CONTENT PLACEMENT AS A KEY TO A CONTENT-DOMINATED, HIGHLY MOBILE INTERNET

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

September 2015

College of Information and Computer Sciences

CONTENT PLACEMENT AS A KEY TO A CONTENT-DOMINATED, HIGHLY MOBILE INTERNET

A Dissertation Presented

by

ABHIGYAN SHARMA

Approved as to style and content by:

Arun Venkataramani, Chair

Ramesh Sitaraman, Member

Donald Towsley, Member

Lixin Gao, Member

James Allan, Chair
College of Information and Computer Sciences

ABSTRACT

CONTENT PLACEMENT AS A KEY TO A CONTENT-DOMINATED, HIGHLY MOBILE INTERNET

SEPTEMBER 2015

ABHIGYAN SHARMA

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

Most of the Internet traffic is content, and most of the Internet connected hosts are mobile. Our work focuses on the design of infrastructure services needed to support such a content-dominated, highly mobile Internet. In the design of these services, three sets of decisions arise frequently: (1) *content placment* for selecting the locations where a content is placed, (2) *request redirection* for selecting the location where a particular request is served from and (3) *network routing* for selecting the physical path between clients and the services they are accessing. Our central thesis is that content placement is a powerful factor, and is often more powerful than redirection and routing, in determining the cost, performance and energy-related metrics for these services. In support of this thesis, we consider three types of infrastructure.

Internet service provider (ISP): In an ISP carrying content-dominated traffic, we show that combinations of simple placement and routing schemes are effec-

tive in optimizing an ISP's performance and cost objectives. Further, we show that effective content placement contributes more than optimizing network routing to achieve an ISP's objectives. Our findings question the value of traditional ISP traffic engineering schemes that optimize routing alone, while simplifying the task of traffic engineering for the operators.

Global name service (GNS): We design and implement a GNS, Auspice, that resolves names to network addresses for highly mobile entities, thereby providing a key building block for establishing communication between mobile entities in the Internet. A key distinction between Auspice and other name services is a *demand-aware* replica placement engine that intelligently replicates name records to provide low lookup latency, low update cost, and high availability. In our experiments, Auspice's placement scheme enables it to significantly outperform commercial managed DNS providers, DHT-based replication as well as static placement schemes that use the same redirection scheme as Auspice.

Content datacenter (CDC): Content datacenters cache and serve content to improve user-perceived performance for content accesses. In a CDC, we quantify the tradeoff between energy savings via consolidation and the user-perceived performance impact based on a real CDC workload. A key insight, supported via experiments, is that despite server consolidation, a simple caching scheme is able to achieve cache hit rates close to an unconsolidated datacenter, which helps mitigate the impact of consolidation on user-perceived latencies. Further, our work is the first to propose a network-aware server consolidation approach that enables additional network energy savings over network-unaware server consolidation schemes for common datacenter topologies.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Most of the Internet traffic is content, and most of the Internet connected hosts are mobile. A key example of the content-dominated nature of the Internet is that online video alone accounts for nearly 67% of traffic today and is expected to grow to nearly 80% by 2018 [45]. The evidence of a highly mobile Internet is that close to 5 billion mobile devices connect to the Intenet today and they generate more traffic than wired devices [42]. Due to the prevalence of content traffic and mobile users, content-based services and mobile applications are cornerstones of today's Internet-based economy.

A content-dominated, highly mobile Internet needs several forms of infrastructure support to sustain itself. It needs network switches and links with sufficient capacity to carry traffic to end users. It needs servers to sustain and accelerate delivery of content. It also needs infrastructure to help establish and maintain connections in the presence of high network mobility (or changing of addresses) of connection end-points. This thesis seeks to design *infrastructure services* that manage the resources on such infrastructures towards achieving their desired goals.

1.1 Infrastructure characteristics

Our service designs are strongly influenced by the following characteristics of the infrastructures that support a content-dominated, highly mobile Internet.

Cost-intensive: There is a substantial captial and operational cost in running a large-scale infrastructure [80], and which causes a sigificant reduction in the profit

margins of the infrastructure owner. A prominent example of cost-intensive, low-profit infrastructures are Internet service provider (ISP) networks [159]. Thus, a main goal of service design is to reduce cost while meeting performance and other constraints.

Geo-distributed: These infrastructures are often highly geo-distributed, e.g., a large ISP or a content delivery network (CDN) has Points-of-Presence at hundreds of locations [56]. Thus, it is necessary that services be designed to leverage geo-distributed deployments for reducing user-perceived latencies.

Locality-exhibiting workload: Real-world workloads served by these infrastructures tend to exhibit significant geographic and temporal locality [155, 38, 94, 61]. Exploiting locality of demand to reduce infrastructure cost as well as user-perceived latencies is another key focus in our service designs.

1.2 Degrees of freedom in infrastructure service design

Due to a geo-distributed infrastructure deployment, infrastructure services commonly make three sets of decisions: *content placement*, *request redirection*, and *network routing*. As each of these decisions can be made relatively independent of others, we call them "degrees of freedom" available to a service.

- **Content placement** selects the locations at which a content is placed. Its objective is to balance resource cost of keeping multiple content replicas with the benefits of improved availability and reduced user-perceived latency.
- **Request redirection** selects a location to send a request to, with a preference for selecting a location that is nearby and has sufficient resources to serve the request.

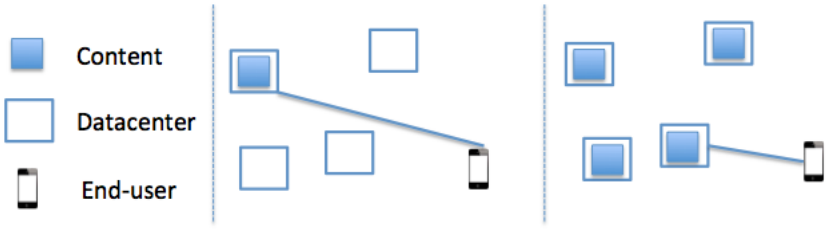


Figure 1.1. Placement vs. redirection: Content placement creates options for the redirection scheme to choose a nearby location for reducing user-perceived latencies.

- **Network routing** (or **traffic engineering**) selects the physical paths between nodes in the network based on topoology and traffic demand patterns. A key goal is to avoid congestion hotspots in the network.

1.3 Thesis statement

Content placement is a powerful factor, and is often more powerful than redirection and routing, in determining the cost, performance and energy-related metrics for infrastructures supporting a content-dominated, highly mobile Internet.

1.4 Why placement is more powerful than redirection, routing

Geo-distributed infrastructure and workload locality makes content placement a powerful degree of freedom. Workload locality implies that a content is typically popular in only a few regions at any given time, and as a result only a few replicas of a content placed in those regions are sufficient to reduce user-perceived latency for most users. Further, the geo-distributed infrastructure enables these few replicas to be placed close to regions of demand, thereby achieving a good tradeoff between user-perceived latency and cost of content replication. We illustrate with a couple of examples why placement can be a more powerful degree of freedom than redirection and routing in designing infrastructure services.

Figure 1.2. Placement vs. routing: content placement is more powerful than routing since it can change the traffic matrix itself.

Placement vs. redirection: Suppose a user requests a content that is placed only at a remote location as in Figure 1.1 (left). Irrespective of the redirection scheme, a user will observe high latencies to fetch content from the remote location. Placement is more powerful that redirection becasue it can create more options for the redirection scheme, enabling it to choose a location that is nearby as shown in Figure 1.1 (right). Thus, effective placement is a prerequisite for a redirection scheme to provide low latencies.

Placement vs. routing: Content placement is more powerful than routing because it can change the *traffic matrix* for which the routing is to be computed. A traffic matrix is a 2-dimensional matrix representing the demand in the network. The i, j-th entry in this matrix is the traffic from node i to node j in the network topology [68]. Let us take an example traffic matrix for which routing is to be computed (Figure 1.2). Content placement can turn any traffic matrix into a null matrix (one whose all entries are zero) provided all content that a node in a network needs is placed at the same node. Such a placement obviates sending traffic to any other node, and therefore makes routing a trivial problem. While this is an extreme example, and one may not have sufficient resources to place content at all locations in the network, it demonstrates that the ability to change the traffic matrix makes content placement more powerful than routing.

1.5 Research overview

We study the design of services for the following infrastructures in this thesis.

Internet service provider (ISP): ISPs perform traffic engineering to achieve several objectives such as cost and congestion reduction, fault tolerance etc. As we explain in Section 1.5.1, our work designs and evaluates traffic engineering in content-dominated ISP networks while accounting for its interaction with placement and redirection schemes.

Global name service (GNS): GNS can enable establishing and maintaining connections between mobile entities on the Internet. As we explain in Section 1.5.2, our name service design, Auspice, meets the challenge posed by high mobility, which is to return up to date addresses for billions of mobile names with frequently changing network address.

Content datacenter (CDC): Content datacenters are used to cache and serve content to end users. As we explain in Section 1.5.3, our work quantifies the tradeoff between user-perceived performance and energy savings achieved via a coordinated consolidation of servers and switches, and presents the design and implementation of a system to leverage this tradeoff.

1.5.1 Traffic engineering in content-dominated ISP networks

In an ISP network with content-dominated traffic, traffic engineering decisions that compute the network layer routing are not isolated from traffic adaptation occuring at the application layer. For example, content placement and request redirection determine the traffic matrix and hence influence traffic engineering. While the interaction between network routing and application adaptation has been studied previously [151, 140, 98, 58, 71, 178], a distinguishing aspect of our work it to study the role of placement schemes on this interaction.

We design and evaluate traffic engineering schemes in two scenarios that vary in terms of the flexibility of content placement. The first scenario is a more common occurence in the present day Internet, in which an ISP has little control over placement and redirection. The second scenario is motivated by a recent, potentialy transformative trend: Network CDNs (NCDNs) – CDNs deployed by ISPs on their infrastructures. Unlike traditional ISPs, NCDNs enjoy full control over placement, redirection and routing on their networks.

1.5.1.1 ISP network with content location diversity

We model a content-dominated ISP network by accounting for its *location diversity* – the presence of content at multiple network locations, and the ability of end users to download content from those locations. Location diversity is enabled by several types of applications and services, such as CDNs, P2P applications, mirrored websites. We create location diversity using a simple content placement scheme – randomly placing content at multiple locations –, to reflect the limited control of ISPs on content placement in their networks.

Our work presents an experimental comparison of several classes of traffic engineering schemes using data from real ISP topologies and traffic matrices. A key finding is that accounting for the application adaptation to location diversity increases the capacity achieved by several state-of-the-art traffic engineering schemes, and, surprisingly, blurs the capacity achieved by them. Here, capacity refers to the ability of a traffic engineering scheme to tolerate an increase in traffic demand. Even a static shortest-path routing, or in other words a "no" traffic engineering scheme is at most 30% sub-optimal in terms of capacity. Overall, these results suggest that even a limited placement flexibility reduces the value of carefully engineering routing. The value of traffic engineering is further reduced in an NCDN where the content placement flexibility is even greater.

1.5.1.2 Network CDNs

There are strong economic factors motivating ISPs to transform into NCDNs for delivering content to users on its network: need for additional revenue source in the face of falling bandwidth prices due to competition and technology trends, viability of monetizing NCDNs by selling content-based services to end users, easy availability of CDN technology in the form of licensed and managed CDNs and reduction in backbone traffic due to content caching via NCDNs [46]. These factors have motivated more than 30 ISPs to deploy NCDNs on their network.

NCDNs represent a paradigm shift in which both content delivery and traffic engineering is handled by a single entity. This change affects the metrics of interest of an NCDN as well the techniques it can use. While a traditional ISP is concerened with traffic enginerring related objectives such as network link utilization, and a traditional CDN is concered with optimizing user-perceived performance, an NCDN is concered with both these metrics. While a traditional ISP can decide network routing and a traditional CDN can decide placement and redirection, but only an NCDN can decide placement, redirection and routing. Besides using existing techniques to handle these tasks independently, an NCDN also has the option of optimizing these decisions jointly.

Our contribution is to evaluate existing schemes as well as a new joint optimization scheme for NCDNs based on real network topologies and extensive real world content access traces from Akamai CDN. The schemes we evaluate include (1) a simple caching scheme for placement and a static-shortest path routing, (2) a joint optimization based on historic demand patterns and (3) an ideal joint optimization with future knowledge of content demand. Of particular interest to NCDNs are demand-oblivious schemes such as the first scheme, as they make placement and routing decisions without measurement of content demand and simplify network management for the operators.

Our main finding is that the simple demand oblivious scheme is effective in improving network and latency cost for an NCDN: it performs signficantly better than the history-based joint optimization scheme and performs close to the ideal-joint optimization scheme. The history-based joint optimization scheme suffers due to poor placement for workloads with poor predictability of content demand and sigificant daily churn. The demand-oblivious scheme performs well largely due to its placement scheme, which achieves high cache hit rates from caches placed inside the network. We also show that the demand-oblivious scheme can gain only a small ($< 10\%$) cost reduction by using traffic engineering instead of a static-shortest path routing. Overall, our findings question the value of traditional traffic engineering for NCDNs and simplify the task of managing NCDNs.

1.5.2 Global name service for a highly mobile Internet

The Internet has a poor support for estabilishing and maintain connections between mobile entities. Today, it is difficult to initiate a connection with a mobile device since there is no global infrastructure to locate it. As a result, mobile communication initiation is mostly unidirectional in the Internet, from the mobile to the fixed hosts. The lack of this basic functionalily has led to application-specific solutions which causes both redundant effort by developers and possibly redundant infrastructure expenditure also. Yet, support for mobility is patchy among applications: SSH sessions terminate unexpectedly, HTTP downloads terminate before completion and VoIP calls get disrupted when network addresses change.

A key reason for the Internet's poor support for moblity is that communication on the Internet is based on IP addresses that keep changing due to mobility. On the other hand, what remains unchanged is the name or the identity of communication end-points. Thus a *name-based communication* paradigm is a promising solution to mobilty. A global name service(GNS) can enable such a name-based communica-

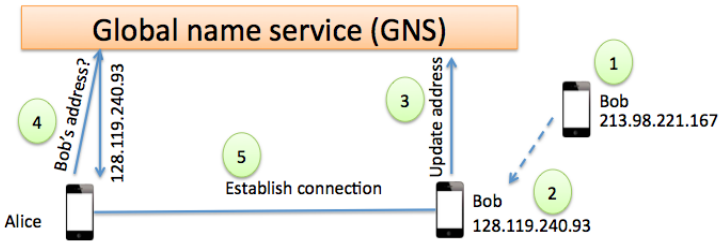


Figure 1.3. A global name service helps establish and maintain connections between mobile entities by keeping an up-to-date mapping from their names to their network addresses.

tion by maintaining an up-to-date mapping from names to network addresses for all names.

Figure 1.5.2 illustrates how a GNS helps establish connections in a common network mobility scenario. A mobile user Bob changes its network address and updates the GNS with its new address soon after. Afterwards, another mobile user Alice obtains Bob's new address from the GNS and is able to establish connection with Bob despite Bob's mobilty.

To appreciate the challenges in implementing such a GNS, we discuss the limitations of the Internet's existing naming service DNS as a solution for mobility.

Passive caching: DNS relies heavily on passive caching based on TTLs for reducing both system load and user-perceived latency. However, high mobility severely limits effectiveness of TTL-caching. Establishing connection under high mobility requires up-to-date knowledge of network addresses, that must be obtained from *authoritative name services* in DNS. So, the load and the user-perceived latency increase with the mobility rate irrespective of the TTL.

Static placement: Under high mobility, the latency to an authoritative name service determines user-perceived request latency in the common case. Today, authoritative name service locations are chosen statically irrespective

of where the request demand is coming from, which could result in highly sub-optimal request latencies for users requesting the name service from a location distant from the statically placed replicas.

Hierarchical names: DNS uses a hierarchical design of namespace as well as federation structure. Due to DNS's hierarchical design, the main proposal to enhance DNS's security, DNSSEC, is dependent on a single root of trust.

While the first two problems are related to authoritative name services and can be addressed within the scope of current DNS, the third issue requires a clean slate redesign of the naming system. Below, we discuss our naming system and authoritative name service designs.

Global naming system: Our global naming system improves upon DNS's design in two aspects. First, it supports multiple roots of trust as a part of the naming system, thereby addressing the single root of trust problem in DNS. Second, it supports name-to-address resolution for arbitrary names unlike DNS, which restricts names to be hierarchical. In doing so, our naming system gives applications the full flexibility in choosing names.

Auspice: Our name resolution service, Auspice, resolves names to their network addresses, both represented in any format, under high mobility. The flexibility of name and address formats makes Auspice deployable in the Internet today as a scalable authoritative name service, potentially enhacing support for mobility in the present day Internet as well.

A key distinction betweeen Auspice and existing name services is Auspice's *demand-aware placement* of name records. Name records store name-to-address mapping as well other attributes. Auspice's demand-aware placement heuristic adapts the set of replicas for a name record based on read-to-write ratios, overall

system load and the geo-distribution of demand for a name record to provide low latency name-to-address lookups in a cost-effective manner. Auspice' placement enables it to significantly outperform commercial managed DNS services, DHT-based name service designs as well as static placement schemes such as random-k and replicate-all. This finding demonstrates the importance of placement in Auspice's design.

1.5.3 Energy optimization in content datacenters

In today's content-dominated Internet, it is not surprising that many datacenters and Points-of-Presence (PoPs) are dedicated to storing and serving content to end users. We call these datacenters and PoPs *content datacenters* (or, CDCs). The three degrees of freedom – placement, redirection and routing – discussed in a wide-area setting previously, are also available inside a CDC: content placement to select which servers to store on content on, request redirection to select which server inside a CDC to send a request to, and network routing on CDC topology.

Energy use is a key concern for CDC operator managing a large global network of CDC's, which can comprise of 100K servers or even more [16, 141]. A potential approach to reduce energy use is *consolidation* in a CDC. Consolidation of servers can reduce server energy use by using only a fraction of CDC's servers for placement and redirection and turns remaining servers off [118]. Similarly, consolidation of network can reduce network energy use by routing network traffic via only a fraction of switches and links and turns remaining switches and links off [166]. A key concern for operators seeking to deploy these techniques is to understand the tradeoff between energy use and performance, i.e., do energy saving benefits outweigh the user-perceived performance penalty incurred?

Our work presents the first quantitative analysis of the energy-performance tradeoff for a CDC based on a real CDC workload and present the design and

implementation of a system, Shrink, to leverage this tradeoff. A key insight, supported via experiments, is that cache hit rates, despite server consolidation, remain close to an unconsolidated datacenter. A small reduction in hit rate helps mitigate the impact of consolidation on user-perceived latency. This finding is explained by the skewed content popularity obseved in real workloads, due to which working set size of content remains small compared to the storage available in a CDC. Our quantitative analysis shows that Shrink reduces energy use by 35% over a baseline scheme that keeps entire datacenter always on while increasing the mean, 95-th %-ile and 99-th %-ile latencies by 8%, 3% and 15% respectively.

Further, we show that a coordinated approach for server and network consolidation reduces *network* energy use more than otherwise possible. In comparison, previous work has studied server and network consolidation in isolation. We consider a simple *network-aware server consolidation* scheme that selects the active servers in a left-to-right order in a topology. The same network consolidation scheme results in up to 42% less network energy use when used with our network-aware server consolidation instead of a network-unaware server consolidation scheme that selects servers randomly.

Our results show that placement affects both server and network energy use in a CDC. Due to an effective placement, cache hit rates remain high despite consolidation, which helps reduce server energy use with a modest performance impact. Further, our network-aware server consolidation is able to significantly reduce network energy use over a network-unaware server consolidation by coordinating the placement and redirection with the routing in a CDC. These results further support our statement that content placement is of key importance in a content-dominated Internet.

1.6 Thesis organization

The thesis comprises of three parts that correspond to our three research topics.

The first part, which includes Chapter 2, Chapter 3 and Chapter 4, presents our research on traffic engineering in content-dominated ISP networks. Chapter 2 presents background on traffic engineering, content delivery and the interaction between traffic engineering and application-layer adaptation, and motivates the key research questions we address on this topic. Chapter 3 presents a comparison of traffic engineering schemes in a network with location diversity of content. Chapter 4 designs and evaluates traffic engineering and content delivery schemes in a Network CDN.

The second part, which includes Chapter 5, presents the design, implementation and evaluation of Auspice – a global name service for a highly mobile Internet.

The second part, which includes Chapter 6, presents the design, implementaiton and evaluation of Shrink - a system for reducing energy use of content datacenters via a coordinated consolidation of servers and switches.

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 5 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 proposal, 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. Aditya Yadav has developed the msocket library used in performing experiments with Auspice. Further, Hardeep Uppal, David Westbrook and Arun Venkataramani are contributors in Auspice's implementation.

CHAPTER 2

TRAFFIC ENGINEERING IN CONTENT-DOMINATED ISP NETWORKS: BACKGROUND AND MOTIVATION

This chapter serves two main purposes. First is to provide the reader with necessary background on ISP traffic engineering, content delivery and the interaction between traffic engineering and application-layer traffic adaptation (Section 2.1). Second is to explain the motivation for our work on traffic engineering in content-dominated ISP networks in light of the existing work in this area (Section 2.2).

2.1 Background

Our review of prior work provides following main findings:

- ISP traffic engineering schemes compute routing for optimizing link-utilization based cost functions. These schemes commonly take a demand-aware approach that uses previously measured traffic matrics for computing future network routes (Section 2.1.1).

- CDNs commonly use *demand-oblivious* content placement and request redirection techniques towards improving user-perceived performance across the Internet (Section 2.1.2). These techiques requires do not require content-level measurement of demand but instead use simple online heuristics to make their decision.

- Prior work on the interaction between traffic engineering and application-layer adaptation focuses primarily on two aspects of application-layer adaptation – overlay routing and request redirection. These interactions result in

globally sub-optimal network cost as well as user-perceived latency, and co-operative mechanisms can leverage these interactions to improve cost and performance metrics.

2.1.1 Traffic engineering

A key goal of ISP traffic engineering is to avoid congestion hotspots in the network by optimizing routes based on network topology and expected traffic demand that is represented in the form of traffic matrix. In ISP networks, traffic engineering decides both intra-domain routing (within the ISP) and inter-domain routing (across ISPs). We focus here on intra-domain routing and refer the reader to [64, 149] for a survey of inter-domain traffic engineering.

The evaluation metric for ISP traffic engineering is a cost function that is dependent on the utilization of network links. A well-known cost function is maximum link utilization or MLU [149, 171]. A low link utilization is desirable to ISPs for two reasons. First, a low utilization of all links implies that the network is free from congestion hotspots. Second, it also implies that a network has more spare capacity to tolerate an increase in demand. Consider MLU as a capacity metric for example. If a traffic engineering 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.

We give an example to show how traffic engineering reduces link utilization. In Figure 2.1, there are two links of capacity 1 Mbps and 3 Mbps between nodes A and B. The traffic from A to B is 2 Mbps. If a shortest path routing is followed, all traffic must be sent through either of the two links. The least MLU = 0.67 is achieved by using the large capacity link. A more flexible routing approach is to split traffic among the two links. Such a flow-split routing achieves MLU = 0.5 by sending 0.5 Mbps and 1.5 Mbps via the two links respectively. Thus, a better engineered routing resulted in a lower MLU in this example.

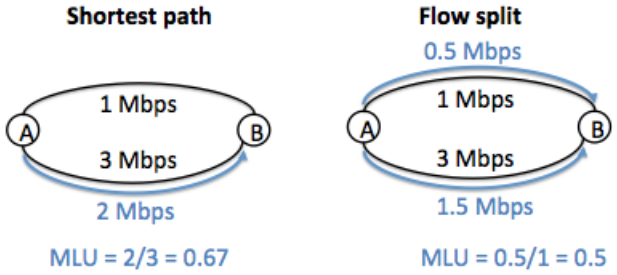


Figure 2.1. A flow-split routing reduces the maximum link utilization (MLU) over shortest-path routing.

