGOPS Oper. Syst. Rev. 36*, SI (Dec. 2002), 315–327.

[108] Savage, S, Anderson, T, Aggarwal, A, Becker, D, Cardwell, N, Collins, A, Hoffman, E, Snell, J, Vahdat, A, Voelker, G, and Zahorjan, J. Detour: Informed Internet routing and transport. In *IEEE MICRO* (1999).

[109] Sharma, Abhigyan, Venkataramani, A, and Sitaraman, R. Distributing content simplifies isp traffic engineering. In *SIGMETRICS* (2013).

[110] Study, IPOQUE Internet. `http://wwwipoquecom/resources/internet-studies/internet-study-2008_2009`.

[111] Su, AJ, Choffnes, DR, Kuzmanovic, A, and Bustamante, F. Drafting behind Akamai. In *SIGCOMM* (2006).

[112] Topology, Abilene. `http://bit.ly/Lf8k7a`.

[113] Vasić, Nedeljko, Bhurat, Prateek, Novaković, Dejan, Canini, Marco, Shekhar, Satyam, and Kostić, Dejan. Identifying and using energy-critical paths. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies* (New York, NY, USA, 2011), CoNEXT '11, ACM, pp. 18:1–18:12.

[114] Verizon. HBO for FIOS Customers. `http://bitly/JQ2dn8`.

[115] Verizon. Velocix at Verizon, 2011. `http://bitly/LlqGn3`.

[116] Vu, T., Baid, A., Zhang, Y., Nguyen, T. D., Fukuyama, J., Martin, R. P., and Raychaudhuri, D. Dmap: A shared hosting scheme for dynamic identier to locator mappings in the global internet. In *Proceedings of IEEE ICDCS* (2012).

[117] Wang, H, Xie, H, Qiu, L, Yang, Y R, Zhang, Y, and Greenberg, A. COPE: Traffic Engineering in Dynamic Networks. In *SIGCOMM* (2006).

[118] Wendell, Patrick, Jiang, Joe Wenjie, Freedman, Michael J, , and Rexford, Jennifer. DONAR: Decentralized Server Selection for Cloud Services. In *SIGCOMM* (2010).

[119] Wessels, Duane. *Squid: the definitive guide*. O'Reilly, 2009.

[120] Williams, A, Arlitt, M, Williamson, C, and Barker, K. Web Workload Characterization: Ten Years Later. In *Web Content Delivery, Volume 2* (2005).

[121] Xie, H, Yang, Y R, Krishnamurthy, A, Liu, Y G, and Silberschatz, A. P4P: Provider Portal for Applications. In *SIGCOMM* (2008).

[122] Zhang, C, Liu, Y, Gong, W, Kurose, J, Moll, R, and Towsley, D. On optimal routing with multiple traffic matrices. In *INFOCOM* (2005).

[123] Zhang, Mingui, Yi, Cheng, Liu, Bin, and Zhang, Beichuan. Greente: Power-aware traffic engineering. In *Network Protocols (ICNP), 2010 18th IEEE International Conference on* (2010), pp. 21–30.