If the traffic matrix is known accurately, the optimal solution to the traffic engineering problem can formulated as a multi-commodity flow optimization. The routing thus computed is refered to as *optimal traffic engineering* in literature [68, 101]. As it is impossible to have accurate knowledge of future traffic matrices, traffic engineering schemes either take a *demand-aware* or a *demand-oblivious* approach. A demand-aware traffic engineering periodically updates routing using historically observed traffic matrices [68, 67, 171, 179]. A demand-oblivious traffic engineering requires no explicit measurement of traffic matices, but instead configures routing statically. A common demand-oblivious technique is to use shortest path routing in which link weights are set to inverse of link capacities [69], henceforth referred to as InvCap in this thesis. A demand-oblivious traffic engineering is simple to implement because it does periodic traffic matrix measuremnt and routing configuration updates.

In practice, demand-aware traffic engineering based on Open Shortest Path First (OSPF) and Multiprotocol Label Switching (MPLS) are commonly used [171, 179, 68, 60]. 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 as exemplified above [171, 179].

2.1.2 Content delivery

Content delivery systems seek to improve user-perceived performance for content accesses 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 request redirection, and path and protocol optimizations for delivery of several types of content, e.g., video, bulk downloads, and interactive websites [57, 127]. Given their geo-distributed deployment, the decisions of content placement, i.e., locations at which acontent 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.

Content placement: Content placement in CDNs is commonly done using caching schemes. For example a commonly used caching scheme is least recently used (LRU) cache replacement. There are two reason why content cahing is widely used. First, caching naturally captures geographic and temporal locality in content requests to populate caches with content likely to be reused. Second, a vast majority of network traffic is generated by content that gets updated infrequently, e.g., a video, audio, images. As a result, cached copies of content remain reusable for long duration.

Request redirection: In a CDN, request rediection occurs at two tiers - inter-datacenter and intra-datacenter. Our focus in this part of the thesis is on inter-datacenter request redirection because it influences wide-area traffic patterns that affect traffic engineering. We discuss intra datacenter redirection in the context of a content datcenter (Chapter 6).

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 has sufficient resources to serve a request [57].

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 [73, 173]. (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 [173]. (3) *Informing the user:* The user is informed of selected server/s either via DNS resolution [57] or via HTTP redirection [30].

Content delivery techniques can be classified into demand-aware and demand-oblivious similar to our classfication of traffic engineering schemes. We consider common content delivery techniques discussed above to be demand-oblivious since they do not require long term content-level measurement of demand but instead use simple online heuristics to make their decision.. In contrast, a demand-aware approach to placement and redirection has also been studied, e.g., Applegate *et al.* [26] use a demand-aware approach to determine placement and redirection for Video-on-Demand content in the network.

2.1.3 Interaction between traffic engineering and application-layer adaptation

Studying the interaction between network-layer routing computed by traffic engineering and application-layer adaptation has been a topic of long standing interest in computer science. Several related questions have been put forth. Does

this interaction yield globally sub-optimal results? How can we design cooperative mechanisms to leverage these interactions? Do cooperative mechanisms yield benefit in an Internet-like environment? Most previous research on this topic has focused on the interaction of traffic engineering with either overlay routing [151, 140] or with request redirection [98, 58, 71, 178] as discussed below.

Interaction between traffic engineering and overlay routing: Several results show the negative interaction between selfish overlay routing and network routing [151, 140]. Theoretical results indicate that the negative interaction could cause an arbitrary degradation in user perceived delay. Further studies using Internet-like traffic demands and topologies indicate that this interaction hurts traffic engineering metrics. While this body of work focuses on application adaptation to *path diversity* created by overlay routing, our work focuses on *location diversity* present in the Internet (Chapter 3). We do not consider overlay routing since our focus is primarily on intra-domain routing while overlay routing is most commonly used to mask inter-domain routing inefficiencies [152].

Interaction between traffic engineering and request redirection: Recent work has studied the interaction between ISP traffic engineering and request redirection by content providers with geo-distribtued datacenters as well as P2P applications. Both analytical results [98, 58] and system implementations [71, 178] have shown that there is value for joint optimization of request redirection and traffic engineering, and cooperative strategies can help traffic engineering metrics while maintaining or improving user-perceived performance. Our work distinguishes in that we model the three-way interaction between placement, redirection and routing and show that placement flexibility is more powerful degree of freedom than redirection or routing to improve traffic engineering metrics and to reduce user-perceived latency (Chapter 4).

2.2 Motivation

A content-dominated traffic and the presence of content at multiple network locations changes the traditional ISP traffic enginering problem. In such a network, a traffic matrix is no longer an effective representation of demand. It can be changed via application-layer adaptation without any underlying change in demand. Demand is better expressed as a *content matrix* in which each entry represents the demand for a content at a particular network location. A traffic matrix implicitly assumes a fixed source and a fixed destination for each traffic flow. But, a content matrix expresses the fact that the demand for a content at a particular destination can be served from any set of source locations. As we explain using examples below, the flexibility of serving the demand from any location raises questions on the usefulness of link-utilization based metrics that are commonly used in traffic engineering (Section 2.2.1.2) as well on the importance of traffic engineering itself (Example 2.2.2.1). These and other related questions motivate our work on traffic-engineering in content-dominated networks.

Our work presents research on two network scenarios that vary in terms of application adaptation mechanism and the content placement flexibility in the network. We explain the two scenarios and motivate the research questions we address in each case using examples.

2.2.1 ISP network with content location diversity

We consider an ISP network that has little control over placement and redirection, but applications can leverage *location diversity*, or the ability to download content from multiple locatioons. We motivate why traffic engineering schemes must be evaluated while accounting for their interaction with location diversity, and why we need to look beyond link-utilization based metrics in evaluating traffic engineering schemes. These questions are addressed in Chapter 3.

2.2.1.1 Interaction between traffic engineering and location diversity

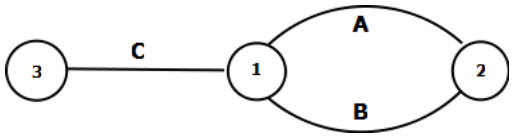


Figure 2.2. Lasso network

This example illustrates how location diversity can change the traffic matrix without any underlying change in demand. In Figure 2.2.1.1, 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. This example illustrates that the outcome of traffic engineering depends on the application adaptation to location diversity, thereby motivating us to study the following question.

Research question: How do traffic engineering schemes compare while accounting for the effect of location diversity in the network?

2.2.1.2 Shortcomings of link-utilization metrics

Link-utilization based metrics may not be a good predictor of user-perceived performance that depends on other factors such as propagtion delay, access link capacity and backbone link capacity also. Moreover, queuing-theoretic models show that only a high link utilization severely affects performance-critical metrics such as queuing delay and packet loss [102]. For low to moderate link utilization that is common in today's ISPs, a reduction in link utilization may not yield a commensurate benefit in application peformance.

Research question: Are link-utilization based metrics good predictors of user-perceived performance for real network topologies and traffic matrices?

Link-utilization based metrics, in particular inverse of MLU, are a poor capacity metric in the presence of location diversity. In using inverse of MLU as a capacity,

we implicitly assume that traffic matrices scale linearly as network traffic increases. But, this assumption may not hold if in a network where application adaptation to location diversity determines the traffic matrix as in the above example. Thus, location diversity necessitates a new capacity metric.

Research question: How do we quantify network capacity under location diversity?

2.2.2 Network CDN

Network CDNs are a recent, potentialy transformative trend in which ISPs deploy CDNs on their infrastructures to deliver content to users on their network. Unlike traditional ISPs and traditional CDNs, NCDNs enjoy full control over placement, redirection and routing on their networks. The interaction between placement and routing in an NCDN illustrated below motivates us to explore the benefit of joint optimization of placement, redirection and routing as well as to evaluate the relative importance of placement and routing in an NCDN. These questions are addressed in Chapter 4.

2.2.2.1 Interaction between placement and routing in an NCDN

To appreciate how placement can shape traffic in an NCDN, consider the simple example in Figure 2.3. Node C has an object in its cache that is requested by end-users at nodes A and D. Suppose that one unit of traffic needs to be routed from C to A and 0.5 units from C to D to satisfy the demand for that object. The routing that achieves the minimum MLU of 0.5 to serve the demanded object is shown in the figure. Note that the routing that achieves the MLU of 0.5 is not possible with a simple, demand-oblivious scheme like InvCap as that would route all the traffic demand from C to A via B, resulting in an MLU of 1. Thus, a (demand-aware) traffic engineering scheme is necessary to achieve an MLU of 0.5.

On the other hand, NCDNs can shape the traffic demand matrix by using a judicious placement and redirection scheme. Suppose that there is some space left

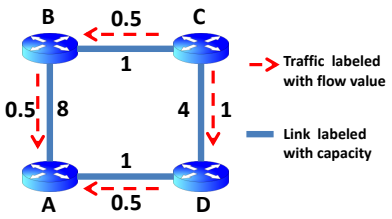


Figure 2.3. A simple NCDN example

in the content server's cache at node B to accommodate an additional copy of the demanded object. By creating an additional copy of the object at B, the traffic demand of A can be satisfied from B and the demand of D from C achieving an MLU of 0.125. In this case, judicious content placement decreased the MLU by a factor of 4. Even more interestingly, this best MLU can be achieved using a simple routing scheme like InvCap routing while also improving user-perceived latency (assuming that the latency of link BA is lower than that of the two-hop paths from C to A). While this is a toy example, it shows that content placement flexibility reduces network cost and enables simpler routing. This interaction between placement and routing motivates us address the following questions for real content workloads and network topologies.

Research question: Does a joint optimization of placement, redirection and routing yield benefits over independently making these decisions using simple demand-oblivious schemes in an NCDN?

Research question: Does content placement flexibility obviate sophisticated traffic engineering in an NCDN?

CHAPTER 3

AN APPLICATION-CENTRIC COMPARISON OF ISP TRAFFIC ENGINEERING SCHEMES

Traffic engineering (TE) techniques are used by Internet Service Providers (ISPs) for computing network routing based on expected traffic demands [149]. Our work focus on two aspects of ISP traffic engineering that have received less attention in prior work and are of key importance in a content-dominated network. The first is the impact of traffic engineering on user-perceived peformance such as TCP file download times. The second is the impact of application adaptation on traffic engineering. In particular we focus on a form of adaptation enabled by *location diversity*, or the ability of applications to download content from multiple locations. Location diversity is prevalent in the Internet due to several applications and services such as content delivery networks, peer-to-peer applications and mirrored websites [127, 48].

While ISP traffic engineering schemes are commonly evaluated using link-utilization based metrics, these metrics are ill-suited to compare schemes based on user-perceived performance or network capacity, i.e., the factor of surge in traffic demand that a network can tolerate, under location diversity. As discussed previously in Chapter 2, link utilization-based metrics may not reflect application performance that depends on other metrics such as propagtion delay, access link capacity and backbone link capacity also. Further, link utilization based-metrics, in particular MLU, do not reflect network capacity under location diversity.

There are two key parts of our approach to evaluate a TE scheme. First, we carefully and at scale simulate network traffic as a collection of TCP flows. This empir-

ical approach enables us to accurately measure application performance metrics — TCP throughput for elastic traffic and a quality-of-service metric, MOS score for VoIP quality [49], for inelastic traffic — and model application adaptation to location diversity, e.g., how an application splits traffic among multiple content locations. Second, we propose a new capacity metric called Surge Protection Factor (SPF). Unlike MLU, SPF captures the factor of increase in demand that can be sustained while accounting for location diversity.

We show following results from comparing user-perceived performance of TE schemes based on real network topologies and traffic demands. All state-of-the-art TE schemes achieve nearly identical application performance at typical Internet load levels. In fact, even a demand-oblivious TE scheme — 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, demand-aware TE schemes that engineer for unexpected traffic spikes (e.g., COPE [171]) consistently hurt TCP throughput despite achieving near-optimal MLU. Overall, these results show that link utilization based metrics are poorly correlated with user-perceived performance.

The explanation for the above results is that link loss rates and queuing delays remain negligibly small at typical Internet loads, and in fact until the utilization starts approaching the capacity. This observation above also been confirmed by explicit measurements on Internet backbones [32], 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 [90] and 90% of TCP flows experience no packet loss [70]. Furthermore, end-to-end Internet path delays are known to be largely determined by propagation delays as opposed to queueing delays [70, 130].

Our comparison of the SPF of TE schemes in which application leverage location diversity by downloading content in parallel from all locations shows following somewhat surprising results. A small degree of location diversity (2-4 locations) increases SPF achieved by TE schemes by up to 2×. But, the capacity increase benefits sub-optimal TE schemes, e.g., demand-aware TE via OSPF link weight optimization [68], much more than the optimal TE scheme. As a result, all demand-aware TE schemes end-up achieving the same SPF as the optimal TE scheme. Even the demand-oblivious scheme, InvCap, achieves an SPF at most 30% worse than optimal TE.

We also experiment with a second adaptation scheme in which users download content from a single location with the smallest propagation delay. In these experiments, an increase in location diversity yields little improvement in SPF of any TE scheme. This adaptation reduces the SPF of Optimal to bridge the gap between Optimal and sub-optimal schemes such as OptWt. Overall, these results show that application adaptation to location diversity signficantly reduces the differences between TE schemes.

The rest of the chapter is organized as follows. Section 3.1 introduces location diversity and proposes a new capacity metric SPF to compare traffic engineering schemes. 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

This section introduces location diversity and proposes 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 [109], EdgeCast [37] 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 [162]. Popular P2P applications such as BitTorrent [47], PPLive [114] 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 [36], Rapidshare [22], 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 [148]. 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 [161, 148]. 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: 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, MLU is not a meaningful metric of capacity when application adaptation to location diversity determines the traffic matrix (Chapter 2, Section 2.2.1.1).

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. 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 in Chapter 2, Section 2.2.1.1 shows, even if the demand is unchanged, the mere act of engineering routes can change the traffic matrix yielding a different MLU than expected.

3.1.2.1 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 in-

crease 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.

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

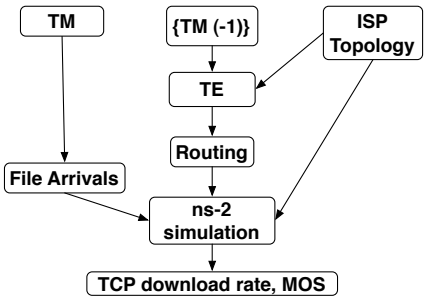


Figure 3.1. Block diagram of experiment process

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

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

Figure 3.2. 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 3.3. Bandwidth Distribution

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.

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 [164]. (2) **Geant**, the un-anonymized version of the Geant topology obtained from the TotemData [139] project personnel. (3) **US-ISP**, a large Tier-1 ISP topology obtained from authors of [179]. TMs for all ISPs were logged in the period from 2004-2005. Figure 3.2 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" [128] released by Akamai, one of the largest commercial content distribution networks in operation today. Figure 3.3 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 [161, 85, 78, 177].

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 [136] 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 including optimal TE (Optimal), demand-oblivious TE (InvCap) and demand-aware TE (OptWt, MPLS and COPE).

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 [129].

OptWt, a shortest-path routing algorithm using link weights computed using a heuristic algorithm to optimize a cost function [68]. We use its implementation in the Totem Toolbox [139]. 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 [171].

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 user-perceived 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 user-perceived performance. Like Optimal, MinDelay has perfect knowledge of traffic matrix.

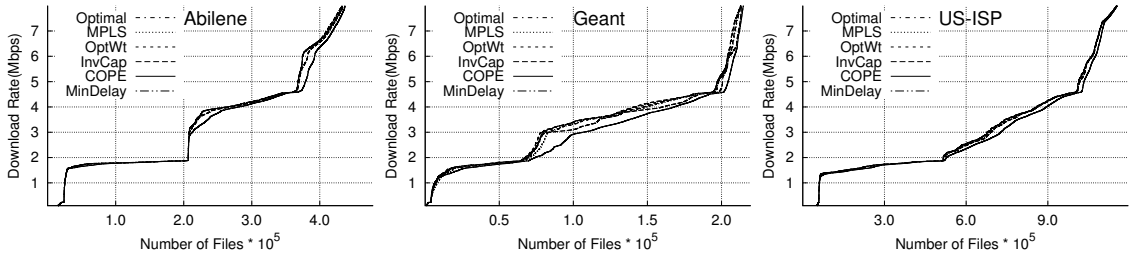


Figure 3.4. Download rate CDFs for all TE schemes are near identical except COPE which has slightly lower performance

3.3 User-perceived performance

In this section, we present a comparative analysis of the impact of different TE schemes on user-perceived performance. A summary of our findings is as follows. First, all TE schemes including InvCap show nearly identical user-perceived performance for TCP and UDP traffic. Second, different TE schemes do achieve different MLUs as expected, suggesting that MLU is a poor predictor of user-perceived 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 user-perceived 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 user-perceived 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 3.5 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 3.4 shows the corresponding CDFs for the mean download rates in Figure 3.5. 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 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 3.5 and Figure 3.4, we analyze the empirically observed MLU for all TE schemes in the experiments. Figure 3.7 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

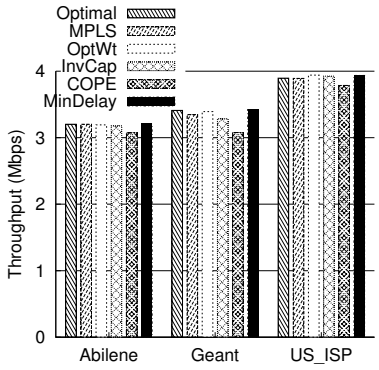


Figure 3.5. Mean download rates

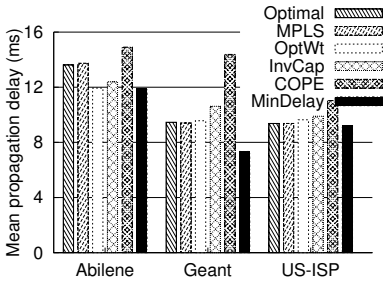


Figure 3.6. COPE has the highest propagation delay among TE schemes

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 [32] 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.

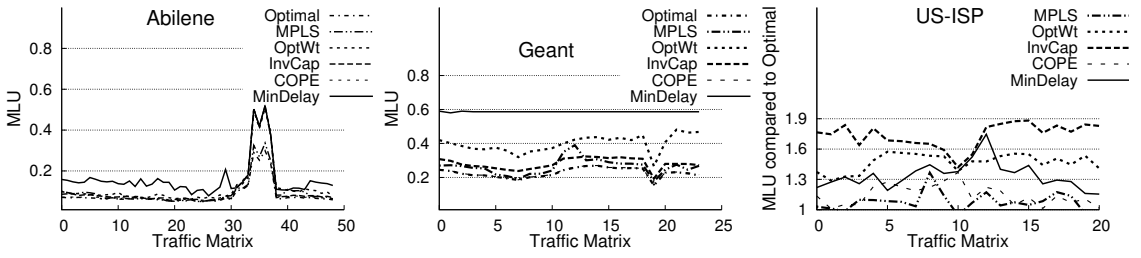


Figure 3.7. TE schemes differ as much as 2× in MLU

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 3.6 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 [25]. 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 user-perceived 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., [25], is likely to incur a similar penalty in user-perceived 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 [70]. 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 [49] 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 user-perceived 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 3.1.2.1). 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 3.8, 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 3.9 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.

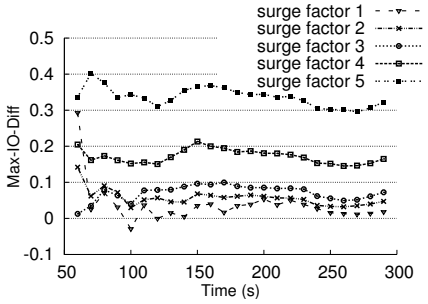


Figure 3.8. Profile of Max-IO-Diff at increasing surge factors for a Geant TM

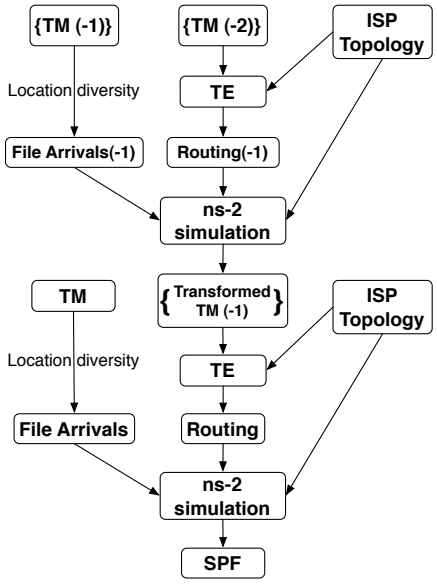


Figure 3.9. Block diagram of experiment process with location diversity

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. 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.

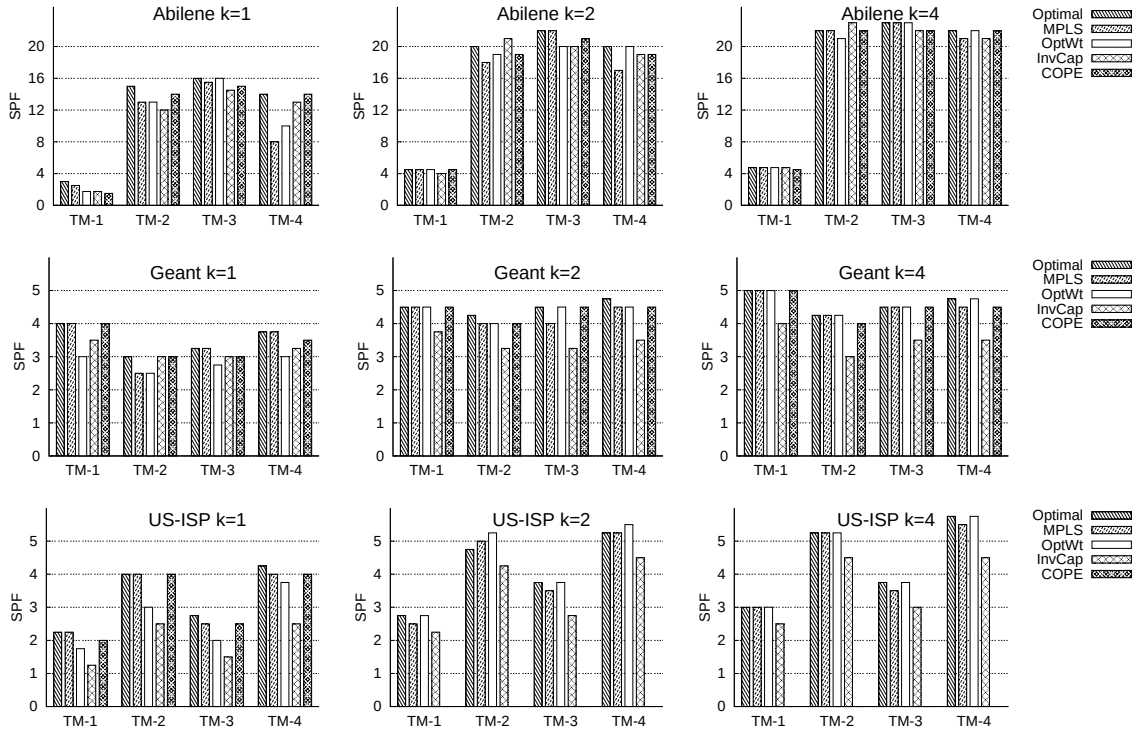


Figure 3.10. Comparison of SPF among TE schemes for different levels of location diversity; SPF values are obtained using ns-2 simulations

3.4.4 Capacity increase with location diversity

In Figure 3.10, we present the SPF values obtained using ns-2 simulations for the selected TMs. We compared all TE schemes for three levels of location diversity: $k = 1, 2$ 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.

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 3.11. Comparison of SPF values

The average capacity increase for Optimal from $k = 1$ to $k = 4$ is 1.41$\times$ and from $k = 1$ to $k = 2$ is 1.31$\times$. 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 3.11 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 3.10 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 3.10, 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 [12]. 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.

3.4.5 Experiments with least latency adaptation

In experiments until now, users leverage location diversity by downloading a file in parallel from all locations. Next, we present our experiment with least latency adaptation in which a user downloads a file from the closest location it is available. This adaptation is inspired by redirection schemes that are widely used by CDNs today [55, 173]. These experiments reinforce our earlier finding that all TE schemes achieve nearly the same SPF values when applications adapt to location diversity. With least latency adaptation, an increase in location diversity yields little improvement in SPF of any TE scheme. This adaptation reduces the SPF of Optimal to bridge the gap between Optimal and sub-optimal schemes such as OptWt.

3.4.5.1 Experiment procedure

We model least latency adaptation as follows: a user downloads a file from the location that has the smallest propagation delay. If there are multiple paths between two locations, propagation delay is measured as the weighted average of latencies of all paths; the weight of a path is equal to the fraction of traffic it carries. To emulate a location diversity of k, each file is replicated at the original location based on traffic matrix and at $(k-1)$ additional locations. We select additional locations such that the probability of replicating a file at a location is proportional to its total outgoing link capacity.

We use flow-level simulations instead of packet level simulations for this set of experiments. Flow-level simulations are two orders of magnitude faster than packet-level simulations (few hundred seconds vs. few hours). Moreover, flow-level simulations suffice to measure SPF in this experiment. Our earlier experiments with parallel downloads require packet-level simulations to measure PoP-to-PoP traffic, and hence SPF. In case of parallel downloads, a user downloads fractions of a file from multiple locations. The fractions of a file downloaded from all locations can only be estimated if TCP throughput of every connection is known. We use packet-level simulations to accurately estimate TCP throughput in the earlier experiment.

To calculate SPF, flow level simulations measure PoP-to-PoP traffic and link utilizations. PoP-to-PoP traffic (in bits/sec) is equal the sum of sizes of files requested at the sink PoP from another PoP divided by the experiment duration. For a pair of PoPs, the PoP-to-PoP traffic crossing a link is equal to the fraction of flows between the PoPs that cross this link times the PoP-to-PoP traffic. Summing the PoP-to-PoP traffic that crosses a link for all pairs of PoPs gives the total traffic on a link, which yields link utilization. We increase the surge factor for a TM in small increments as before. The least surge factor at which any link utilization exceeds 1 is the SPF.

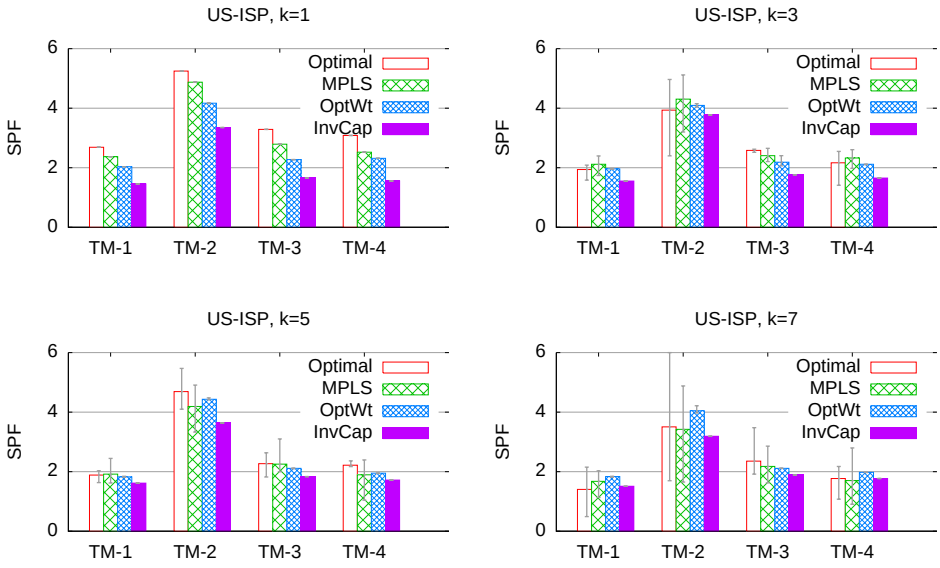


Figure 3.12. [US-ISP] Mean SPF values of TE schemes with least latency adaptation. Error bars show the maximum and minimum SPF values over 20 repetitions. At higher location diversity, SPF values do not increase, in some cases even reduce, as location diversity increases from k = 1 to k = 7.

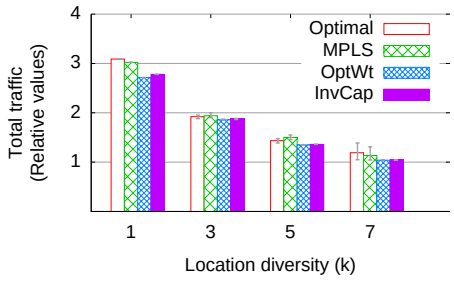


Figure 3.13. [US-ISP, Traffic matrix TM-1, surge factor = 1] Due to least latency adaptation, total traffic reduces to one-third as location diversity increases from k = 1 to k = 7. Despite reduction in total traffic, SPF does not improve.

3.4.5.2 Results

We discuss here the results for the Tier-1 US ISP network. The Abilene and Geant topologies show qualitatively similar conclusions. Figure 3.12 shows the SPF values for four traffic matrices (same TMs as in Figure 3.10). Location diversity increases from the top graph (k = 1) to the bottom graph (k = 7). In a network with location diversity (k = 3, 5, 7), all schemes including Optimal have nearly same SPF values. In comparison, Optimal has higher SPF than other schemes in

the absence of location diversity (k = 1). Even a static routing scheme, InvCap is at most 20% worse compared to Optimal. In our experiments with the Abilene and Geant topologies, InvCap is at most 30% worse compared to Optimal. These observations are consistent with our findings in earlier set of experiments with parallel downloads.

Contrary to expectation, increasing location diversity yields little improvement in SPF of any TE scheme. In Figure 3.12, the bars for each TM in the top graph (k = 1) are nearly as tall as the bars in the bottom graph (k = 7). In some cases, bars in the bottom graph are slightly shorter, that is, SPF worsens on increasing location diversity, e.g., TM-2 for Optimal.

With more location diversity, the utilization of most links reduces dramatically. This is evident from Figure 15, which shows the total traffic on all links at different levels of location diversity for a US-ISP traffic matrix. The total traffic on all links at k = 7 is one-third of the total traffic at k = 1 for every scheme. But the peak link utilization does not reduce, it even increases in some cases, as location diversity increases. This is why SPF does not improve despite a reduction in total traffic.

Contrary to expectation, SPF yields no improvement with increasing location diversity. In Figure 3.12, the bars for each TM in the top graph (k = 1) are nearly as tall as the bars in the bottom graph (k = 7). In some cases, bars in the bottom graph are slightly shorter, that is, SPF worsens on increasing location diversity, e.g., TM-2 for Optimal.

Figure 15 shows the total traffic on all links at different levels of location diversity for a US-ISP traffic matrix. In this graph, the surge factor is always equal to 1, so the aggregate demand at end-nodes remains the same. But, the total traffic on all links at k = 7 is one-third of the total traffic at k = 1 for every scheme.

With more location diversity, users, on average, download files from a location closer than before. This reduces the total traffic on all links, and reduces utilization

of most links. But the maximum link utilization does not reduce, it even increases in some cases, as location diversity increases. The most utilized link is the first to experience congestion at higher surge factors. Least latency adaptation does not respond to congestion by moving traffic from the most utilized link to other under-utilized links. It always downloads from the least propagation delay location, irrespective of the congestion on the path from that location. This is why SPF does not improve despite a reduction in total traffic. An implication of this finding is that an adaptation scheme must be responsive to network congestion to leverage location diversity and increase the network's tolerance to traffic demand surges.

Least latency adaptation does not respond to congestion by moving traffic from the most utilized link to other under-utilized links. It always downloads from the least propagation delay location, irrespective of the congestion on the path from that location. An implication of this finding is that an adaptation scheme must be responsive to network congestion to leverage location diversity and increase the network's tolerance to traffic demand surges.

3.5 Conclusion

Our comparison of TE schemes based on user-perceived metrics while accounting for application adaption to location diversity reveals 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 user-perceived performance. Under typical Internet load conditions, all TE schemes and even demand-oblivious static routing achieve nearly identical user-perceived 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 mul-

tiple locations, eliminates differences in the achieved capacity of all TE schemes including "optimal" TE. With location diversity, even demand-oblivious 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 at today's traffic levels as well as under reasonable projections of increased demand.

CHAPTER 4

NETWORK CDNS

The traditional tri-partite view of content delivery consists of three sets of entities: content providers (e.g., media companies, news channels, e-commerce sites, software distributors, enterprise portals, etc.), networks (e.g., telcos such as AT&T, multi-system operators such as Comcast, and ISPs), and content delivery networks (CDN) (e.g., Akamai, Limelight).

Recent powerful trends are reshaping the simplified tripartite view of content delivery. A primary driver is the torrid growth of video [124, 45] 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 [92, 119, 110, 28, 168] (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 subscribers (e.g., Verizon's recent offering that delivers HBO's content to FIOS subscribers [167].

[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[37] licenses the CDN software to a network to create an NCDN.

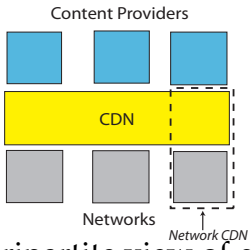


Figure 4.1. A tripartite view of content delivery.

As NCDNs control both the content delivery 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 as illustrated using the example in Chapter 2, Section 2.2.2. 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 [178, 58, 98, 72]. 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 delivery 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 demand-aware strategies (i.e., using knowledge of recently observed demand patterns or hints about anticipated future demands) for placement and routing compare against simpler, demand-oblivious 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 demand-oblivious 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 demand-oblivious routing (InvCap).

- 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 engineer-

[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)

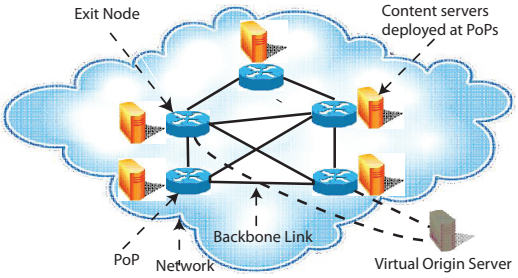


Figure 4.2. NCDN Architecture

ing 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

4.1.1 NCDN architecture

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.

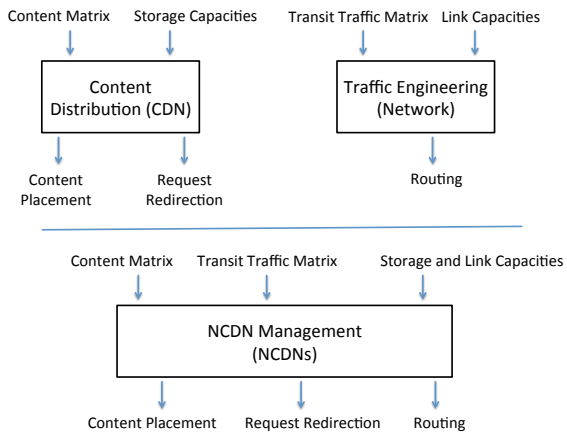


Figure 4.3. (Top) Traditional formulation with content delivery and traffic engineering optimized separately. (Bottom) Our new formulation of NCDN magamenent as a joint optimization.

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.

4.1.2 NCDN management objectives and schemes

Managing content delivery 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 delivery 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.

NCDN management refers to the combined task content delivery and traffic engineering performed by an NCDN. We classify the possible NCDN mangement schemes into two axes as discussed below.

Demand-aware vs. demand-oblivious: We summarize pros and cons of demand-aware and demand-oblivious approaches for content delivery and ISP traffic engineering discussed in Chapter 2. A demand-aware approach for ISP traffic engineering, which periodically computes and updates routing based on traffic matrices, has been shown to be superior to a demand-oblivous approach that configures routes statically. A demand-oblivious approach is more common for content delivery, in which content placement and request redirection is done using a simple online algorithm such as LRU cache replacement for content placement. In comparison, a demand-aware approach is more complex as it requires network-wide measurement of content-level demand and is potentially computationally expensive, but it could yield benefits for some workloads [24].

Independent vs. joint optimization: Unlike a traditional ISP or a traditional NCDN, an NCDN controls placement, redirection and routing on its network. An NCDN may simply treat placement, redirection and routing independently, or it may seek to jointly optimize these decisions to leverage the interaction between them. The example in Chapter 2, Section 2.2.2 illustrates the interaction between placement and routing in an NCDN. A joint-optimization approach may potentially yield benefits, as has been demonstrated in a different scenario where two distint entities ISPs and content providers jointly optimize their network routing and request redirection. Developing a joint optimization for NCDN and evaluating its benefits over independent optimization of placement, redirection and routing are among the key goals of our work.

4.2 NCDN joint optimization

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

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 $j \in V \cup \{o\}$

Figure 4.4. List of input and decision variables for the NCDN problem formulation.

hardness and inapproximability results, and discuss approximation heuristics to solve MIPs for realistic problem sizes.

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 [68]. 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 [68]).

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 Figure 4.2. The MIP to minimize an NCDN cost function C (either MLU or latency) is as follows:

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

subject to

$$\sum_{j \in V} t_{ijk} + t_{iok} = T_{ik}, \quad \forall k \in K, i \in V \tag{4.2}$$

$$\sum_{k \in K} t_{ijk} = f_{ij}, \ \forall j \in V - X, i \in V \tag{4.3}$$

$$\sum_{k \in K} t_{ijk} + \sum_{k \in K} \delta_{ij} t_{iok} = f_{ij}, \ \forall j \in X, i \in V \tag{4.4}$$

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.

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

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

$$\sum_{i \in V, j \in V} f_{ije} \leq \alpha \times C_e, \quad \forall e \in E \tag{4.6}$$

$$\sum_{k \in K} x_{ik} S_k \leq D_i, \quad \forall i \in V \tag{4.7}$$

$$x_{ok} = 1, \quad \forall k \in K \tag{4.8}$$

$$\sum_{i \in V} x_{ik} \geq z_k, \quad \forall k \in K \tag{4.9}$$

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

$$t_{ijk} \leq x_{jk} T_{ik}, \quad \forall k \in K, i \in V, j \in V \cup \{o\} \tag{4.11}$$

$$t_{iok} \leq T_{ik}(1 - z_k), \quad \forall k \in K \tag{4.12}$$

$$x_{jk},\ z_k \in \{0,1\}, \quad \forall j \in V, k \in K$$

$$f_{ije},\ t_{ijk},\ t_{iok} \geq 0, \quad \forall i, j \in V, e \in E, k \in K$$

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 [13]. Finally, a simple extension to this MIP presented in a techreport [13] jointly optimizes routing given a TTM as well a CM. We have presented a CM-only formulation here as our find-

ings (in Section 4.4) 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 Appendix A.

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.

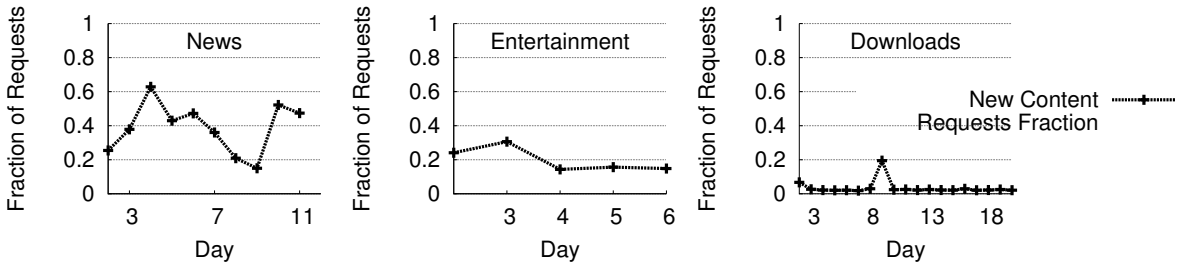


Figure 4.5. News and entertainment have a significant fraction of requests for new content on all days. Downloads has a small fraction of requests for new content on all days, except one.

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 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 [95] 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 [124, 45]. 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 [127]. We collect extensive and anonymized

access data reported from the download manager using Akamai's NetSession interface [96] 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 4.5 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 demand-aware 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 demand-aware versus being demand-oblivious 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 [164].

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 equiva-

lent 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 InvCap (with ECMP) as the routing strategy. InvCap 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 a demand-oblivious placement, LRU, but it uses an demand-aware, 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 [68]. 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 demand-oblivious placement and the implicit assumption is that the content demand and demand-oblivious 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 InvCap 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.

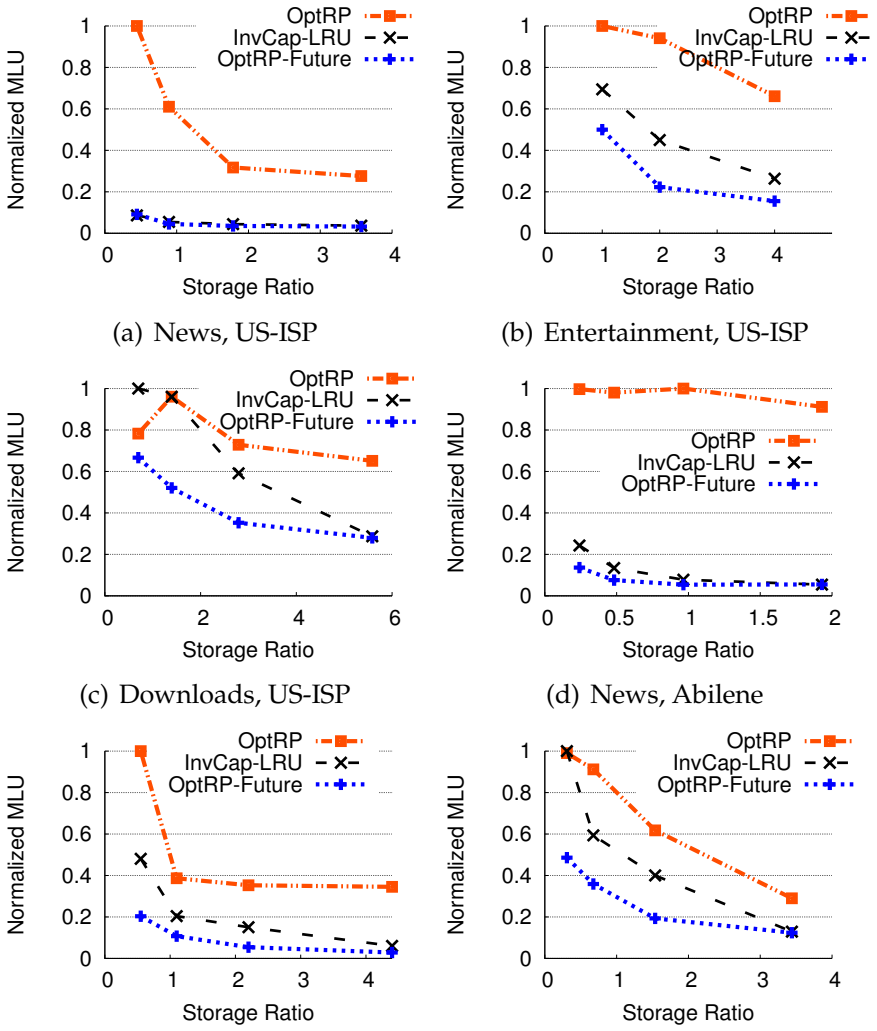
We add a suffix **-L** to the names of a scheme if it is optimizing for latency cost instead of MLU, e.g. **OptRP-L**.

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.

4.4.3 Comparison of network cost

4.4.3.1 Analysis of video & downloads traffic

Figure 4.6 shows the results for the news, entertainment and downloads traces on Abilene and US-ISP. Our first observation is that a realistic demand-aware placement and routing scheme, OptRP, performs significantly worse than a completely demand-oblivious 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

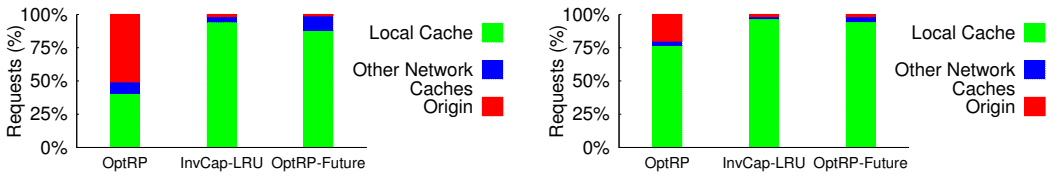


(a) News, US-ISP (b) Entertainment, US-ISP

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

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

Figure 4.6. Demand-aware OptRP performs much worse than demand-oblivious InvCap-LRU. OptRP-Future performs moderately better than InvCap-LRU primarily at small storage ratios.



(a) News, Abilene (b) Entertainment, Abilene

Figure 4.7. [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.

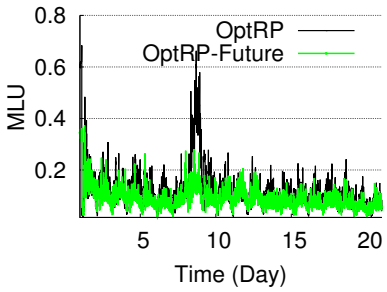
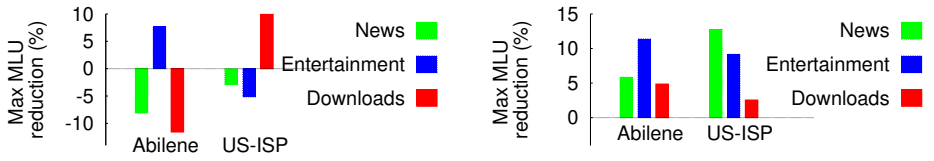


Figure 4.8. [Downloads, US-ISP] OptRP incurs a very high MLU on one "peak load" day.



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

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

Figure 4.9. [All traces] Optimizing routing yields little improvement to MLU of either InvCap-LRU or InvCap-OptP-Future

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 4.5). 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 4.7 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 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 4.8). 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 4.5). 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, demand-aware 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 demand-aware placement only if the future demand can be accurately known or estimated. Otherwise, a simpler demand-oblivious scheme such as LRU suffices.

How are the above conclusions impacted if InvCap-LRU were to optimize routing or OptRP-Future were to use InvCap 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 4.9. 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 demand-oblivious 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 [150] and Apple HLS [23] 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 4.10). Due to chunking, the maximum difference between the MLU of InvCap-LRU and OptRP-Future reduces from 2.5× to 1.4×. At

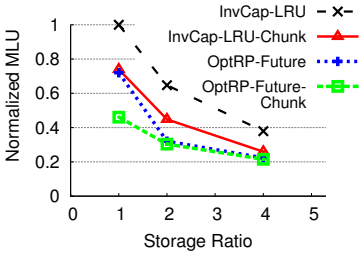


Figure 4.10. [Entertainment, US-ISP] Content chunking helps bridge the gap between InvCap-LRU and OptRP-Future.

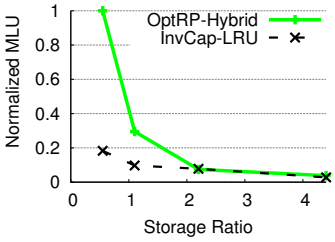


Figure 4.11. [Entertainment, Abilene] Hybrid placement schemes perform at best as well as InvCap-LRU.

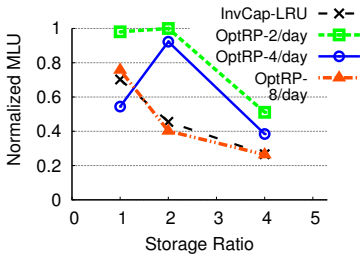


Figure 4.12. [Entertainment, US-ISP] OptRP does not outperform InvCap-LRU despite engineering 8 times a day.

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 4.10). 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 demand-aware schemes

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

First, we evaluate a hybrid placement scheme, which splits the storage at each node into two parts - one for a demand-aware placement based on the previous day's content demand (80% of storage) and the other for placing the content in a

demand-oblivious LRU manner (20% of storage). This hybrid strategy is similar to that used in [24]. 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 demand-oblivious 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 demand-oblivious and demand-aware schemes, both of which are extreme cases of a hybrid strategy. However, we were unable to design simple hybrid strategies that consistently outperformed fully demand-oblivious placement and routing.

Next, we analyze the performance of demand-aware 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 4.12 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 demand-aware placement requires considerable effort—measuring content matrix, solving a computationally intensive optimization, and moving content to new locations. Further, a demand-aware placement needs to be executed 8 times a day (or possibly more) even to match the cost achieved by a demand-oblivious strategy. Our position is that NCDNs are better served by opting for a much simpler demand-oblivious strategy and provisioning more storage, in which case, a demand-oblivious strategy already obtains a network cost close to the best a demand-aware strategy can possibly achieve.

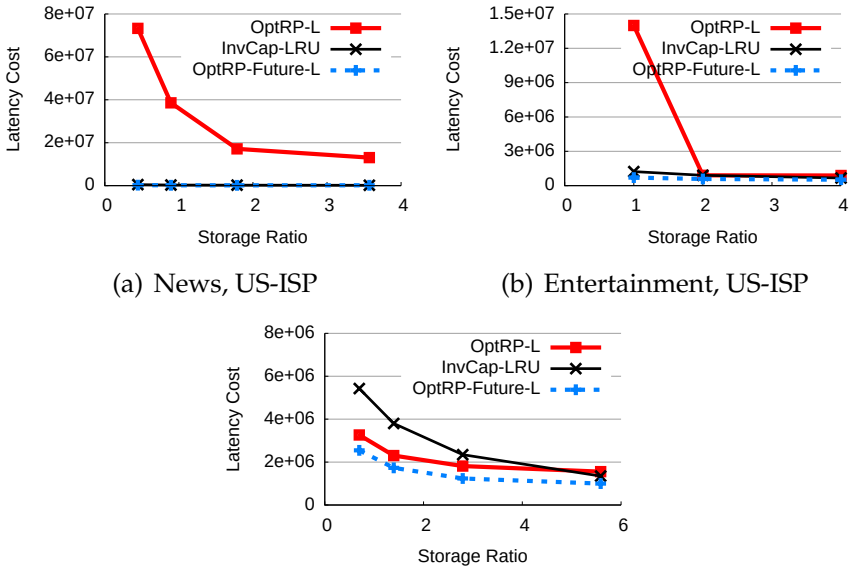
4.4.4 Comparison of latency cost

Latency cost metric models user-perceived latency in the network. In our evaluation, we compare InvCap-LRU scheme, which is a completely demand-oblivious 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.

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 4.13. 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



(a) News, US-ISP

(b) Entertainment, US-ISP

(c) Downloads, US-ISP

Figure 4.13. A realistic demand-aware scheme, OptRP-L causes excessively high latency costs in some cases. InvCap-LRU achieves latency costs close to ideal demand-aware scheme, OptRP-Future-L, at higher storage ratios.

that NCDNs can achieve close to best latency costs with a demand-oblivious 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 exam-

ine the benefit of optimizing routing over an unplanned routing scheme, InvCap. 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

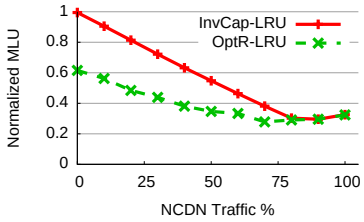


Figure 4.14. [News, US-ISP]Network costs at varying fractions of NCDN traffic in an ISP network.

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 (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 4.14. 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 InvCap 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 delivery 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 demand-oblivious schemes for routing and placement of NCDN content, such as InvCap 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, demand-oblivious schemes to perform as well as demand-aware 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

A GLOBAL NAME SERVICE FOR A HIGHLY MOBILE INTERNETWORK

"Mobile" has long arrived, but the Internet remains unmoved. Today, there is roughly one cellphone per human; the number of smartphones sold last year alone roughly equals the number of wired hosts on the Internet [77]; and the total traffic originated by mobiles is poised to approach that by wired devices [42]. However, the current Internet continues to operate as it did when dominated by tethered hosts, simply ignoring frequent endpoint mobility.

Today, an application developer can not easily initiate communication with a smartphone even when it has a public IP address as there is no global infrastructure support for locating it. Applications like smartphone notification systems, playback video, or cloud storage have to develop application-level support to enable a seamless experience for their users even as they change addresses several times a day, or let connections break (as popular VoIP apps do today). The lack of intrinsic support for mobility means that developers are forced to redundantly develop and maintain common-case functionality. Furthermore, we are paying an unknowable price in terms of long-term growth and innovation by straitjacketing communication initiation to be unidirectional.

Many before us have criticized the Internet architecture's poor support for mobility as well as multihoming [99, 9, 65, 126]. A common criticism is the Internet's so-called conflation of identity and location, i.e., the use of an IP address both to represent the identity of an interface as well as its network location, which is problematic for mobility (same identity, changing locations) and multihoming (single

identity, multiple locations). It is commonly accepted wisdom that a cleaner separation of identity and location is instrumental to fixing these problems. However, the Internet does separate identities (domain-names) from network locations (IP addresses) through DNS. Most high-level programming languages also provide syntactic sugar to `connect` to names remaining oblivious to IP addresses; and techniques from a long line of work on connection migration could be employed to seamlessly handle mid-connection mobility.

But a key missing element from this package today is a distributed name resolution infrastructure that can scale to orders of magnitude higher update rates than envisioned when DNS was created. To appreciate the envisioned scale, consider tens of billions of mobile identifiers changing network addresses at least tens of times per day. DNS's heavy reliance on TTL-based caching, a key strength recognized by its creators, researchers, and operators alike, poses a significant handicap by increasing update propagation delays, load on name servers, and overall client-perceived latency. It is not uncommon for DNS update propagation to take a day or more, resulting in long outage times when online services have to be moved unexpectedly, prompting cries for help on operator forums [8, 133]. A less widely noted limitation of DNS is its reliance on hierarchical names for scaling via federation and its single root of trust, which constrains mobile applications from selecting arbitrary application-specific names (as elaborated in Section 5.1.2 and Section 5.2.2).

Our position is that seamless support for mobility requires a logically centralized global name service that rapidly translates identities to locations irrespective of how exactly identities and locations are individually represented. Our primary contribution is the design, implementation, and evaluation of Auspice, a distributed system that helps address this challenge. Compared to today's ICANN/DNS-based approach, our approach cleanly separates name resolution from adjudica-

tion and certification issues (Section 5.2.2). Auspice is also deployable as a managed DNS provider in today's Internet; compared to them, a key strength of Auspice is a *demand-aware* replica placement engine that significantly reduces the *time-to-connect* to mobile destinations in a cost-effective manner. Under light load, Auspice's demand-aware replica placement aggressively uses available resources to massively replicate name records, while under heavy load, it carefully controls the number and choice of replica locations based on the read-write patterns and pockets of high demand for each name.

We have implemented a prototype of Auspice as a geo-distributed key-value store to serve as a flexible name resolution service for the current Internet as well as several "future" Internet or endpoint architectures such as MobilityFirst[126], HIP[99], or XIA[88]. We have extensively evaluated Auspice using a combination of Planetlab, emulation clusters, and Amazon EC2. Our contributions are as follows.

1. A case for a global name service as an indispensable part of any Internetwork design with intrinsic support for high mobility (Section 5.1).

2. Auspice, a scalable, geo-distributed, federated global name service that significantly reduces the time-to-connect under any given resource constraints despite high mobility and arbitrary endpoint identifiers (Section 5.2,Section 5.3).

3. A proof-of-concept demonstration of intrinsic support for—*(i)* all four types of endpoint mobility; *(ii)* novel context-aware delivery primitives that generalize name- or address-based communication—over the current Internet as well as MobilityFirst [126] (Section 5.3.3).

4. Comparison against several best-of-breed managed DNS services showing that Auspice's demand-aware approach significantly lowers time-to-connect

and/or update cost even for today's (hardly mobile) domain names (Section 5.3.4).

To provide a historical perspective, until the early 80s, the Internet relied on a system called `HOSTS.TXT` for name resolution, which was simply a centrally maintained text file distributed to all hosts. The current Internet's distributed DNS arose in response to the rapidly increasing file size and distribution costs. Mockapetris and Dunlap [121] point to TTL-based caching to reduce load and response times as a key strength, noting that "*the XEROX system* [Grapevine [153]] *was then ... the most sophisticated name service in existence, but it was not clear that its heavy use of replication, light use of caching ... were appropriate*". We have since come a full circle, turning to active replication (Section 5.1.2) in Auspice in order to address the challenges of mobility, a concern that wasn't particularly pressing in the 80s. Compared to classical systems like Grapevine or ClearingHouse, Auspice enables support for automated *demand-aware* replica placement for *arbitrary names* (using several modern design elements such as consensus, the key-value abstraction, self-certifying names, consistent hashing, etc). Auspice, through its support for context-aware delivery, is also a step towards addressing some of the challenges to which Lampson alludes on representing "descriptive names" [108].

5.1 Case for a global name service

Given the huge body of prior work specifically on mobility as well as more broadly on Internet architecture, it is natural to begin by asking: Is a global name resolution service critical to handling mobility if we had the luxury of refactoring Internet naming and routing from a clean slate?

5.1.1 Internet mobility background

Despite a staggering diversity of proposals re-architecting Internet naming and routing, we find that they explicitly or implicitly embed one of three broad approaches to handling mobility–*indirection-based routing*, *global name-to-address resolution*, or *name-based routing*–based on how they go from the name of an endpoint to the endpoint itself.

Indirection-based routing schemes are simple as an endpoint remains oblivious to the mobility of other endpoints. No name-to-address[1] lookup is needed at connection initiation time as a human-readable name maps to a *home address* (an IP address in Mobile IP [135] or a flat identifier's consistent-hash location in i3 [160]) that rarely changes by design. Mid-connection mobility, even when both endpoints move concurrently, is seamless to endpoints. However, the data plane pays the price for this simplicity—every data packet must be routed via an indirection agent at the home address, potentially causing significant routing stretch, e.g., two participants at a conference may in each direction need to detour packets halfway across the world despite being in the same room. Furthermore, indirection-based schemes require widespread deployment of indirection agents across different domains, posing a barrier to immediate adoption.

Global name-to-address resolution schemes rely on a distributed service to resolve names to addresses as the first step in connection establishment. The current Internet's DNS as well as a number of designs addressing the Internet's so-called identity-location conflation problem also need such a resolution infrastructure, e.g., to translate a self-certifying host identifier in HIP [99], AIP[20], XIA[88], or MobilityFirst[5]) or an identifier in LISP [9] or HAIR [65] to either an IP address [99], a self-certifying network identifier [20, 88, 5], or a hierarchical locator

[1]We use the terms *name* and *identifier* interchangeably; likewise for the terms *address* and *location*.

[65] that encodes routing information. Global name-to-address resolution schemes also subsume DHT-based Internet architectures such as LNA [29, 170] as well as resolution systems like CoDoNS [143] that present a DHT-based drop-in replacement for DNS.

Global name-to-address resolution schemes need explicit support at endpoints to handle mid-connection mobility. There is a general consensus [156, 27, 74] that end-to-end connection migration, i.e., bilaterally without relying on an external service, suffices to migrate connections efficiently when endpoints move one at a time, but an external resolution service is needed to support concurrent mobility. Although the latter is seen as a rare case in most connection migration work, it can be common in disconnection-tolerant, mobile application scenarios, e.g., when a user closes her laptop at home and opens it at a coffee shop to continue watching a movie, by which time the cloud-hosted virtual server may have been migrated for load balancing.

Name-based routing schemes in the ideal have a tantalizing intellectual lure—to seamlessly handle mobility by routing directly over names with no resolution step—but are marred by several fundamental and practical challenges. First, name-based routing approaches can support seamless mobility only if routing update propagation delays are on the order of milliseconds, a daunting challenge given that interdomain routing can take several minutes to converge today. Second, theoretical results on compact routing [104] suggest discouraging fundamental tradeoffs between the size of forwarding tables at routers and path stretch even without any mobility or multihoming, e.g., routing over N flat identifiers entails a prohibitive $\Omega(N)$ forwarding table size per router in order to ensure a small constant stretch factor ($\approx$3) compared to shortest-path routing. Simulation-based studies of flat-label routing strategies (e.g., ROFL [35]) reaffirm pessimistic conclusions about its scalability.

Although it may appear that the scalability limitations of name-based routing can be alleviated by adding a hierarchical structure to names [83, 103, 97] (e.g., NDN-style [97] names such as `/umass/phone42/call3/frame7`), frequent mobility still poses a challenge as routers would have to maintain special forwarding entries for "displaced names", i.e., names that move from their hierarchically organized namespace (say, from `/umass` to `/comcast` in this example) for longest-prefix matching to work correctly. That is, high mobility effectively makes routing directly over structured names as hard as routing over flat names unless indirection or a name resolution infrastructure is used, a conjecture that has recently been empirically reinforced by Gao et al. [76].

Summary. Our position is that a global name-to-address resolution service is critical for handling high mobility in any network architecture as it offers the best combination of trade-offs: (1) a constant update overhead per mobility event to the name service, (2) a modest connection establishment overhead and rapid mid-connection mobility, (3) no data path inflation beyond underlying policy routing, and (4) small forwarding table sizes in conjunction with aggregatable addresses (IP prefixes like today or self-certifying network addresses [5, 88]). Perhaps the most compelling argument for global name-to-address resolution is our decades of familiarity with DNS and the Internet; handling mobility would be a drop-in replacement to DNS provided we address the challenge of building a distributed system that scales to billions of devices making many updates a day and yet returns up-to-date responses within milliseconds.

5.1.2 Limitations of DNS

What specific design traits of DNS make it poorly suited for mobile applications? The first two traits below limit its scalability with respect to the rate of endpoint mobility, and the third limits its scalability with respect to the size of the

namespace if applications were to have the luxury of using arbitrary (but fixed) names.

(1) *TTL-based caching:* TTL-based caching is the single-most important mechanism for DNS's scalability; caching not only helps DNS sustain essentially arbitrarily high *lookup load* but also dramatically reduces client-perceived *lookup latency* for cache hits. However, caching is ineffective when TTLs are near-zero, as would have to be the case under high mobility, causing both increased load on name servers and higher client-perceived latencies. Caching is also less effective if lookups are distributed relatively uniformly, as could be the case with mobile device names, unlike lookups for today's domain names that are highly skewed [100, 131].

(2) *Static placement:* Authoritative DNS servers are essentially rendezvous points that allow a mobile endpoint to inform potential correspondents of its current location(s). In order to reduce the time-to-connect, authoritative servers must be located close to potential correspondents. However, authoritative server locations today are static, either close to a mobile endpoint's "home" location or a prepackaged set of geo-distributed locations provided by a managed DNS provider. Engineering a scalable geo-distributed system that can dynamically move object replicas in a demand-aware manner is nontrivial and real-world examples of such systems have only recently begun to emerge [51].

(3) *Hierarchical names:* The hierarchical structure of DNS names is key to leveraging *federation* to scale to an arbitrary number of names by delegating different portions of the name space (or zones) to different organizations. For example, root name servers today only have to maintain state for a small number of top-level domain names. In contrast, arbitrary or flat names, e.g., "`JohnSmith3142's watch`" can not be supported in DNS while retaining the scaling benefits of federation as the root name servers would have to maintain nonzero state, e.g., at least the

authoritative name server(s) and the DNSSEC key of a name, for essentially all names. Our position is that the design of a general-purpose global name service must not restrict the structure of names as names carry application-specific semantics; in Section 5.3.3.3, we show examples of novel context-aware communication primitives that are feasible with unrestricted names.

Our approach to address the first two issues above relies on *active* and *demand-aware* replication: (1) Active replication significantly reduces (but does not eliminate) the reliance on passive caching; (2) Demand-aware replication ensures that active replicas of a name record are accessible close to clients querying the name, so as to reduce the overall time-to-connect. A glib but pedagogically helpful way to highlight the difference from DNS is that, in the extreme case, our approach can create an active replica of a name record near every DNS local name server that stores a passively cached copy today. Our approach addresses the third issue above by cleanly separating resolution of names from adjudication and certification. We explain our approach in detail next.

5.2 Auspice design & implementation

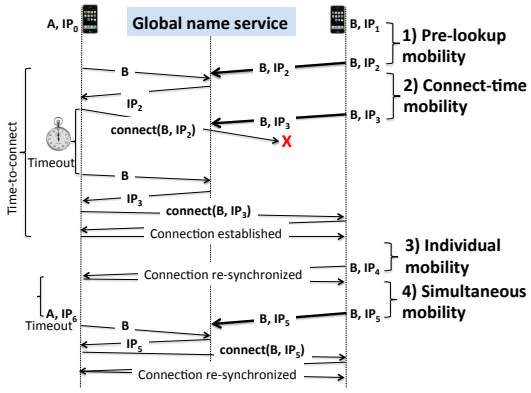


Figure 5.1. Four kinds of mobility—(1) *pre-lookup*, (2) *connect-time*, (3) *individual*, (4) *simultaneous*—three of which require a global name service.

Our envisioned GNS enables endpoint mobility as shown in Figure 5.1. An endpoint A initiates communication with another endpoint B by querying the GNS for B's current addresses and connecting to one of them, thereby enabling *pre-lookup* mobility. If B moves after A's query but before before a connection has been mutually established via a three-way handshake (*connect-time* mobility), A times out and reverts back to the GNS. After a connection has been established, if either endpoint moves one at a time (*individual* mobility), it can re-synchronize the connection with a bilateral three-way handshake without relying upon the GNS (noting however that router-level late-binding proposals relying on a GNS-like infrastructure have also been proposed [125, 5]). If an endpoint moves mid-connection after the other endpoint has moved but before it could re-synchronize the connection (*simultaneous* mobility), one or both endpoint(s) eventually query the GNS and re-synchronize the connection.

5.2.1 Design goals

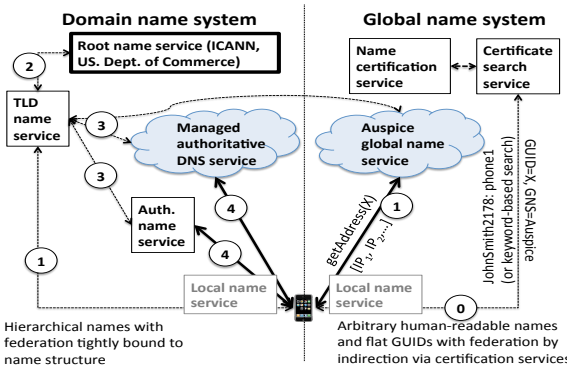


Figure 5.2. DNS vs. GNS: Auspice can be deployed as a managed DNS provider today (left) or as a GNS provider that provides resolution service for its customer GUIDs (right). Name certification services bind a human-readable name to a GUID and its GNS provider, and certificate search services can help index and distribute certificates from all certification services. Solid (dotted) lines represent frequent (infrequent) query paths for a given mobile destination. Except for the tightly controlled DNS root service, all services above are designed to be purveyed competitively.

Much of the envisioned functionality of a GNS as above boils down to one over-arching distributed systems challenge: *any principal–endpoint or router–should get the look and feel of a high-availability name service that is nearby (≈ few milliseconds) and rapidly returns up-to-date responses*. A more precise breakdown of goals is as follows.

(1) Time-to-connect performance: The design must ensure low latencies for name lookups to return up-to-date values, which determines the *time to connect* to a destination when the value being queried for is an address like above.

(2) Resource cost: The design must ensure low replication cost. A naive way to minimize lookup latencies is to replicate every name 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) Security: The design must be robust to malicious users attempting to hijack or corrupt name records. The design must support flexible access control policies to ensure the desired level of privacy of name records.

(5) Federation: The design must allow different name service providers to coexist and for users to freely choose one or more preferred providers.

(6) Extensibility: The design must be agnostic to how names, addresses, and resolution policies are represented by a future Internetwork. In particular, it should support flat names and a rich set of attributes and resolution policies for multihomed mobility (e.g.,"prefer WiFi to cellular"), etc.

5.2.2 Design overview

To address the above goals, the Auspice GNS 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 enables extensibility. Each *name record* in Auspice is associated with a globally unique identifier (GUID) that is the record's primary key. A name 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 $\mid K_1, V_1 \mid K_2, V_2 \mid \cdots$

The GUID is a self-certifying identifier computed as a compact one-way hash of a public key. Each name record is aliased to one or more globally unique human-readable names that are bound to the GUID by a certificate supplied by one or more *name certification service(s)* (NCS). Loosely speaking, the human-readable name is analogous to a DNS domain name and a name record to a zone file, but with the following important differences.

Security. As shown in Fig. 5.2, to initiate communication with a destination Y, an endpoint X must first obtain a certificate of the form [`JohnSmith2178:Phone1`, $Y, P]_{K^-}$ that binds the human-readable name to the GUID Y and its GNS provider P, and is signed by the private key K^- of an NCS that X trusts. A certificate search service (e.g., a search engine or ISP) can help index certificates from different NCSes, and help X find a certificate from a trusted NCS, and even find the human-readable name based on keywords.

Federation. Unlike ICANN and root DNS servers that respectively act as a single name adjudication authority and root of trust for certification, our approach decentralizes trust across different NCS providers, potentially allowing endpoints to use quorum-based approaches to resolve conflicting name certificates. More importantly, our federation approach allows endpoints to select arbitrary human-

readable names and NCS providers unlike DNS that restricts domain names to be hierarchical and federation and the DNSSEC keychain to strictly follow the name structure. An inevitable implication of decentralizing trust is that two end-points can communicate securely only if they share a trusted NCS provider, but this change we argue is preferable to and a strict generalization of the single-root-of-trust model that some perceive as arbitrary [11, 10].

Extensibility. Our design cleanly separates the GNS provider's resource-intensive responsibility of name resolution under high mobility from the slow-changing certification process. It also allows for the GNS provider to be deployed today as a managed authoritative DNS provider (Fig. 5.2) with the DNSSEC key deriving the GUID. Finally, the key-value store API enables an extensible name record representation. By default, each top-level key has associated read and write ACLs that could either be a blacklist or whitelist of GUIDs that respectively have read or write access. For example, a name record for GUID `X` that helps context-aware delivery or multihoming policies (detailed in Section 5.3.3.3) is below. `{X: {IPs: [{IP: 23.55.66.43, pla 62.44.65.75, plan: Limited}], geoloc: {[lat,long], readWhitelist: [Y,Z]}, multihome_policy: Unlimited}}`

5.2.3 Auspice's geo-distributed design

Next, we explain how Auspice achieves the first three design goals. 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 name record in accordance with (1) the lookup and update request rates for the name, (2) the geo-distribution of requests for the name, and (3) the aggregate request load across all names.

Figure 5.3 illustrates the placement engine. Each name is associated with a fixed number, F, of *replica-controllers* and a variable number of *active replicas* of

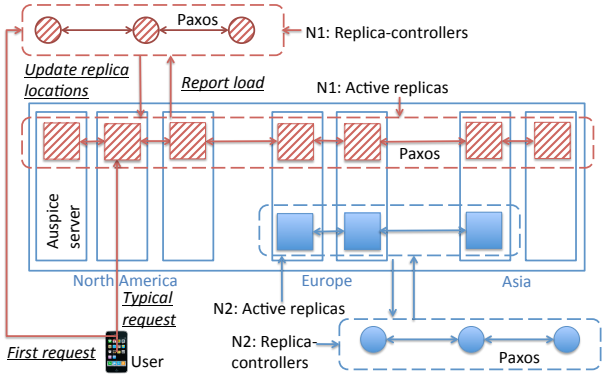


Figure 5.3. Geo-distributed name servers in Auspice. Replica-controllers (logically separate from active replicas) decide placement of active replicas and active replicas handle requests from end-users. N1 is a globally popular name and is replicated globally; name N2 is popular in select regions and is replicated in those regions.

the corresponding name record. The name'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 names and nodes. The replica-controllers are responsible only for determining the number and locations of the active replicas, and the actives replicas are responsible for maintaining the actual name record and processing client requests. The replica-controllers implement a replicated state machine using Paxos [106] in order to maintain a consistent view of the current set of active replicas.

A name'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 name 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 name.

5.2.3.1 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 name servers to determine the number and locations of active replicas for each name 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 name 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 name while placing the rest randomly so as to balance the potentially conflicting goals of reducing latency and balancing load among name servers.

Specifically, the placement algorithm computes the number of replicas for a name as $(F + \beta r_i/w_i)$, where r_i and w_i are the lookup and update rates of name i; F is the minimum number of replicas needed to meet the availability goal (§5.2.1); 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 uti-

lization fraction μ. For simplicity of exposition, suppose read and write operations impose the same load, and the total capacity across all name servers (in reads/sec) is C. Then, β is set so that

$$\mu C = \sum_i r_i + \sum_i (F + \beta \frac{r_i}{w_i}) w_i \tag{5.1}$$

where the right hand side represents the total load summed across all names. 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 name 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 $\nu(F + \beta r_i/w_i)$ name servers that are respectively the closest to the top $\nu(F + \beta r_i/w_i)$ regions sorted by demand for name 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 name 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.

5.2.3.2 Client request routing

A client request is routed from an end-host to a suitable name server as follows. The set of all name servers in an Auspice instance is known to each member name server and can be obtained from a well-known location. End-hosts can either directly send requests to a name server or channel them through a local name server like today. When a local name server encounters a request for a name for the first time, it uses the known set of all name servers and consistent hashing to determine the replica-controllers for that name and sends the request to the closest replica-controller. The replica-controller returns the set of active replicas for the name 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 local name server. Each local name server maintains an estimate of the round-trip latency to all name 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 [116]. To incorporate load-induced latency, the latency estimate to a name server is passively measured as a moving average over lookups sent to that name server. The local name server also maintains a timeout value based on the moving average and variance of the estimates. If a lookup request sent to a name server times out, the local name server infers that either the server or network route is congested, and it multiplicatively increases its latency estimate to that name server by a fixed factor. Thus, if multiple lookups sent to a name server time out, the estimated latency shoots up and the local name server stops sending requests to that name 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).

5.2.3.3 Consistency with static replication

As a global name-to-address resolution service, Auspice must at least ensure this eventual consistency property: *all active replicas must eventually return the same value of the name 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 name record and no replica failures for sufficiently long. Violating this property means that a mobile client may be persistently unreachable even though it is no longer moving (updating addresses).

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 [53], 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 write ordering. As Auspice is designed to be an expressive name service with sophisticated attributes, it may be useful in some scenarios to ensure that update operations (like appending to or deleting from a list) to a name are applied in the same order by all active replicas. Ensuring a total ordering of all updates to a

name is a stronger property than eventual consistency, calling for a state-machine approach, which Auspice supports as an option.

To this end, active replicas for a name participate in a Paxos instance maintained separately for each name (distinct from Paxos used by replica-controllers to compute active replicas for that name). 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).

5.2.3.4 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 name to its *current* set of active replicas as obtained from any of the consistently-hashed replica-controllers for the name. 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 [107], 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].

5.2.3.5 Scalability: A performance-cost analysis

Cost. Auspice's replica placement scheme (Eq. 5.1) 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 name, where M is the total number of name server locations. Thus, at light load, Auspice may replicate every name at every location, while under heavy load, it may create exactly F replicas for all but the most popular names. In the common case, a lookup involves one request and response between a local name server and an active replica; an update involves $\approx$ thrice (twice) as many messages as the number of active replicas with total write ordering (eventual) consistency.

Performance. The worst-case time-to-connect latency for a name i depends on the lookup latency l_i, the update rate w_i, the worst-case update propagation latency d_i, i.e., the time for all active replicas to receive an update, and the connect timeout T (Fig. 5.1), as follows [1].

$$\text{TTC}_i \;=\; l_i[1 + (e^{w_i d_i} - 1)(1 + \frac{T}{l_i})] \tag{5.2}$$

Thus, the time-to-connect increases with (1) the lookup latency l_i that in turn improves with demand-aware replication; (2) the update rate w_i and update propagation delay d_i that in combination determine the likelihood of obtaining a stale response, noting that the latter increases with more aggressive replication; and (3) the connect timeout T that is at most the default transport-layer timeout (e.g., a few seconds for TCP) and potentially as low as the round-trip delay between the connecting client and the destination being connected to if the destination network is capable of generating an "no route to host" error message.

TTLs. The above analysis implicitly assumes near-zero TTLs. With a nonzero TTL_i for name i, the worst-case time-to-connect can be approximated as [1].

$$\begin{aligned} \text{TTC}_i \;\simeq\; & \tau_i \, \frac{(r_i + w_i + 1/\text{TTL}_i + r_i w_i \text{TTL}_i)}{(1 + r_i \text{TTL}_i)(r_i + w_i + 1/\text{TTL}_i)} \\ & + \frac{T \; r_i \; w_i}{(r_i + 1/\text{TTL}_i)(r_i + w_i + 1/\text{TTL}_i)} \end{aligned} \tag{5.3}$$

where τ_i above is TTC_i (with a 0 TTL) as in Eq 5.2. Thus, a long TTL is meaningful only if the update rate w_i is low; if so, a carefully chosen TTL can reduce the load on the system as well as the client-perceived time-to-connect; if not, a long TTL can inflate the time-to-connect by the connect timeout T (= the second term above for $w_i \gg r_i$ and high TTL_i).

Comparison to DNS. All of the above analyses are applicable also to geo-replicated managed DNS providers were they to employ Auspice's demand-aware replication approach. The main difference between Auspice and today's managed DNS providers that rely on simplistic static replica placement schemes is in the lookup latency l_i achieved for any given resource cost; we evaluate this performance-cost tradeoff extensively in our experiments (Section 5.3.2 and Section 5.3.4).

5.2.3.6 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.

5.3 Evaluation

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) Can Auspice serve as a complete, end-to-end solution for mobility and enable novel communication abstractions? (3) How does Auspice's cost-performance tradeoff compare to best-of-breed managed DNS services for today's (hardly mobile) domain name workloads?

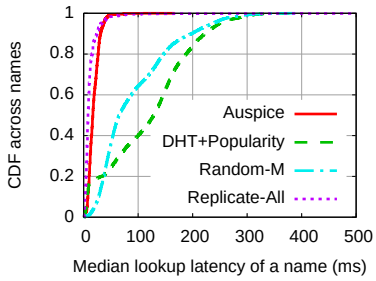
5.3.1 Experimental setup

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.

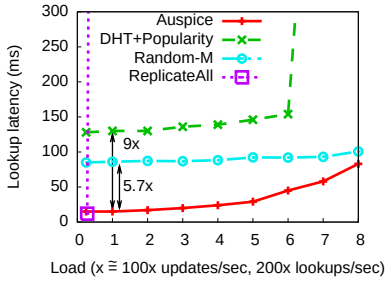
Workload: There is no real workload today of clients querying a name service in order to communicate with mobile devices frequently moving across different network addresses, both because such a name service does not exist and mobile devices do not have publicly visible IP addresses. So we conduct an evaluation using synthetic workloads for device names (Section 5.3.2), 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$\times$ the average lookup rate; the update rate is similarly distributed and drawn independently.

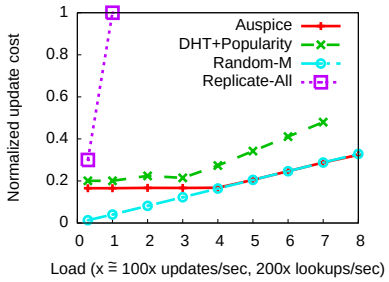
How requests are geographically distributed is clearly important for evaluating a replica placement scheme. We define the *geo-locality* of a name as the fraction of requests from the top-10% of regions where the name is most popular. This parameter ranges from 0.1 (least locality) to 1 (high locality). For a device name with geo-locality of g, a fraction g of the requests are assumed to originate from



(a) Lookup latencies (load = 0.3)



(b) Lookup latency vs. load



(c) Update cost vs. load

Figure 5.4. Auspice has up to 5.7× to 9× lower latencies than Random-M and DHT+Popularity reps. (5.4(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.

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 [105], 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).

Replication schemes compared: Auspice uses the replica placement strategy as described in Section 5.2 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[143]. 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.

5.3.2 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 nameservers or local nameservers so as to emulate an 80-nameserver 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.

5.3.2.1 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 5.4(a) shows the distribution of median lookup latency across names at the smallest load level (load = 0.3). Figure 5.4(b) shows load-vs-lookup latency curve for schemes, where "lookup latency" refers to the mean of the median lookup latencies of names. Figure 5.4(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 5.4(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. 5.4(b)) due to its low update costs, but its lookup latencies are higher as it only creates 3 replicas randomly.

Auspice has $5.7\times - 9\times$ lower latencies over Random-M and DHT+Popularity respectively (Figure 5.4(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. 5.1), 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

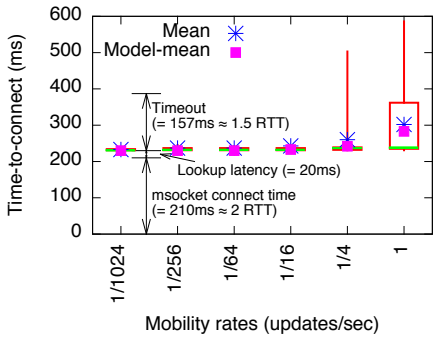
and Random-M have equal update costs for loads ≥ 4 (Figure 5.4(c)). Reducing the number of replicas at higher loads allows Auspice to limit the update cost and sustain a maximum request load that is equal to Random-M.

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.

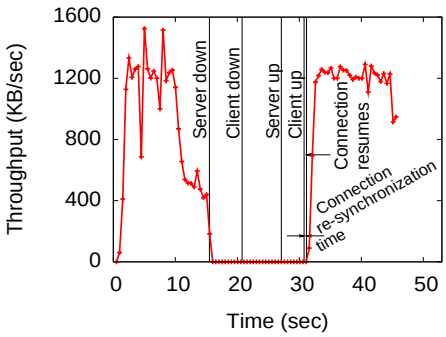
DHT+Popularity incurs higher update costs than Auspice even though both schemes create nearly equal numbers of replicas at every load level. This is because DHT+Popularity decides the number of replicas of a name only based on its popularity, i.e., lookup rates, while Auspice decides the number of replicas 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.

5.3.2.2 Update latency, update propagation delay

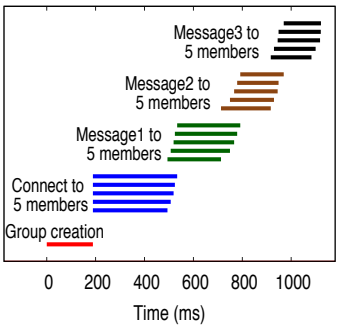
The *client-perceived update latency*, i.e., the time from when when a client sends an update to when it receives a confirmation. These numbers are measured from the experiment in Section 5.3.2.1 for load=0.3. The median and 90th percentile update latency for Auspice with total write ordering is 284ms 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.



(a) E2E time-to-connect



(b) Simultaneous mobility



(c) Context-aware delivery

Figure 5.5. (a) Time-to-connect≈lookup latency for moderate mobility rates ($< \frac{1}{10s}$) as Auspice returns up-to-date responses w.h.p., but sharply rises thereafter (Eq. 5.2); (b) Simultaneous mobility recovery in ≈2 RTTs after both endpoints resurface; (c) Context-aware delivery showing 3 messages geo-cast to 5 members.

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. As shown in Section 5.2.3.5, 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.

5.3.3 End-to-end mobility case studies

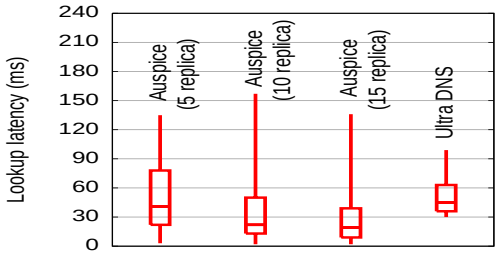


Figure 5.6. Lookup latency: Auspice with 5 replicas is comparable to UltraDNS (16 replicas); Auspice with 15 replicas has 60% lower latency than UltraDNS.

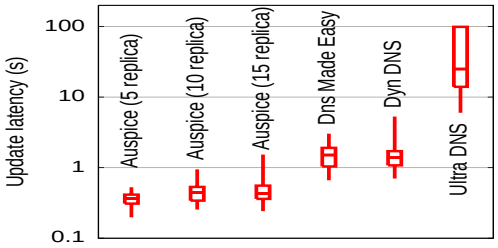


Figure 5.7. Update propagation delay: Auspice with 5 replicas is 1.0 to 24.7 secs lower than three top-tier managed DNS service providers.

Can Auspice serve as the basis of a complete end-to-end mobility solution? To address this question, we have developed msocket, a user-level socket library that interoperates with Auspice, and supports all four types of endpoint mobility. The details of msocket's design and implementation is the subject of a separate paper [7]. Here, we *use* msocket to show proof-of-concept of some of Auspice's capabilities.

5.3.3.1 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. 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 Section 5.2.2) 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 (as in Fig. 5.1). The experiment is conducted on Plan-

etLab 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 (2×105 ms) [7]. As defined in Eq. 5.2, 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 5.5(a) 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 ≈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 ≈two round-trips between the client and the mobile without relying on Auspice (not shown here).

Figure 5.5(a) also shows that the time-to-connect as predicted by our analytical model (Eq. 5.2) are close to those observed in the experiment, thereby re-affirming our design.

5.3.3.2 Simultaneous endpoint mobility

Figure 5.5(b) shows an experiment involving simultaneous mobility. The client is an Android phone using msocket via a WiFi interface to connect to a publicly addressable Planetlab machine at time 0. The server and client shut down their interfaces respectively around 15 and 20 sec. Subsequently, the server restarts its interface and starts listening on a different port and updates Auspice accordingly. After that, the client restarts its interface and attempts to re-synchronize the connection. This re-synchronization time is roughly 300ms as shown and consists of the following delays. The client performs a query to Auspice to resolve the server's GUID to its new socket address (IP, port), which takes roughly 50ms and mostly corresponds to the round-trip delay between the client and the Auspice nameserver. The remaining 250ms roughly correspond to 2 RTTs of delay between the client and the server that are separated by a round-trip delay of 120ms.

5.3.3.3 Context-aware delivery

Next, we show a proof of concept of context-aware communication, a novel communication primitive enabled by Auspice's extensible key-value API. Auspice allows applications to bind an msocket not only to human-readable names or GUIDs, but also to abstract context descriptors as in `msocket.bind("[geoloc: [lat,long],radi`
Writes to this msocket are reliably delivered to all GUIDs in the geo-fence created by this descriptor. Underneath the covers, msocket invokes Auspice to create on-demand a group GUID, i.e., a GUID with a membership field consisting of a set

of member GUIDs, and obtains this member set. msocket internally resolves each member GUID to its socket address and establishes an msocket connection for reliably delivery.

Figure 5.5(c) shows an experiment involving a group creator (also the message sender) on an Android phone and a number of potential members on PlanetLab nodes, 5 of which fake-register their coordinates in Auspice so as to appear to be within the created geo-fence. The RTT between the group creator and members is 125ms. The figure shows that group creation, a single call to Auspice that returns all member GUIDs, takes roughly 200ms. Subsequently, an internal msocket connect to each member involves another Auspice lookup to resolve the GUID to a socket address and connect in parallel to all 5 members, which takes 250-280ms. After this, the creator sends 3 short messages back-to-back that each take roughly 1 RTT to be reliably delivered.

More details of optimizing context-aware queries in Auspice, reducing membership staleness, the connection migration protocol, etc. are outside the scope of this paper [7]. This experiment seeks only to exemplify a powerful, new communication primitive enabled by context descriptors compared to strictly hierarchical DNS names, as argued in Section 5.1.2.

5.3.4 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).

5.3.4.1 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 [147]. We evaluate Auspice for three configurations with 5, 10, and 15 replicas of a name respectively.

Figure 5.6 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.

5.3.4.2 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 5.7.

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 [147] 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 [147].

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 findings in this paper, and are deferred to the technical report [1].

5.4 Related work

Our work draws on lessons learned from an enormous body of prior work on network architecture as well as distributed systems, as described in Section 5 and Section 5.1.1. We discuss related work not covered elsewhere in the paper here.

DNS: Many have studied issues related to performance, scalability, load balancing, or denial-of-service vulnerabilities in DNS's resolution infrastructure [132, 143, 34, 59]. Several DHT-based alternatives have been put forward [143, 52, 131] and we compare against one representative proposal, Codons [143]. 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 [169], an in-network DHT scheme that is similar in spirit to Random-M as evaluated in our experiments (Section 5.3) (with a more direct comparison in [1]), showing that demand-aware placement can dramatically outperform randomized placement. A more important qualitative distinction is that DMap ties federation to the interconnection structure between ISPs, which entails commensurate lookup latency penalties and potential incentive mismatches by mapping GUIDs to non-provider ISPs. In comparison, the Auspice

approach decouples the federation structure between GNS providers from that between ISPs.

Request redirection: Many prior systems have addressed the request redirection problem with data or services replicated across a wide-area network as discussed in the context of CDNs in Chapter 2. Examples include anycast services [73, 33, 173] 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 request redirection 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 low 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 [86] or within a single datacenter, or without replication. For example, Volley [15] 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 [6, 62, 4], 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 [51] 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 [107] and the more recent report on Viewstamped Replication Revisited [112].

5.5 Conclusions

In this paper, we presented the design, implementation, and evaluation of Auspice, a scalable, geo-distributed, federated global name service for any Internetwork where high mobility is the norm. The name service can resolve flexible identifiers (human-readable names, self-certifying identifiers, or arbitrary strings) to network locations or other attributes that can also be defined in a flexible manner. At the core of Auspice is a placement engine for replicating name records to achieve low lookup latency, low update cost, and high availability. 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. Our case studies confirm that Auspice can form the basis of an end-to-end mobility solution and also enable novel context-aware communication primitives that generalize name- or address-based communication. A pre-release version of Auspice on EC2 can be accessed through the developer portal at `http://gns.name`.

CHAPTER 6

SHRINK: QUANTIFYING AND LEVERAGING ENERGY-PERFORMANCE TRADEOFF IN CONTENT DATACENTERS

In today's content-dominated Internet [45, 124], it is not surprising that many datacenters and Points-of-Presence (PoPs) are dedicated to storing and serving content to end users. We call these datacenters and PoPs *content datacenters* (or, CDCs).

The increasing energy use of datacenters [81, 145, 87] has motivated a long line of research in *consolidation* schemes that aggregate a datacenter's load on a fraction of components and save energy by turning off unused components [39, 118, 142, 111]. Consolidation exploits the over-provisioning of resources in a datacenter and the diurnal variations in load to potentially reduce datacenter energy use by up to 50% [118].

The practicality of consolidation as an energy-saving tool for a CDC operator depends on its impact on user-perceived latency. User-perceived latency is a key metric on which operators are evaluated [146]. Further, a sharp latency inflation may be perceived as service unavailability, thereby causing a service-level agreement (SLA) violation and revenue loss for an operator [18, 120, 93]. Despite a large body of literature on consolidation schemes, quantifying the precise impact on user-perceived latencies is not well understood today.

Our position is that a lack of quantitative understanding of energy vs. latency tradeoff is potentially a major roadblock to widespread deployment of consolidation to reduce CDC energy use. To provide the insights needed for an operator

to reduce its energy use, we address these key questions: (1) What is the energy-latency tradeoff achieved by simple schemes for server and network consolidation in a CDC that uses well-known schemes for load balancing and network routing? (2) What is the additional energy savings achieved by network-aware server consolidation over network-unaware server consolidation considered previously?

Our primary contribution is to quantify the tradeoff between energy savings via consolidation and latencies for a real CDC's workload, and the design and implementation of a system, Shrink, to leverage this tradeoff. Shrink reduces the energy use of servers and switches via consolidation, while enabling operators to achieve the desired latency and hardware reliability. A novel aspect of Shrink is a network-aware server consolidation scheme that selects the active servers in a left-to-right order in a topology and achieves greater network energy savings over network-unaware server consolidation schemes. Further, our server and network consolidation schemes require only ECMP (equal-cost multipath) support for routing, and hence, are deployable with existing datacenter network fabrics.

Our work is grounded in implementation, due to which it accounts for several factors affecting latency - increased load on servers and network links, reduced storage and its impact on cache hit rates, non-steady state cache behavior due to on-off transitions, imperfect load balancing among servers - and hence, accurately quantifies the impact of consolidation on user-perceived latency. A key insight, supported via experiments, is that cache hit rates, despite server consolidation, remain close to an unconsolidated datacenter. A small reduction in hit rates helps mitigate the impact of consolidation on latencies. This finding is explained by the skewed content popularity obseved in real workloads, due to which working set size of content remains small compared to the storage available in a CDC.

We have built a prototype of Shrink using Squid [54], a caching proxy, and have deployed it on Amazon EC2 [19] and Emulab [175]. To conduct a realistic evalu-

ation, we have collected and used traces containing more than 2 billion requests generating nearly 200 TB of traffic from a datacenter of the world's largest CDN, Akamai. Our key empirical results are the following.

(1) *Comparison to baseline:* Shrink reduces energy use by 35% over a baseline scheme that provisions resources according to the peak demand while increasing the mean, 95-th %-ile and 99-th %-ile latencies by 8%, 3% and 15% respectively.

(2) *Comparison to ideal:* Shrink achieves a mean, 95-th %-ile and 99-th %-ile latency that is 15%, 3% and 25% higher than a scheme that characterizes the ideal energy-latency tradeoff while using 5% more energy than it.

(3) *Comparison of network energy use:* When one-fourth of the servers are active, Shrink's network energy use is lower than a network-unaware server consolidation scheme by 37% and 42% on the FatTree [17] and the VL2 [82] topology respectively.

Scope of the paper: This paper focuses on energy saving schemes based on server and network consolidation deployable at the scale of a single CDC. The following schemes, although related, are outside the scope of this paper: (1) Adapting global load balancing across CDCs to reduce the total energy use of all CDCs. (2) Making each server or switch power-proportional, e.g., server energy use can be reduced by turning off a fraction of its disks. (3) Optimizing the cost of energy used such as by accounting for time-of-day pricing for electricity; Section 6.5 discusses electricity costs.

6.1 Content datacenter background

The server to handle a particular end user's request is determined by a *load balancer*. Servers in a CDC have local storage used for caching content. When a request reaches a server, it serves the content from its cache, if the content is available there, or else, it may fetch the content from a *peer* cache in the same CDC.

Topology	Server count	Switches	Switch pow. / Server pow.
1 Gbps server link			
FatTree	3456	Cisco 2224TP (80W, 720 count)	0.17
VL2	2880	Cisco 2224TP (80W, 144 count), Cisco Nexus 5548P (390 W, 24 count)	0.08
10 Gbps server link			
FatTree	3456	Cisco Nexus 5548P (390 W, 360 count)	0.40
VL2	2304	Cisco 6001 (750W, 48 count), Cisco 6004 (2900W, 6 count)	0.23

Figure 6.1. Ratio of network to server energy use at typical operating conditions. Switch power use data from Cisco [44]. Server power use of Acer Altos T350 F2 at a load of 30% is 98.5 W [158].

If the requested content is unavailable on the CDC's servers, the request results in a *cache miss*. On a cache miss, content is fetched from remote locations, either from another CDC or from *origin servers*, at a cost of an increased latency, and served to the end user.

A CDC reduces user-perceived latency by avoiding load hotspots on servers and increasing cache hit rates, both with the help of its load balancer. Load balancer distributes request among servers to avoid load hotspots, which keeps server-load-dependent delays small. Further, it serves a content's requests from a small number of servers, which results in a small number of content replicas and increases overall cache hit rates.

CDCs should reduce their energy use because it can bring significant greenhouse emission reduction as well as cost savings to operators. Energy use is known to be 15-20% of the total cost of ownership of datacenters [81, 145, 87]. A CDC operator usually manages a global network of such CDCs, which could comprise of 100K servers or even more [16]. In a network with 100K servers with each server consuming 100 W, a 20% energy savings translates to 17520 MWh of yearly energy savings, which is equivalent to average annual energy use of 1616 homes in the US [14], and at a cost of 10c/KWh translates to $1.752 M in yearly cost savings.

Datacenters, including CDCs, should reduce their network energy use as it consumes a non-trivial fraction of datacenter energy. As shown in Figure 6.1, switches in the FatTree [17] and VL2 [82] toplogies, which support full bisection bandwidth, consume 17% and 8% of the server energy use at typical operating conditions in networks with a 1 Gbps server link capacity. In networks with a 10 Gbps server link capacity, which are gaining adoption, this fraction increases to 40% and 23% for FatTree and VL2 respectively.

Despite a large body of research on automated server and switch consolidation schemes showing significant potential for reducing energy use, the impact of consolidation on user-perceived latencies is not well understood. For example, a common problem approach in the literature [118, 84] is to focus on dynamically estimating spare resources, implicitly assuming that shutting down the spare resources will have little or no impact on latencies. However, in reality, it is rarely the case that one can, say, shut down 25 out of 400 servers with negligible latency impact. The precise impact depends on the workload dynamics and whether the application is constrained by compute, memory, disk, and/or network capacity. In the following section, we present a simple model that sheds light on these issues specifically in the context of CDCs.

6.2 Energy vs. user-perceived latency model

We present a simple analytical model in order to quantify the relationship between energy savings and user-perceived latency inflation in CDCs, and to gain a deeper understanding of the factors that determine this trade-off. The primary purpose of this model is expository, so we make a number of simplifying assumptions for ease of exposition first, and we progressively relax them in this and subsequent sections.

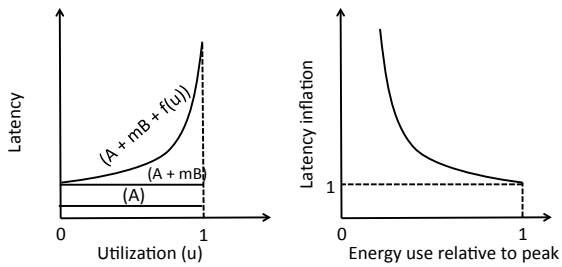


Figure 6.2. [Left] An illustrative server utilization vs. latency curve. [Right] Corresponding energy-latency curve.

6.2.1 Single server

Consider a single CDC server serving a workload of content requests from end users that is unchanging, i.e., the arrival rate, popularity distribution, and size distribution of content across requests is fixed. Let m denote the cache miss rate at the server, i.e., the fraction of requests for which the server contacts the origin server. Assume that the power drawn by a server $p(u)$ is a linear function of its utilization u, or the ratio of the incoming load and the server's capacity [118]. In this model, an idle server's power consumption is a fraction I of its peak power P, and the power consumed increases linearly from IP to P as the utilization increases from 0 to 1, i.e., $p(u) = (I + (1 - I)u)P$.

The end-to-end user-perceived latency is assumed to be the sum of three components as also illustrated by the three curves in Figure 6.2 (left): (1) Mean server latency $f(u)$, assumed to be a convex, increasing function of the utilization u ($0 \leq u \leq 1$); (2) Server-to-origin latency B, which is constant but incurred only upon a cache miss; (3) Client-to-server latency A, a constant. Thus, the total end-to-end user-perceived latency is $f(u)+mB+A$. As shown in the figure, realistic utilization vs. server latency profile are typically somewhere in between a straight line where the latency increases linearly with the utilization and an L-shaped curve where the latency is zero for all values of utilization less than 1 and is infinity at 1.

6.2.2 Datacenter as a single logical server

The simplistic model above can serve as a first-order approximation of a datacenter viewed as a single logical server as follows. Suppose the datacenter consists of N homogeneous servers, each identical to the one above. Suppose the total incoming load is uniformly distributed across all N servers resulting in a fixed utilization U at each server, which implies a power consumption of $(I+(1-I)U)P$ per server. Thus, the total power consumed is $(I+(1-I)U)PN$, and the mean latency is $f(U)+mB+A$. We have implicitly assumed here that the miss rate m remains the same as above even though each server is getting a sampled transformation of the original request distribution.

What is the impact of consolidating the datacenter to $n < N$ servers on the total power consumed and user-perceived latency? To answer the first part, we observe that the utilization at each server increases by a factor N/n, so its power consumption would increase to $(I+(1-I)UN/n)P$. Thus, the consolidated datacenter's total power consumption is given by $(nI+(1-I)UN)P$, and the corresponding energy use relative to the peak (the x-axis in Figure 6.2 (right)) is

$$\textit{benefit} = \frac{(nI+(1-I)UN)P}{(I+(1-I)U)PN} \tag{6.1}$$

Computing the corresponding end-to-end latency with the consolidation as above is nontrivial as it requires us to account also for the increase in cache miss rates. Assuming that shutting down servers results in a proportional decrease by a factor N/n in the total available storage (an assumption that is natural for clusters of commodity PCs–the more common option in practice–but not for CDCs relying on network storage), we need to compute the mean miss rate m' that in general would be lower than m. In our numerical examples below, we derive m' using a characteristic-time approximation model for an LRU cache [40]. The latency is given by $f(UN/n)+m'B+A$, and the corresponding latency inflation is given by

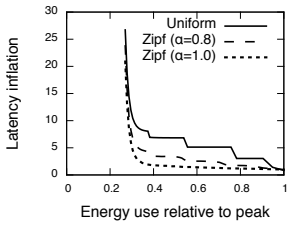


Figure 6.3. Energy-latency tradeoff for workloads with varying Zipf exponents.

$$cost = \frac{(f(UN/n) + mB + A)}{(f(U) + m'B + A)}. \tag{6.2}$$

We make two observations based on the expression for the latency inflation above. First, the more skewed (closer to L-shaped as opposed to linear) the server's utilization vs. latency profile is, the less noticeable the impact on latency as $f(UN/n) \approx f(U)$ unless $UN/n \to 1$. Second, the more skewed (e.g., a high Zipf exponent) the popularity distribution is, the less noticeable the impact on latency as $m' \approx m$ assuming that the consolidated storage also suffices to cache the small fraction of popular objects contributing to the overwhelming portion of hits. We numerically exemplify this second insight next.

Numerical example: We evaluate the evaluate the energy-latency tradeoff for Zipfian content popularity distributions. We assume that the server's utilization vs. latency profile is given by $f(u) = Cu^K$, $K >= 1$ is a model parameter and C is a constant latency. Other model parameters are as follows: each server has a capacity = 1 request/sec, total load L = 15 requests/sec, latency from clients to servers A = 10 ms, latency from servers to origin servers B = 100 ms, server latency coefficient C = 400 ms, idle power fraction I = 0.5, total number of unit-sized content M = 100 million, storage per server S = 1 million, number of servers N = 100.

Figure 6.3 presents results for workloads with Zipf exponent α = 1.0, 0.8 and 0; $\alpha = 0$ results in a uniform distribution. The energy use is shown relative to the peak energy use with N = 100 active servers, and latency inflation is shown relative

to the least latency in the graph. The server latency model parameter is $K = 9$ for all workloads which implies that the difference in latencies among them is due to a difference in miss rates. We observe that workloads with higher Zipf exponents achieve a better energy-latency tradeoff. The reason is that a higher Zipf exponent means that most of the hits result from a small fraction of objects, so reducing the available storage only results in a small increase in miss rates.

Summary: Our simple expository model suggests that the energy vs. latency tradeoff is more favorable as the skew in the server's utilization vs. latency profile increases and the skew in the content popularity distribution increases. Admittedly, this simplistic model has several critical limitations as it (1) does not consider workload dynamics, (2) assumes a perfect load balance among servers, (3) ignores the overhead of coordination between servers, and (4) implicitly assumes that the utilization vs. latency behavior and the cache size vs. miss rate behavior can be approximated by treating the CDC as a single logical server of the same total capacity, ignoring the network fabric entirely.

Thus, a natural question is what do real, achievable energy vs. latency inflation tradeoffs look like in CDCs? Further, does an increase server load or an increase in cache miss rates cause a greater impact on latency? To answer these questions, we present the design and implementation of Shrink and evaluate the tradeoff using a real workload from a large CDC in the next two sections.

6.3 Shrink design and implementation

Shrink is a system for server and network consolidation as well as load balancing. Shrink reduces a CDC's energy use, while enabling operators to achieve the desired latencies and hardware reliability. A novel aspect of Shrink is its network-aware server consolidation scheme, which increases the energy savings that Shrink's

network consolidation scheme achieves. This section describes Shrink's design goals, its design and its implementation.

6.3.1 Design goals

Shrink balances the following three requirements that are important from the perspective of a CDC operator.

(1) **Energy use:** The design should minimize the energy use of a CDC's servers and switches.

(2) **User-perceived latency:** The design should enable operators to provide the desired latency to end users.

(3) **Hardware reliability:** The design should enable operators to achieve the desired hardware reliability, where reliability is quantified by the rate of on-off transitions for servers and switches. Note that a decrease in hardware lifetime due to frequent on-off transitions would result in the CDC operator incurring greater capital expenditure over time. For example, if a hard disk rated at 50K start/stop cycles for reliable operation makes one on-off transition per hour, it reaches the start/stop cycles limit in 50K hours, much before its specified mean-time-before-failure of 1.2M hours [154].

6.3.2 System overview

Shrink runs as a control program that executes at fixed length intervals. In each interval, the control program receives reports from all active servers regarding their load at the granularity of *content buckets*; content are mapped to buckets using a consistent hash function based on their name. Based on three inputs – server load reports, a utilization vs. end-user latency curve for the operator's latency metric of interest (such as a given percentile of latency) and a specified value of end-user latency as per that metric – , the control program computes the following two outputs: (1) **Consolidation:** The set of servers and switches to keep

active, while remaining servers and switches are turned off. (2) **Load balancing:** A *traffic split* for each bucket describing the ratio in which the bucket's requests are split among servers. We next describe Shrink's consolidation and load balancing algorithms, followed by how to compute the utilization vs. end-user latency curve provided to Shrink.

6.3.3 Consolidation

Shrink's consolidation algorithm assumes a datacenter topology in the form of a multi-rooted tree in which a topological ordering of nodes from left to right is well-defined at each level in the topology. Servers and switches reside at leaf and non-leaf nodes respectively; root nodes provide external connectivity. To adapt network routing in response to network consolidation, Shrink requires the support for ECMP [91], which is commonly available in the datacenter network fabrics today.

6.3.3.1 Server consolidation

Shrink selects the active servers, in a network-aware manner, to be the leftmost leaf nodes in a topology. Figure 6.4 shows why such a network-aware server consolidation can increase the energy savings that network consolidation achieves. In the figure, both the left and the right topologies have 20 active servers. The right topology requires all ToR switches to be kept active, but in the left topology the set of active servers are chosen among servers in one rack, which allows ToR switches in other racks to be turned off. This example shows that consolidating servers in a network-aware manner can reduce network energy use.

Let $F(u)$ be the utilization vs. end-user latency curve for the operator's latency metric of interest provided to Shrink. $F(u)$ is similar to the end-to-end latency $(f(u) + mB + A)$ described in Section 6.2. We discuss how $F(u)$ can be obtained in Section 6.3.5. Shrink uses $F(u)$ assuming that if the servers active in the datacenter

are run at an average utilization u, then the resulting latency is $F(u)$. Thus, to achieve a specified end-user latency R, Shrink chooses a server utilization limit $U = r^{-1}(R)$.

Shrink computes the number of active servers using two integer values: *minServers* and *maxServers*. *minServers* $= \lceil L/U \rceil$, where the L is the predicted load for the next interval; L is predicted using linear regression based on the total load across servers in the past few intervals (default = 10 intervals). *maxServers* is the maximum value of *minServers* over the previous time window of length W; W is called the *pre-shutdown wait interval*. If *minServers* $> n$, where n is the current number of active servers, $(\textit{minServers} - n)$ more servers are turned on, else if *maxServers* $< n$, $(n - \textit{maxServers})$ servers are turned off.

Reliability of servers: The reliability of servers depends on the pre-shutdown wait interval (W). With a smaller W, Shrink turns servers off sooner and saves more energy but potentially increases the rate of on-off transitions because of spurious decreases in load. Our empirical results (Section 6.4.2.4) show that W close to 1 hr is a sweet spot for which energy use is 15% higher than that achievable for a small value of W = 1 min, and whose on-off transition rate is nearly 10$\times$ lower than that of W = 1 min. Thus, we expect the default value of W = 1 hr to work well for operators. However, operators can better inform their choice of W with similar tests for their workload.

6.3.3.2 Network consolidation

Network consolidation in Shrink satisfies the following condition: assuming ECMP routing, all active servers can simultaneously send traffic to clients equal to the *external bandwidth* using the set of active switches only. We define the external bandwidth as the outgoing link capacity of a server divided by the oversubscription ratio of the datacenter. For example, in a network with 2:1 oversubscription

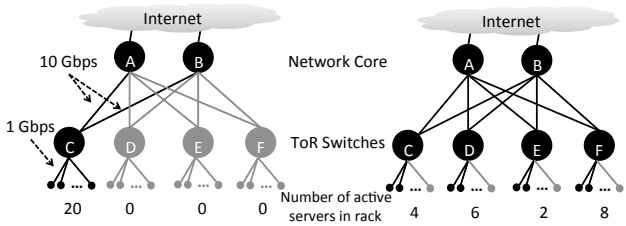


Figure 6.4. Black (grey) components are turned on (off). Each rack has 20 servers. (Right) Randomly choosing the set of active servers results in all ToR switches being turned on. (Left) Choosing the same number of active servers in a "network-aware" manner enables more ToR switches to be turned off.

and 1 Gbps server outgoing links, the external bandwidth is 500 Mbps. If a server sends traffic at a maximum rate up to the external bandwidth, then this condition avoids network bottlenecks.

We explain the above observation using an example. Consider the topology in Figure 6.4 (left) in which the external bandwidth is 1 Gbps. Let us assume that only servers connected to switch C are active due to consolidation. Let the maximum rate of traffic from a server to clients be 800 Mbps. If all active servers are sending traffic at their maximum rate, then the total traffic from switch C to core switches A & B is (800 Mbps)$\times$20 = 16 Gbps. If both A & B are active, the condition above is satisfied, and all servers can simultaneously send traffic to clients at their maximum rate. If A is on but B is off, the condition above is not satisfied, and as a result, a network bottleneck happens on link AC.

Shrink's network consolidation assumes that each active server is sending traffic to clients at a rate equal to the external bandwidth. It selects the set of switches needed to route this traffic to the root nodes. It considers switches in the order of increasing height from leaf to root and among switches at the same height considers them in a left-to-right order. Each switches selects p-leftmost parents that must be active to forward the traffic sent by its children to root nodes; p is the least value for which the sum of capacities of links to the selected parent nodes is equal or more than than the traffic sent by the switch's child nodes. Due to ECMP support,

traffic forwarded to root nodes is divided on links to the p parent nodes equally. Finally, only the switches that are transmitting non-zero traffic are selected to be active.

Reliability of switches: Network consolidation in Shrink satisfies the following property related to reliability of switches: if the number of active servers increases, additional servers and switches are turned on, but none of the servers and switches that are already active are turned off. Due to this property, the rate of on-off transitions of switches is close to that of on-off transitions for servers. For example, consider a time interval in which the number of active servers is monotonically increased from 1 to N, where N is the total number of servers. In this interval, there is at most one transition from off to on state for each server. Due to the above property, the set of active switches can only add new switches as the number of active servers increases. As a result, each switch would also have at most one transition from off to on state in this interval. Thus, the above property ensures that improving server reliability improves switch reliability as a consequence.

6.3.4 Load balancing

Shrink uses randomized load balancing over a set of content buckets. Content is mapped to a fixed number of buckets (default = 100) using a consistent hash function based on its name. The output of load balancing is a *traffic split* for each bucket that determines the ratios in which the bucket's requests will be distributed among CDC's servers. Traffic split for a bucket is determined based on the load predicted for the bucket in the next interval. Shrink predicts a bucket's load using a linear regression model trained based on the observed load for the bucket in the past few intervals.

input : *buckets // set of content buckets*
: *bktsLoad // content bucket to load map*
: *srvrs // set of active servers*
: *U // server utilization limit*
: *k // default server count serving a content bucket*
: *trSplits // bucket to traffic split map (old)*
output: *newTrSplits // bucket to traffic split map (new)*
limit= max($\frac{\sum_{bkt \in buckets} bktsLoad[bkt]}{|srvrs|}$, *U*) *// server load limit*
srvrsLoad = {} *// server to load map*
for *srvr*∈ *srvrs* **do**
  srvrsLoad[*srvr*] = 0
end
newTrSplitBkts = {} *// buckets needing new traffic splits*
for *bkt* ∈ *buckets* **do**
  for $(s_j^{bkt}, f_j^{bkt}) \in trSplits[bkt]$ **do**
    // server s_j^{bkt} *serves a fraction* f_j^{bkt} *of bkt's load*
    if $s_j^{bkt} \notin srvrs$ *or* $srvrsLoad[s_j^{bkt}] + f_j^{bkt} bktsLoad[bkt] > limit$ **then**
      Add *bkt* to *newTrSplitBkts*
      break
    end
  end
  if $bkt \notin newTrSplitBkts$ **then**
    for $(s_j^{bkt}, f_j^{bkt}) \in trSplits[bkt]$ **do**
      $srvrsLoad[s_j^{bkt}] += f_j^{bkt} bktsLoad[bkt]$
    end
    $trSplits[bkt] = newTrSplits[bkt]$
  end
end
for $bkt \in newTrSplitBkts$ **do**
  select *k* servers $(s_1, s_2, ...s_k)$ randomly from *srvrs*
  bktTrSplit= {} *// new traffic split for bkt*
  remBktLoad = *bktsLoad*[*bkt*] *// remaining load for bkt*
  for $srvr \in (s_1, ...s_k)$ **do**
    if $srvrsLoad[srvr] + bktsLoad[bkt]/k < limit$ **then**
      $bktTrSplit = bktTrSplit \cup \{(srvr, 1/k)\}$
      $remBktLoad -= bktsLoad[bkt]/k$
    end
  end
  while $remBktLoad > 0$ **do**
    select *minLoadSrvr* whose *srvrsLoad*[*minLoadSrvr*] is the least
    load = min(*remBktLoad*, $limit - srvrsLoad[minLoadSrvr]$)
    $remBktLoad -= load$
    Add $\{(minLoadSrvr, \frac{load}{bktsLoad[bkt]})\}$ to *bktTrSplit*
  end
  newTrSplits[*bkt*] = *bktTrSplit*
end

Algorithm 1: Load balancing

Shrink's load balancing (Algorithm 1) selects a load limit for servers, which is greater of the server utilization limit U or the average load on an active server

(line 1). The algorithm selects buckets that need new traffic splits compared to the previous execution of the algorithm. These buckets either belong to servers that are no longer active or whose load exceeds the load limit (lines 5-15). To compute new traffic splits for the selected buckets, Shrink selects a fixed number of randomly chosen servers (default = 2) and divides the load for the bucket among them equally. In doing so, if the load on any server exceeds the load limit then Shrink assigns the excess load for that bucket from that server to the least loaded server that is active (lines 16-29).

Load balancing mitigates disruptions to the existing connections by not sending requests to a server that is likely to be turned off. To this end, Shrink's node selection module informs the load balancing module that a server is likely to be turned off in the middle of the pre-shutdown wait interval of the server. As the pre-shutdown wait interval is tens of minutes long, existing connections, except for the long transfers, complete before server shutdown. To mask the server turnoff for these few long transfers, CDCs can use techniques such as IP address takeover [66] or connection migration across machines [157, 163]. We have not implemented these techniques in Shrink.

6.3.5 Computing utilization vs. latency curve

We discuss three ways of computing the utilization vs. latency curve $F(u)$ input to Shrink in the order of increasing complexity. The more complex schemes potentially enable Shrink to provide latencies that are closer to those specified by operators.

The first and second approaches require prior measurements of a single server and of the entire CDC respectively. These measurements obtain the value of the operator's latency metric of interest at varying levels of utilization. These measurements are done with a representative workload in a test environment with real or

emulated client-to-server delay and server-to-origin delay. The first approach implicitly assumes that if the latency of a single server at utilization u is r, then the latency of the CDC whose servers have an average utilization u is also r. This assumption ignores the inefficiencies of a distributed system such as imperfect load balancing and coordination overhead among peer caches in a CDC. These factors are more accurately accounted for in the second approach. Nonetheless, both approaches could be inaccurate as they do not consider the non-steady cache behavior due to on-off transitions and increased cache miss rates due to reduced storage. To account for these inaccuracies, a constant inefficiency factor ρ is added as follows. If $F'(u)$ is the measured utilization vs. latency curve, then Shrink is input a curve $F(u) = \rho \times F'(u)$.

The third approach is to equip Shrink to learn the curve automatically based on online measurements. The overhead of these measurements can be kept small by doing them for a small fraction of requests only. This approach could be more accurate than the previous two approaches because it considers the factors discussed above and can even adapt to changes in workload characteristics that influence the utilization vs. latency behavior. We have not implemented this third approach in Shrink.

6.3.6 Implementation

Our Shrink prototype is implemented in nearly 6K lines of Java code. By default, Shrink's control program runs at 20 sec intervals. Shrink uses Squid as a caching proxy [54]. We configured Squid to use its AUFS storage, which handles disk accesses asynchronously by multiple threads; AUFS results in lower response over synchronous disk access storage mechanisms. We have added support for content chunking in order to reduce origin traffic and improve cache hit rates:

specifically, a large file is requested as a sequence of 2 MB chunks, which enables Squid to cache each chunk independently.

Load balancing: Shrink resembles a DNS load balancer. Similar to a DNS resolution, a client contacts Shrink to receive the traffic split for a content, and then the client contacts a server as per the traffic split to download the content. An alternative would have been to implement Shrink as a middlebox. As a middlebox routes the traffic between clients and servers through it, it would have become a network bottleneck on our testbed. In comparison, a DNS-type load balancer does not become a network bottleneck as it only exchanges load balancing queries and responses from clients.

Server load measurement: We measure a server's load in terms of request rates. We use an approach that requires no modification to the Squid codebase. We run a process on each server that parses the content access logs output by Squid, and sends to Shrink, once every interval, the number of requests for each content bucket at that server in the previous interval.

Peer caching overhead: Each Squid instance runs as an independent cache that fetches content from origin servers upon a cache miss. We tested a configuration of Squid in which all servers were configured as cache peers that used Internet Cache Protocol (ICP) [174] for querying peer caches upon a cache miss. But, we found the overhead of ICP to be nearly 25% of the overall request load while the peer cache hits were less than 3%. Hence, we disabled ICP due to its lower benefit compared to its overhead.

A limitation of our prototype is the lack of power controls for servers and switches. We emulate the startup (shutdown) delay of servers by waiting for a pre-defined interval before restarting (killing) the server-side processes. Finally, we note that remote power management that is necessary for turning servers and switches on and off is available from multiple vendors today [50, 134].

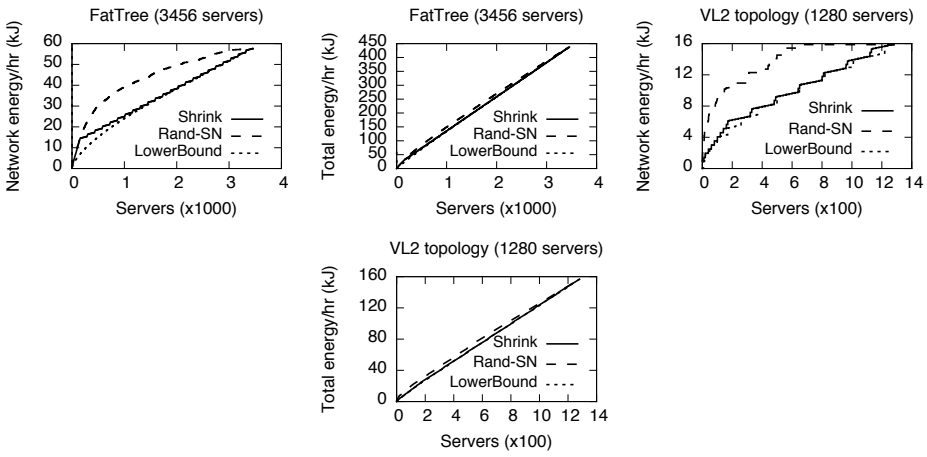


Figure 6.5. [Numerical computation] Shrink's network energy use is lower than a network-unaware server consolidation scheme Rand-SN by 38% on FatTree and 42% on VL2 when one-fourth of the servers are active in each topology.

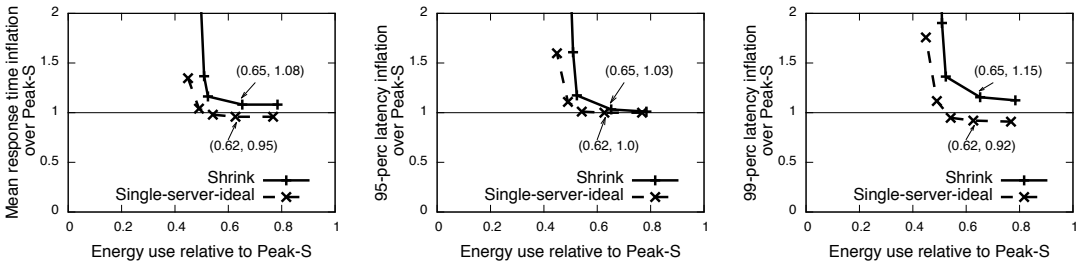
6.4 Experimental evaluation

Our evaluation has two main goals: (1) Comparing the network energy use of Shrink against a network-unaware server consolidation scheme (Section 6.4.1). (2) Quantifying the energy-latency tradeoff achieved by Shrink and compare it to the ideal energy latency tradeoff achievable (Section 6.4.2).

6.4.1 Comparing network energy use

Schemes compared: (1) *Rand-SN:* Rand-SN selects the set of active servers randomly; it uses the same network consolidation scheme as Shrink. Rand-SN is used to evaluate the benefit of network-aware server consolidation in Shrink. (2) *LowerBound:* We define lower bounds on the network energy use for a given number of active servers on the FatTree and the VL2 topologies. Our computation of LowerBound for FatTree and VL2 is described in Appendix B. Rand-SN and LowerBound provide the same per-server bandwidth guarantee to external hosts as Shrink does.

Topologies: We simulate two network topologies: a 3456-server FatTree topology made of 24-port switches (Cisco Nexus 2224P, 80 Watt, 720 count) [44] con-



(a) Mean (b) 95-th percentile (c) 99-th percentile

Figure 6.6. Server consolidation (Section 6.4.2.2): Shrink's reduces energy over Peak-S with a small latency inflation. In Figure 6.6(a), Shrink's energy use is 0.65× of Peak-S and its mean latency is 1.08× of Peak-S; Single-server-ideal's energy use is 0.62× of Peak-S and its mean latency is 0.95× of Peak-S.

suming 80 Watt per switch, and a 1280-server VL2 topology made of 24-port ToR switches (Cisco Nexus 2224P, 80 W, 64 count) [44] and 16-port 10 Gigabit core or aggregation switches (Cisco Catalyst 6500, 480 W, 24 count) [43]. We assume all active servers have identical power use (Acer Altos T350 F2, 130W at 60% utilization [158]).

Results: Figure 6.5 presents our results. The relative difference between Shrink and Rand-SN reduces as the number of active servers increases. When 25% and 50% of servers are active, Shrink's network energy use is lower than Rand-SN by 38% and 26% respectively on FatTree and 42% and 35% respectively on VL2. When 25% and 50% of servers are active, Shrink's network energy use higher than LowerBound by 9% and 2% respectively on FatTree and by 13% and 7% respectively on VL2. These findings show that Shrink's network-aware server consolidation reduces the network energy use over network-unaware server consolidation schemes and gives network energy savings close to the lower bound. When 25% and 50% of servers are active, Shrink's *total* energy use is lower than Rand-SN by 9% and 5% respectively on FatTree and 10% and 7% respectively on VL2. Thus, Shrink's network-aware server consolidation is effective in reducing aggregate CDC energy use as well.

6.4.2 Quantifying energy-latency tradeoff

6.4.2.1 Experiment setup

Akamai dataset: Our evaluation uses content access traces from an Akamai datacenter. The traces include all requests received at a datacenter with 24 servers for a week in December 2013. We restricted our data collection to a small datacenter as we did not have the resources to experiment with traces from a significantly larger datacenter. Our anonymized traces include several major types of traffic observed in a CDN such as 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.

Testbeds: We use prototype-based experiments (on EC2 and Emulab) and trace-based experiments. Our experiment on EC2 evaluates the energy-latency tradeoff due to server-only consolidation. As we do not have control over network topology on EC2, we use Emulab to evaluate the latency inflation due to both server and network consolidation. Finally, we conduct larger-scale trace-based experiments on a simulator.

Schemes compared: We compare Shrink against *Peak-S* and *Single-server-ideal*. Peak-S represents a baseline in which a CDC operator does not use consolidation to reduce energy use, i.e., it keeps all servers and switches active.

Single-server-ideal is a computation of the ideal energy-latency curve, unachievable by any real system. The points on this curve are obtained by varying the utilization u up to which any server can be loaded. For a given u, the latency of Single-server-ideal for a given metric is equal to the measured load-vs.-latency curve of a single server for the same metric at the same utilization. For the same utilization u, any distributed system will have a higher latency because workload

dynamics, load imbalance and non-steady state cache behavior; these factors are ignored by Single-server-ideal. Our single server measurements are done with a representative workload in the respective experimental environment, EC2 or Emulab, with emulated client-to-server delay and server-to-origin delay.

For Single-server-ideal, we compute the set of servers and switches in each time interval that minimizes the total energy use as follows. Based on four inputs – u, the total load in each time interval, the number of server transitions allowed, and the power model of each server –, we use a dynamic programming algorithm to compute the total energy use of servers and the number of active servers in each time interval. The number of transitions is equal to one on-off server transition/server/day or the same number of transitions as Shrink in that experiment, whichever is higher. The ideal network energy use for the tree topology we experiment with (Section 6.4.2.3) is computed based on the number of active servers in each time interval. The set of active servers are selected in a left-to-right order; for each active server, we select the switches on the path to the root. The set of switches selected across all active servers is the set that optimizes network energy use.

6.4.2.2 Prototype-based experiments: server consolidation

This experiment quantifies the energy-latency tradeoff due to server consolidation on EC2. Our EC2 testbed consists of 15 servers, 15 clients and 4 origin servers running on independent m3.xlarge instances (4 core, 15 GB RAM, 40 GB$\times$2 SSD), all in the same datacenter. Our origin server is a trivial Apache Tomcat application that dynamically generates the requested content. We emulate a 60 ms RTT between origin servers and CDC's servers, and a 10 ms RTT between client and server machines. We configured each server to use an 8 GB memory cache and a 30 GB cache on each SSD.

Our workload consists of a 24-hour duration of the trace. We selected one-eighth of the content randomly from the trace but sped up the trace by $8\times$ to send those requests over a 3-hour duration. Thus, we maintain approximately the same load on the servers. We use a short pre-shutdown wait interval $W = 10$ min for Shrink because our workload is a sped up by $8\times$.

We calculate the energy savings relative to Peak-S as per Equation 6.1; the ratio of the idle to peak energy use of servers, I equals 0.5 [31]. Peak-S uses 15 servers in this experiment. We have conservatively chosen the number of servers in Peak-S to be much less than the number of servers in the Akamai datacenter itself so as not to overestimate the energy savings.

We evaluate Shrink in terms of three latency metrics – mean, 95-percentile, 99-percentile. To provide a utilization-vs.-latency curve $F(.)$ to Shrink for each metric, we take the first approach discussed in Section 6.3.5. The function $F(.)$ for each metric is equal to the measured utilization vs. latency curve of a single server with an inefficiency factor $\rho = 1.2$. Based on $F(.)$ for each metric, we specify latencies $F(u)$ to Shrink for values of u from 0.375 to 0.875 at intervals of 0.125 across different runs.

Figure 6.6 compares the latency and the energy use of Shrink relative to Peak-S for the mean, the 95-th percentile and the 99-th percentile of latencies. Shrink reduces energy use by 35% over Peak-S while inflating the mean, the 95-th percentile and the 99-th percentile by 8%, 3% and 15% respectively. To explain the difference between Peak-S and Shrink, consider Figure 6.7 (left) which shows the aggregate load and the number of servers from one of the runs of the system. Shrink adapts the number of active servers based on load in the system keeping only 3 servers active when the load is the lowest, but Peak-S always keeps 15 servers active and hence has a higher energy use. This result implies that an operator for which these

inflations are tolerable, e.g., they do not cause an SLA violation, can achieve the corresponding energy savings as well.

Does an increase server load or a decrease in cache hit rates cause a greater impact on Shrink's latency over Peak-S? In Figure 6.7 (right) the x-axis shows the server utilization limit U computed by Shrink's consolidation algorithm (Section 6.3.3.1) and y-axes show corresponding the hit rates and the mean latency of Shrink. Shrink's hit rates are lower than Peak-S but the decrease is less than 7% across all utilizations. Thus, the latency inflation due to a decrease in hit rates is likely to be small. A small reduction in hit rates is not surprising given that the Zipf exponent for the Akamai trace is 0.8 as per our calculations, and our model in Section 6.2 has suggested that consolidation reduces hit rates by a small fraction for for real workloads with a high skew in content popularity. We find that mean latencies increase sharply at a high server utilization limit, e.g. $U = 0.875$, which is likely due to an increase in server load. To summarize, there is a small latency inflation due to a decrease in hit rates but severe inflation occurs at high server utilization limits, and is likely due to an increase in server load.

Comparing Shrink with Single-server-ideal, in Figure 6.6(a), Shrink's energy use is 0.65$\times$ of Peak-S and its mean latency is 1.08$\times$ of Peak-S; Single-server-ideal's energy use is 0.62$\times$ of Peak-S and its mean latency is 0.95$\times$ of Peak-S. There are two reasons that explain the gap between Shrink and Single-server-ideal. First, Single-server-ideal ignores several factors that increase latency of any distributed system such as workload dynamics, load imbalance and non-steady state cache behavior. Second, Shrink waits for the pre-shutdown wait interval to see if a decrease in load persists before turning servers off. But, in our calculation, Single-server-ideal knows the load for the entire experiment beforehand and hence it can shutdown servers sooner than Shrink and save more energy.

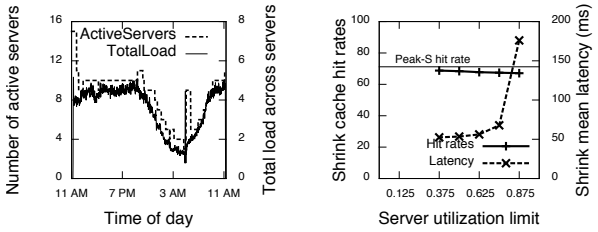


Figure 6.7. Server consolidation (Section 6.4.2.2): [Left] Shrink adapts the number of active servers based on CDC load to reduce energy over Peak-S. [Right] Cache hit rates and mean latency for Shrink and Peak-S.

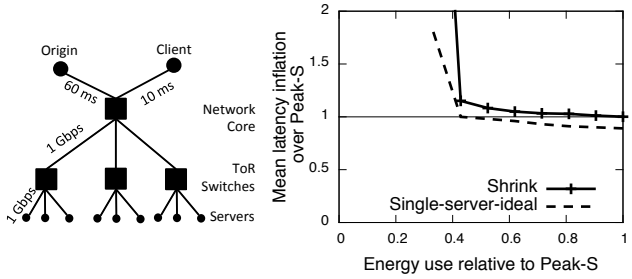


Figure 6.8. Network and server consolidation (Section 6.4.2.3): [Left] Emulab topology for the experiment. [Right] Compared to Peak-S, Shrink has a 15% higher latency but a 57% lower energy use.

6.4.2.3 Prototype-based experiments: server & network consolidation

We use Emulab to evaluate the energy-latency tradeoff when both server and network consolidation are being performed. For our experiment, we configure a tree topology with 1 Gbps links as shown in Figure 6.8 (left). In this topology, the ToR switches have 4 ports, and the core switch has at least 4 ports. Accordingly, we calculate energy use of switches based on the power use of 4-port switch (Netgear GS105), which is 14.4 W [123]. The energy use of servers is computed using the same function as in the previous experiment. Our workload consists of a 1-hour duration of the trace containing requests for one-eighth of the content selected randomly.

Figure 6.8 (right) compares schemes in terms of the mean latencies. Across different runs that vary specified mean latencies, Shrink uses between 2 and 9 servers;

Peak-S uses 9 servers. We discuss the case when Shrink uses 3 servers so that only one of the ToR switches are being used as a result of network consolidation. In this case, the peak utilization of the link between the ToR and the core switch increased up to 76% during the experiment, which is nearly three times higher than the peak link utilization for Peak-S. Despite this increase, Shrink's latency is only 15% higher than Peak-S, while its network energy use is 50% lower and the overall energy use is 57% lower than Peak-S (second point from the left in Figure 6.8 (right)). This result shows that both network and server consolidation can be performed with a small performance impact in CDCs. Finally, we note that the difference between Single-server-ideal and Shrink is consistent with the difference between them in our experiment with server-only consolidation,, e.g., for the same energy savings as Shrink (= 57%), Shrink's latency is 15% more than Single-server-ideal.

6.4.2.4 Trace-based experiments

Methodology: We conduct experiments for a CDC with 16 servers for the week-long Akamai trace. The capacity of each server is defined in terms of network traffic it can support. The rate of network traffic generated by a request is a constant equal to the client bandwidth reported in the Akamai trace. To be able to fit the simulator process in the memory on our machine (32 GB), we filtered requests for one-eighth of the content from the trace. Accordingly, we scale down the capacity of each server to be 150 Mbps, and the cache size per server to be 150 GB. Since trace-based experiments do not provide an accurate estimate of latencies, we used a fixed server utilization limit $U = 0.65$ for our experiments, which is expected to cause a small latency inflation (Figure 6.7 (right)). The cache hit rates of our simulator's LRU caching and Squid differ by less than 2% for the same workload and cache size.

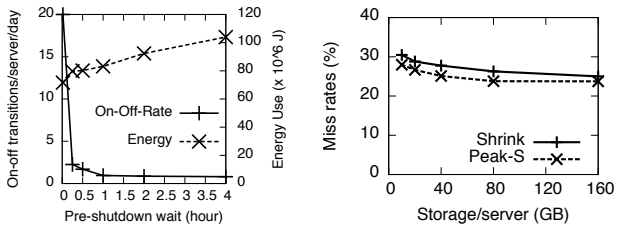


Figure 6.9. [Simulator] [Left] Pre-shutdown wait interval between 30 min & 1 hour keeps on-off transition rate close to 1/server/day. [Right] Comparison of miss rates for varying amounts of storage.

Impact on hardware reliability: Figure 6.9 (left) shows the rate of on-off transitions per server and the corresponding energy savings is achievable. We find that a short pre-shutdown wait interval W = 1 min hurts hardware reliability by increasing the rate of server transitions to more than 20/server/day. On the other hand, a high W = 4 hours reduces server transition rate to 0.81/server/day, but increases energy use by nearly 45% over W = 1 min. The sweet spot for pre-shutdown wait interval is between 30 min to 1 hour, where increase in energy use over W = 1 min is between 12% and 15% but most of the reduction in on-off transition rates can still be achieved.

Storage vs. cache miss rates: To determine the sensitivity of miss rates to available storage, we evaluate Peak-S and Shrink for varying amount of storage from 10 GB/server to 160 GB/server and present results in Figure 6.9 (right). We remark that we have scaled down the CDN trace and hence the storage by a 8$\times$ factor, i.e., we would have provisioned 1.28 TB storage instead of 160 GB if we were to experiment with the full trace. As storage reduces, the miss rates increase as expected. But, the relative difference between miss rates for both Shrink and Peak-S remains nearly the same even on reducing the storage to 10 GB, e.g., Shrink's miss rates are 10.3% higher than that of Peak-S for 160 GB storage and are 9.6% higher than that of Peak-S for 10 GB storage. Thus, we conclude that server consolidation schemes

are expected to increase datacenter miss rates by a small fraction within the range of storage typically available on server-class machines.

Cooperative caching benefit: We evaluate the potential benefit of cooperative caching among servers in a CDC assuming that a cache-coordination protocol with a much smaller overhead can be developed in future. The hit rates at cache peers are 1.65% for Peak-S and 3.02% for Shrink, which suggests that cooperative caching among datacenter servers, if implemented efficiently, could reduce the impact of energy optimization schemes by a small margin.

6.5 Discussion

Energy use vs. energy cost: There are three types of CDCs in terms of their energy cost to an operator.

(1) Operator-owned facility: If a CDC operator owns the datacenter facility, it directly pays to the electricity companies based on its usage. In such datacenters, a reduction in energy use by Shrink is likely to bring a reduction in electricity costs as well.

(2) Co-location facility: A CDC at a co-location facility typically pays by the provisioned power and not the electricity used [141]. Therefore, a reduction in energy use will not bring cost savings to CDC operator with the existing pricing models. However, it is possible a CDC operator may use the reduced energy as a leverage for negotiating a cheaper pricing.

(3) Co-location inside ISP networks: A CDC at a co-location facility maintained by an ISP often has a symbiotic relation with the ISP, where the CDC caches content to reduce the inter-domain traffic for the ISP while an ISP provides co-location free of charge [79]. In such CDCs, energy savings do not translate to cost savings to the CDC operator. Although, energy savings do benefit the ISP, who eventually pays for the electricity.

The type of usage-based energy pricing also determines the cost savings for an operator. Specifically, we distinguish between flat rate pricing and time-of-use pricing [137]. With a flat rate pricing, a given percentage reduction in the energy use results in the same percentage reduction in the energy cost. With a time-of-use pricing, the percentage reduction in the energy use and the energy cost may not be the same. For example, if the peak load on a CDC coincides with the peak hour of electricity prices, the percentage reduction in the energy cost would be lower than the percentage reduction in the energy use.

Impact on web-page load time: Our prototype-based experiments evaluate the latency for individual HTTP requests, and hence do not capture a key metric that is more relevant from an end-user's perspective: web-page load time. However, we expect the inflation in web-page load time to be lower than the inflation in latencies given that computation in web browsers constitutes up to 35% of the critical path of a web-page load time [172].

6.6 Related work

Our effort distinguishes from prior work in quantifying the energy-latency tradeoff in CDCs, presenting the design and implementation of a system to leverage this tradeoff and proposing a network-aware server consolidation scheme to reduce network energy use. Prior work on reducing energy of datacenters can be divided into three topics: (1) power-proportionality of servers and switches. (2) server and network consolidation in a datacenter and (3) global load balancing across datacenters.

Power-proportional servers and switches: Several efforts have focused on reducing energy use of a server's sub-systems such as CPU [176], disk [115], and memory [63]. Similarly, Nedevschi et al. [122] study power management for switches that support sleep states or several power/performance states similar to

CPUs. Nonetheless, today's servers and switches are far from power-proportional. Mahadevan et al. show that networking equipment consumes 62%-91% of their peak energy in idle state [117] and servers consume 32% to 42% of maximum power at a small utilization of 10% [158]. Until the ideal of power-proportionality is achieved, consolidation remains a promising approach to save energy.

Server consolidation: Given the long line of work in server consolidation, our work does not focus on saving more energy than the existing consolidation schemes, but instead on accurately quantifying the impact on latencies of a simple consolidation scheme.

The analytical work in this area shares similar goals as us. Lin et al. [111] propose an algorithm for optimizing a cost metric that incorporates energy costs, on-off switching costs and cost of degradation in performance. Mathew et al. [118] propose an algorithm that balances energy use, reliability and availability of servers, which they evaluate based on load traces from Akamai datacenters. In comparison, our implementation-based approach enables us to accurately model the relation between server utilization and latency, impact of server consolidation on cache hit rates, and non-ideal load balancing, to accurately quantify the impact of consolidation on latency for CDCs.

Several efforts have conducted an implementation-based evaluation of server consolidation for stateless systems. Chase et al. [39] allocate resources among multiple co-hosted services in a cluster while reducing energy via consolidation. Pinheiro's [138] system proposes consolidation and load balancing algorithms given a bound on the performance degradation that is acceptable. Rajamani et al. [142] evaluate consolidation schemes for a modified TPC-W workload. In comparison, our effort focuses on CDCs that maintain a large amount of state in the form of cached content. In CDCs, the effect of consolidation on latencies cannot be evalu-

ated accurately without accounting for the effect of consolidation on the availability of cached content and the resulting cache hit rates.

Trushkowsky et al. [165] dynamically allocate servers and reconfigure the data stored on the servers to meet service-level objectives such as 99-th percentile request latency. However, there are two key differences between their work and ours. First, they focus on a workload exclusively of small (256B) objects stored in-memory, whereas CDCs need to deal with orders of magnitude of heterogeneity in object sizes and extensively use a disk cache to improve hit rates. Second, their system appears to be a backend data store, which always has content available within the datacenter. In comparison, CDCs have a significant fraction of traffic to remote datacenters due to cache misses, and the impact of consolidation on latency in CDCs depends on the increase in traffic to remote datacenters that consolidation causes. For these reasons, it is not clear if their findings on the impact of dynamic server allocation on request latency would be applicable for CDCs.

Network consolidation: Network consolidation has been studied for both wide-area and data center networks [166, 89, 180, 41, 21]. Network consolidation concentrates traffic, represented in the form of a traffic matrix, on a subset of links and switches, and turns off remaining switches and links to save energy. Our work differentiates from prior work in two ways. First, prior work evaluates schemes mostly using traffic engineering metrics such as link utilization, while we evaluate actual end user latency for a real application and show that network consolidation can be performed with a small performance impact in CDCs. Second, we show network consolidation is closely related to server consolidation. Our network-aware server consolidation saves up to 45% more network energy over a network-unaware server consolidation scheme.

Global load balancing: Many papers [113, 141, 75, 144] have shown that geographical load-balancing across data centers can exploit the differences in elec-

tricity prices and in renewable energy availability at various locations to reduce energy costs, energy use, or non-renewable energy use. In comparison, our work focus on improving energy-efficiency of a single CDC by the use of consolidation. We believe that global load balancing can complement Shrink in reducing energy use and its cost across datacenters.

6.7 Conclusions

Content datacenters used for storing and serving content to end-users are common today. A major barrier to widespread adoption of server and network consolidation is CDC operators' concern on the impact on SLAs or user-perceived latencies. Our work takes a step towards addressing this concern by presenting a model to quantify the energy savings vs. latency inflation trade-off in CDCs. A key insight, supported via experiments, is that despite server consolidation, cache hit rates remain close to an unconsolidated datacenter, which helps mitigate the impact of consolidation on user-perceived latencies. We have designed and implemented Shrink, a system that leverages this tradeoff to yield significant energy savings while affecting user-perceived latencies in a controlled manner. Shrink's novel network-aware server consolidation algorithm reduces network energy use by up to 42% compared to network-unaware server consolidation schemes. Shrink, in experiments based on content access traces from an Akamai datacenter, reduced energy use by 35% compared to a baseline scheme that keeps entire datacenter always on while increasing mean latency by 8% over it. Overall, our findings encourage deployment of consolidation techniques to reduce CDC energy use.

CHAPTER 7

CONCLUSION

This thesis focused on infrastructure service design for a content-dominated, highly mobile Internet. We considered services for managing three types of infrastructures: an Internet service provider, a global name service and a content datacenter.

Our main contribution was to show that content placement is key to infrastructure service design and effective placement improves cost, performance, and energy-related metrics. In an ISP network, we saw that location diversity improved effective network capacity for all traffic engineering schemes, thereby blurring the difference between sub-optimal and optimal traffic engineering. In an NCDN, we saw that a simple demand-oblivious LRU caching scheme single-handedly yield network and latency costs close to the best possible even in conjunction with simple redirection and routing schemes. In Auspice, a demand-aware placement of name records resulted in significant improvents in latency and cost over existing managed DNS services, DHT-based name service designs and static placement policies. In Shrink, the effectiveness of a simple LRU caching contributed significantly to mitigating the performance impact of consolidation schemes.

Further, our work showed that, in practice, simple placement schemes perform well. Content placement problems are often NP-complete because they need to make binary decisions on whether to place content at a location or not. In practice, solving such computationally expensive problems may not be necessary because simple placement heuristics – random placement in an ISP network, LRU

caching in an NCDN and a CDC, and a simple demand-aware heuristic in Auspice – achieve significant benefits over existing strategies and/or achieve close to optimal results as shown in this thesis.

We also studied the relative importance of optimizing placement, redirection and routing in this thesis. In an ISP network with content location diversity and a Network CDN, we showed that content placement flexibility signficantly reduces the value of a carefully engineered routing. In a global name service, a better redirection scheme does improve latencies, e.g., random-k outperforms DHT due to a better redirection scheme. However, a demand-aware placement can significantly improve cost and performance even on top of an effective redirection scheme, e.g., Auspice does signficantly better than random-k, which uses the same redirection scheme as Auspice, as well as managed DNS, which uses global anycast routing. Overall, these findings strengthen our thesis that placement is of key importance in infrastructure service design.

APPENDIX A

COMPLEXITY OF NCDN PROBLEM

Opt-NCDN is the decision version of the NCDN problem described in §4.2. Opt-NCDN asks if the MLU of the network can be α while satisfying the constraints of the problem.

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

Proof: We show a reduction from the well known SetCover problem. We first define the SetCover problem that we will reduce to Opt-NCDN.

SetCover: Let $S = \{1, 2, ..., n\}$ be a set of n elements. Let $X = \{S_1, ..., S_m\}$ where $S_i \subseteq S, 1 \leq i \leq m$. Let k be an integer. SetCover asks if there exists $Y = \{Y_1, ..., Y_k\}$, where $Y_k \in X$ and $Y_1 \cup ... \cup Y_k = S$. Set Y is called a set cover of size k.

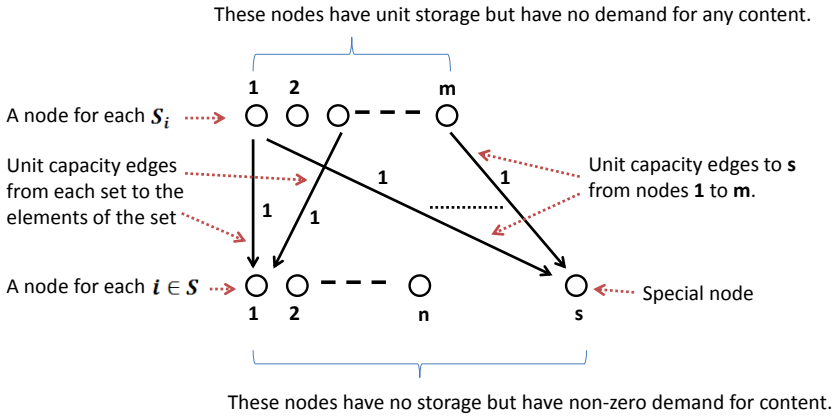


Figure A.1. Reduction from SetCover to Opt-NCDN

The reduction from SetCover to Opt-NCDN is described using the network in Figure A.1. Set $V_1 = \{1, ..., m\}$ refers to nodes in the top row. Each node $i \in V_1$ maps to the set $S_i \subset S$. Set $V_2 = \{1, ..., n\}$ refers to nodes in the bottom row excluding node s. Each node $i \in V_2$ maps to element $i \in S$. Node s is called a special node.

Directed links (i, j) exist from all nodes $i \in V_1$ to all nodes $j \in V_2$. The capacity of (i, j) is 1 unit if $i \in S_j$, otherwise capacity is zero. Node s has incoming links (i, s) from all nodes $i \in V_1$ such that the capacity of all incoming links is 1 unit.

All nodes in the top row V_1 have unit storage whereas nodes in the bottom row $V_2 \cup \{s\}$ have zero storage.

The set of objects is $\{o, 1, 2, ..., (m - k)\}$ and all objects have unit size. Object o is a special object that has unit demand at nodes in set $V_2 = \{1, ..., n\}$ and zero demand at all other nodes. Objects 1, 2, .. $(m - k)$ have unit demand at special node s and zero demand at all other nodes.

CLAIM: There is a set cover of size k if and only if the above network can achieve MLU ≤ 1.

If there is a set cover of size k, then the network can achieve MLU of 1. Store the special object o at the k set cover locations in the top row and satisfy demand for o at nodes $V_2 = \{1, ..., n\}$ in the bottom row from these locations with MLU = 1. The remaining $(m - k)$ nodes in the top can be used for objects $\{1, 2, ..., (m - k)\}$ to satisfy the demand at special node s with MLU of 1.

If there is no set cover of size k, then the network must have a MLU > 1. Objects must be placed in some $(m - k)$ nodes in the node $V_1 = \{1, ..., m\}$ in the top row to satisfy the demand for special node s. Thus, at most k nodes are available for placing special object o. Since there is no set cover of size k, some bottom node $i \in V_2$ must satisfy its demand for special object o using an edge whose capacity is zero resulting in MLU = ∞ on that edge.

It is easy to show that Opt-NCDN $\in$ NP. Hence, Opt-NCDN is NP-Complete.

THEOREM 2 *Opt-NCDN is inapproximable within a factor β for any $\beta > 1$ unless P = NP.*

The proof of THEOREM 1 shows that if there is a set cover of size k, MLU = 1 and MLU $= \infty$ otherwise. Thus, if we find a solution for which MLU is finite, it implies that MLU = 1, which immediately gives a solution to the corresponding SetCover instance.

Lets assume a β-approximation ($\beta > 1$) exists for Opt-NCDN. Then, we can solve SetCover in polynomial time by mapping SetCover instance to Opt-NCDN instance, and checking if MLU $\leq \beta$ (which implies MLU = 1). As SetCover $\in$ NP-Complete, therefore, no $\beta-$approximation for Opt-NCDN exists unless P = NP.

A.1 Joint optimization of transit traffic matrix and content matrix

We present here a modification to the MIP in §4.2.3 to jointly optimize routing for an ISP transit traffic matrix (TTM) and a content matrix. Let D be a TTM and D_{ij} denote the traffic from PoP i to PoP j. We modify only the constraints (3) and (4) in the earlier MIP as follows:

$$\sum_{k \in K} t_{ijk} + D_{ij} \quad = \quad f_{ij},\ \forall j \in V - X, i \in V \tag{A.1}$$

$$\sum_{k \in K} t_{ijk} + \sum_{k \in K} \delta_{ij} t_{iok} + D_{ij} \quad = \quad f_{ij},\ \forall j \in X, i \in V \tag{A.2}$$

A.2 Limiting content placement update traffic

In this section, we describe our extension to the MIP presented in §4.2.3 which allows us to limit the MLU due to traffic from updating the content placement. To

this end, we add constraints to the MIP to ensure that the MLU due to the placement update traffic is less than a constant β. In our experiment, we dynamically update the value of β to be two/third of the MLU within the past 24 hours of the experiment.

We follow the same notation as in §4.2.3. The binary variable x_{ik} denotes if the content $k \in K$ is stored at node $i \in V$, S_k denotes the size of the content. To describe the constraint, we define the following parameters : T denotes the duration over which traffic due to placement update will be spread out. X_{jk} denotes whether content $k \in K$ is stored at node $j \in V$ currently. The current routing in the network is r_{ije}, the fraction of traffic from node j to node i crossing link e. In addition we define a function $\gamma(i, j, k)$. $\gamma(i, j, k) = 1$ if $X_{jk} = 1$ and node j is the closest node in terms of hop count from node i which has stored a copy of content k, otherwise. Both r_{ije} and $\gamma(i, j, k)$ depend on current routing and placement in the network and hence are known constants. We assume that the transfer of content of size S_k happens at a constant bit rate of S_k/T. In terms of these variables we can define the total traffic u_e on any link $e \in E$ during the placement update period.

$$u_e = \sum_{i \in V} \sum_{j \in V} \sum_{k \in K} \gamma(i, j, k) r_{ije} x_{jk} S_k / T \quad \forall e \in E \tag{A.3}$$

Finally, in order to limit the maximum utilization of any link $e \in E$, we add the following constraint,

$$u_e / C_e < \beta \quad \forall e \in E \tag{A.4}$$

APPENDIX B

NETWORK ENERGY LOWER BOUND

We derive lower bounds on the network energy use for FatTree [17] and VL2 [82] under the constraint that n servers that are active must be able to simultaneously send traffic to clients equal to the external bandwidth E via the set of active switches only.

VL2: Let the energy use of each core, aggregation and ToR switch be PC, PA and PT respectively. Let L be the capacity of links between each pair of core and aggregation switches. If c core switches and a aggregation switches be active, then the maximum number of servers that can be supported is *nmax* $= (a \times c \times L/E)$ and the total energy use of core and aggregation switches is *etotal* $= (c \times PC + a \times PA)$. We select the optimal values of *a_opt* and *c_opt* (by enumerating all values) such that *etotal* is minimized under the constraint that *nmax* $> n$. Assuming each ToR switch connects to k servers, the minimum number of ToR switches needed is $\lceil n/k \rceil$. Thus, a lower bound on the total network energy use is $(\lceil n/k \rceil PT) + (c\_opt \times PC + a\_opt \times PA)$.

FatTree: Switches are identical in a FatTree. So, a lower bound the number of active switches gives a lower bound on network energy use also.

Let $m_1, \cdots m_k$ be the active servers in the k pods so that $m_1 + \cdots + m_k = n$. In a pod with m active servers, at least $2\sqrt{m}$ switches must be active. Thus, a total of $(2(\sqrt{m_1} + \cdots + \sqrt{m_k}))$ pod switches must be active.

Let c be the number of active core switches. We claim that the number of active servers in any pod can at most be c. The reason is that a core switch has

only one link to switches in each pod, and hence can receive traffic from only one server sending traffic at its outgoing link capacity to external clients. The values of $m_1, \cdots m_k$ that minimizes the number of active pod switches is given by $m_1 = m_2 = \cdots = m_l = c$, $m_{l+1} = (n \bmod c)$, and $m_{l+2} = m_{l+3} = \cdots m_k = 0$, where $l = \lfloor n/c \rfloor$. Let p be minimum number of active pod switches thus computed. Then, the minimum number of total active switches active is given by $(p + c)$.

Computing the minimum number of switches for all possible values of c ($c \leq n, c \leq k^2/4$) and taking their minimum gives a lower bound on the number of active switches for this topology.

BIBLIOGRAPHY

[1] A Global Name Service for a Highly Mobile Internetwork. UMass Amherst Computer Sciene Technical Report. https://web.cs.umass.edu/publication.

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

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

[4] Cassandra. http://cassandra.apache.org.

[5] MobilityFirst Future Internet Architecture Project. http://mobilityfirst.cs.umass.edu/.

[6] Mongo DB. http://www.mongodb.org/.

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

[8] Server fault: DNS - Any way to force a name server to update the record of a domain? http://serverfault.com/questions/41018/dns-any-way-to-force-a-nameserver-to-update-the-record-of-a-domain.

[9] The Locator/ID Separation Protocol (LISP). RFC 6830.

[10] ICANN hears concerns about accountability, control, 2008. http://www.infoworld.com/t/networking/icann-hears-concerns-about-accountability-control-216.

[11] Debate rages over who should control ICANN, 2009.

[12] 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).

[13] Abhigyan, Venkataramani, A, and Sitaraman, R. Distributing Content Simplifies ISP Traffic Engineering. *UMass Amherst Computer Sciene Technical Report*. https://web.cs.umass.edu/publication.

[14] Administration, U.S. Energy Information. FAQs. http://www.eia.gov/tools/faqs/.

[15] 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).

[16] Akamai. Facts and figures, 2014. `http://www.akamai.com/html/about/facts_figures.html`.

[17] Al-Fares, Mohammad, Loukissas, Alexander, and Vahdat, Amin. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review* (2008), ACM.

[18] Amazon. Cloudfront service level agreement, 2013. `http://aws.amazon.com/cloudfront/sla/`.

[19] Amazon. Ec2, 2014. `http://aws.amazon.com/ec2/`.

[20] Andersen, David G., Balakrishnan, Hari, Feamster, Nick, Koponen, Teemu, Moon, Daekyeong, and Shenker, Scott. Accountable internet protocol (aip). In *SIGCOMM* (2008).

[21] 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.

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

[23] Apple. HTTP Live Streaming. `http://tools.ietf.org/html/draft-pantos-http-live-streaming-08`.

[24] 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).

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

[26] Applegate, David, Archer, Aaron, Gopalakrishnan, Vijay, Lee, Seungjoon, and Ramakrishnan, K. K. Optimal content placement for a large-scale vod system. CoNEXT.

[27] Arye, M., Nordstrom, E., Kiefer, R., Rexford, J., and Freedman, M. J. A formally-verified migration protocol for mobile, multi-homed hosts. In *ICNP* (2012).

[28] AT&T. Content Distribution, 2011. `http://www.business.att.com/enterprise/Service/hosting-services/content-delivery/distribution/`.

[29] Balakrishnan, H., Lakshminarayanan, K., Ratnasamy, S., Shenker, S., Stoica, I., and Walfish, M. A layered naming architecture for the internet. In *SIGCOMM* (2004).

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

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

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

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

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

[35] Caesar, Matthew, Condie, Tyson, Kannan, Jayanthkumar, Lakshminarayanan, Karthik, and Stoica, Ion. ROFL: routing on flat labels. *SIGCOMM.* (2006).

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

[37] Cast, Edge. `http://www.edgecast.com/solutions/licensed-cdn/`.

[38] 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.

[39] Chase, Jeffrey S, Anderson, Darrell C, Thakar, Prachi N, Vahdat, Amin M, and Doyle, Ronald P. Managing energy and server resources in hosting centers. In *ACM SIGOPS Operating Systems Review* (2001), ACM.

[40] Che, Hao, Tung, Ye, and Wang, Zhijun. Hierarchical web caching systems: Modeling, design and experimental results. *Selected Areas in Communications, IEEE Journal on 20*, 7 (2002), 1305–1314.

[41] 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.

[42] Cisco. Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2012-2017. `http://ciscovni.com`.

[43] Cisco. Catalyst 6500, 2014. http://www.cisco.com/c/en/us/products/collateral/switches/catalyst-6500-series-switches/.

[44] Cisco. Data center switches, 2014. http://www.cisco.com/c/en/us/products/switches/data-center-switches/index.html.

[45] Cisco. Visual Networking Index, 2014. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf.

[46] citrix. The 4 Keys to Telco CDN Success. https://www.citrix.com/content/dam/citrix/en_us/documents/products-solutions/the-4-keys-to-telco-cdn-success.pdf.

[47] Cohen, B. BitTorrent Protocol. http://www.bittorrent.org/beps/bep_0003.html.

[48] Cohen, Bram. Incentives build robustness in Bittorrent. In *P2PEcon* (2003).

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

[50] console, Remote, and power management systems. Wti, 2014. http://www.wti.com/t-remote-console-and-power-management-for-cisco-routers.aspx.

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

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

[53] 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).

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

[55] Dilley, J, Maggs, B, Parikh, J, Prokop, H, Sitaraman, R, and Weihl, B. "Globally distributed content delivery,". In *IEEE Internet Computing* (Sep/Oct2002).

[56] Dilley, John, Maggs, Bruce, Parikh, Jay, Prokop, Harald, Sitaraman, Ramesh, and Weihl, Bill. Globally distributed content delivery. *Internet Computing, IEEE 6*, 5 (2002), 50–58.

[57] 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.

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

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

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

[61] 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.

[62] 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.

[63] Fan, Xiaobo, Ellis, Carla, and Lebeck, Alvin. Memory controller policies for dram power management. In *Proceedings of the 2001 international symposium on Low power electronics and design* (2001), ACM, pp. 129–134.

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

[65] Feldmann, Anja, Cittadini, Luca, Mühlbauer, Wolfgang, Bush, Randy, and Maennel, Olaf. Hair: hierarchical architecture for internet routing. In *ReArch* (2009).

[66] Flickenger, Rob. *Linux server hacks: 100 industrial-strength tips and tools*, vol. 1. " O'Reilly Media, Inc.", 2003.

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

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

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

[70] 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).

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

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

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

[74] Funato, D., Yasuda, K., and Tokuda, H. Tcp-r: Tcp mobility support for continuous operation.

[75] 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.

[76] Gao, Zhaoyu, Venkataramani, Arun, and Kurose, James F. Towards a quantitative comparison of location-independent network architectures. In *ACM SIGCOMM* (2014).

[77] Gartner. Mobile Connections Will Reach 5.6 Billion in 2011, 2011. `http://www.gartner.com/it/page.jsp?id=1759714`.

[78] 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.

[79] Google. Google caching overview, 2014. `https://peering.google.com/about/ggc.html`.

[80] Greenberg, Albert, Hamilton, James, Maltz, David A., and Patel, Parveen. The cost of a cloud: Research problems in data center networks. *SIGCOMM Comput. Commun. Rev. 39*, 1 (Dec. 2008), 68–73.

[81] Greenberg, Albert, Hamilton, James, Maltz, David A., and Patel, Parveen. The cost of a cloud: Research problems in data center networks. *SIGCOMM Comput. Commun. Rev. 39*, 1 (Dec. 2008), 68–73.

[82] Greenberg, Albert, Hamilton, James R, Jain, Navendu, Kandula, Srikanth, Kim, Changhoon, Lahiri, Parantap, Maltz, David A, Patel, Parveen, and Sengupta, Sudipta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM Computer Communication Review* (2009), ACM.

[83] Gritter, Mark, and Cheriton, David R. An architecture for content routing support in the internet. In *USITS* (2001).

[84] 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.

[85] 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).

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

[87] Hamilton, James. Cost of power in large-scale data centers, 2008. `http://perspectives.mvdirona.com/2008/11/28/CostOfPowerInLargeScaleDataCenters.aspx`.

[88] Han, Dongsu, Anand, Ashok, Dogar, Fahad, Li, Boyan, Lim, Hyeontaek, Machado, Michel, Mukundan, Arvind, Wu, Wenfei, Akella, Aditya, Andersen, David G., Byers, John W., Seshan, Srinivasan, and Steenkiste, Peter. Xia: efficient support for evolvable internetworking. In *NSDI* (2012).

[89] 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.

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

[91] Hopps, Christian E. Analysis of an equal-cost multi-path algorithm.

[92] HP. The edge of bandwidth growth. `http://www.hpcloud.com/products-services/cdn`.

[93] HP. Service level agreement for hp cloud cdn, 2014.

[94] 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.

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

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

[97] Jacobson, Van, Smetters, Diana K., Thornton, James D., Plass, Michael F., Briggs, Nicholas H., and Braynard, Rebecca L. Networking named content. In *CoNEXT* (2009).

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

[99] Jokela, P., Nikander, P., Melen, J., Ylitalo, J., and Wall, J. Host identity protocol, extended abstract. In *Wireless World Research Forum* (2004).

[100] Jung, Jaeyeon, Sit, Emil, Balakrishnan, Hari, and Morris, Robert. Dns performance and the effectiveness of caching. *IEEE/ACM Trans. Netw.*.

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

[102] Kleinrock, Leonard. In *Queueing Systems Vol 1: Theory. p. 77* (1975).

[103] Koponen, Teemu, Chawla, Mohit, Chun, Byung-Gon, Ermolinskiy, Andrey, Kim, Kye Hyun, Shenker, Scott, and Stoica, Ion. A data-oriented (and beyond) network architecture. In *SIGCOMM* (2007).

[104] Krioukov, Dmitri, claffy, k c, Fall, Kevin, and Brady, Arthur. On compact routing for the internet. *SIGCOMM CCR* (2007).

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

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

[107] 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.

[108] Lampson, Butler W. Designing a global name service. In *Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing* (1986), PODC '86.

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

[110] Level3. Content Delivery Network, 2011. `http://www.level3.com/en/products/content-delivery-network/`.

[111] 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).

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

[113] 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.

[114] Live, PP. `http://www.pplive.com/`.

[115] Lu, Yung-Hsiang, and De Micheli, Giovanni. Adaptive hard disk power management on personal computers. In *Great Lakes Symposium on VLSI* (1999), IEEE Computer Society, pp. 50–50.

[116] 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.

[117] Mahadevan, Priya, Sharma, Puneet, Banerjee, Sujata, and Ranganathan, Parthasarathy. A power benchmarking framework for network devices. In *NETWORKING 2009*. Springer, 2009, pp. 795–808.

[118] 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.

[119] Media, Streaming. Telco-CDN Whitepapers. `http://blog.streamingmedia.com/2011/03/three-new-whitepapers-released-on-the-telco-cdn-space.html`.

[120] Microsoft. Windows azure support, 2014.

[121] Mockapetris, P. Domain names - concepts and facilities. *The Internet Society RFC 1034* ("1987").

[122] 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.

[123] Netgear. Gs105 5-port gigabit ethernet switch, 2014. `http://www.downloads.netgear.com/files/GDC/GS105/GS105_datasheet_04Sept03.pdf`.

[124] Nielsen. Online Video Usage Up 45%. `http://http://www.nielsen.com/us/en/insights.html`.

[125] Nordstrom, Erik, Shue, David, Gopalan, Prem, Kiefer, Robert, Arye, Matvey, Ko, Steven Y., Rexford, Jennifer, and Freedman, Michael J. Serval: An end-host stack for service-centric networking. In *NSDI* (2012).

[126] NSF. MobilityFirst Future Internet Architecture Project, 2012. `http://mobilityfirst.winlab.rutgers.edu//`.

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

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

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

[130] 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).

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

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

[133] Parwez, M.R., and et al. DNS propagation delay: An effective and robust solution using authoritative response from non-authoritative server. In *ICIME* (2010).

[134] PDU, Switched. Server tech, 2014. `http://www.servertech.com/products/switched-pdus/`.

[135] Perkins, Charles E. Mobile IP. *IEEE Comm. Magazine* (May 1997).

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

[137] PG&E. Rate information center, 2014. `http://www.pge.com/en/mybusiness/rates/rateinfo/index.page`.

[138] Pinheiro, Eduardo, Bianchini, Ricardo, Carrera, Enrique V, and Heath, Taliver. Load balancing and unbalancing for power and performance in cluster-based systems. In *Workshop on compilers and operating systems for low power.* (2001).

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

[140] 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.

[141] 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).

[142] Rajamani, Karthick, and Lefurgy, Charles. On evaluating request-distribution schemes for saving energy in server clusters. In *Performance Analysis of Systems and Software, 2003. ISPASS. 2003 IEEE International Symposium on* (2003), IEEE, pp. 111–122.

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

[144] 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.

[145] Rasmussen, Neil. Determining total cost of ownership for data center and network room infrastructure. *White Paper* (2011).

[146] Rayburn, Dan. Comparing CDN Performance: Amazon CloudFront Last Mile Testing Results.

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

[148] Report, ATLAS Internet Observatory 2009 Annual. `http://www.nanog.org/meetings/nanog47/presentations/-Monday/Labovitz\_ObserveReport\_N47\_Mon.pdf`.

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

[150] RFC. 2616. `http://www.ietf.org/rfc/rfc2616txt`.

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

[152] 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).

[153] Schroeder, Michael D., Birrell, Andrew D., and Needham, Roger M. Experience with grapevine: the growth of a distributed system. *ACM Trans. Comput. Syst.* (1984).

[154] Seagate. Barracuda es.2 serial ata product manual, 2009. http://www.seagate.com/staticfiles/support/disc/manuals/NL35\%20Series\%20&\%20BC\%20ES\%20Series/Barracuda\%20ES.2\%20Series/100468393h.pdf.

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

[156] Snoeren, A. C., and Balakrishnan, H. An end-to-end approach to host mobility. In *MobiCom* (2000).

[157] Snoeren, Alex C, Andersen, David G, and Balakrishnan, Hari. Fine-grained failover using connection migration. In *USITS* (2001), vol. 1, pp. 19–19.

[158] SPEC. Standard performance evaluation corporation, 2014.

[159] Starr, Matt, and Rinehart, Will. Does cable really have a 97 percent profit margin?. http://dailycaller.com/2013/02/15/does-cable-really-have-a-97-profit-margin/.

[160] Stoica, Ion, Adkins, Daniel, Zhuang, Shelley, Shenker, Scott, and Surana, Sonesh. Internet indirection infrastructure. In *SIGCOMM* (2002).

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

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

[163] Sultan, Florin, Srinivasan, Kiran, Iyer, Deepa, and Iftode, Liviu. Migratory tcp: Connection migration for service continuity in the internet. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on* (2002), IEEE, pp. 469–470.

[164] Topology, Abilene. http://www.cs.utexas.edu/~yzhang/research/AbileneTM/.

[165] Trushkowsky, Beth, Bodík, Peter, Fox, Armando, Franklin, Michael J., Jordan, Michael I., and Patterson, David A. The scads director: Scaling a distributed storage system under stringent performance requirements. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies* (Berkeley, CA, USA, 2011), FAST'11, USENIX Association, pp. 12–12.

[166] 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.

[167] Verizon. HBO for FIOS Customers. http://www.verizon.com/home/MLP/Premium.html?x1=HBO.

[168] Verizon. Velocix at Verizon, 2011. http://www.lightreading.com/velocix-verizon-deal-changed-everything/d/d-id/669602.

[169] 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).

[170] Walfish, Michael, Stribling, Jeremy, Krohn, Maxwell, Balakrishnan, Hari, Morris, Robert, and Shenker, Scott. Middleboxes no longer considered harmful. In *OSDI* (2004).

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

[172] Wang, Xiao Sophia, Balasubramanian, Aruna, Krishnamurthy, Arvind, and Wetherall, David. Demystifying page load performance with wprof. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)* (2013).

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

[174] Wessels, D, and Claffy, K. Rfc 2186: Internet cache protocol (icp). Tech. rep., version 2. RFC, IETF, 1997.

[175] White, Brian, Lepreau, Jay, Stoller, Leigh, Ricci, Robert, Guruprasad, Shashi, Newbold, Mac, Hibler, Mike, Barb, Chad, and Joglekar, Abhijeet. An integrated experimental environment for distributed systems and networks. In *OSDI* (2002).

[176] Wikipedia. Voltage and frequency scaling, 2014. http://en.wikipedia.org/wiki/Voltage_and_frequency_scaling.

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

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

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

[180] 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.