Towards a Scalable, Adaptive and Network-aware Content Distribution Network

by

Yan Chen

B.E. (Zhejiang University) 1995
M.S. (State University of New York at Stony Brook) 1998

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

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Randy H. Katz, Chair
Professor Ion Stoica
Professor John Chuang

Fall 2003

The dissertation of Yan Chen is approved:

Chair Date

Date

Date

University of California at Berkeley

Fall 2003

Towards a Scalable, Adaptive and Network-aware Content Distribution Network

Abstract

Towards a Scalable, Adaptive and Network-aware Content Distribution Network

by

Yan Chen
Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor Randy H. Katz, Chair

The Internet has evolved to become a critical commercial infrastructure for service delivery. However, The Internet being an enormous, highly-dynamic, heterogeneous, and untrusted environment raises several challenges for building Internet-scale services (such as content delivery) with good scalability, efficiency, agility and security. In this thesis, we explore these issues by developing a scalable, adaptive and network-aware infrastructure for efficient content delivery, namely Scalable Content Access Network (SCAN).

SCAN has four components: object location, replica placement and update multicast tree construction, replica management, and overlay network monitoring services.

First, we propose a novel simulation-based network Denial of Service (DoS) resilience benchmark, and apply it to evaluate and compare the centralized, replicated, and emerging distributed object location services. We find that distributed hash table (DHT) based object location service has the best resilience in practice. Thus we leverage it for replica location.

Second, we propose the first algorithm that dynamically places close to optimal number of replicas while meeting client QoS and server resource constraint, with overlay network topology only. These replicas further self-organize them into an application-level multicast tree on top of a DHT, Tapestry, which enables distributed "join" so that each node (including the root) in the tree only needs to maintain states for its parent and direct children.

Third, we apply cooperative clustering-based replication to SCAN, which achieves comparable users' perceived performance to the conventional CDNs, while having only 4 - 5% of replication and update traffic, and 1 - 2% of the computation and replica management cost. Furthermore, we propose a unique online Web object popularity prediction algorithm based only on hyperlink structures, and applied it for online incremental clustering and replication to adapt to changes in users' access patterns. This scheme adds new content to the appropriate existing cluster replicas even before accessed, to improve their availability during flash crowds.

Fourth, to provide a general foundation for applications to take advantage of network awareness, we develop a scalable overlay network measurement and monitoring system with two components: Internet Iso-bar for latency estimation, and TOM (Tomography-based Overlay network Monitoring) for loss rate estimation. Internet Iso-bar clusters hosts

based on the similarity of their perceived network distance to a small number of landmark sites, and chooses the centroid of each cluster as a monitor site. Evaluation using real Internet measurements shows that our scheme offers much better latency accuracy and stability than the traditional network/geographical proximity-based clustering. Furthermore, it detects 78% of congestion/failures with 32% false positive.

For mission-critical applications that require more accurate loss rate estimation, we developed TOM with a bit more measurements than Internet Iso-bar ($O(n \log n)$ instead of $O(k^2 + n)$, but still much less than the current work $O(n^2)$, where n is the number of end hosts and k is the number of clusters). TOM selectively monitors and measures the loss rates of a minimal basis set of $O(n \log n)$ linearly independent paths, and then applies them to estimate the loss rates of all other paths. Both extensive simulation and Internet experiments show that TOM achieves high path loss rate estimation accuracy, has good load balancing, tolerates topology measurement errors and is adaptive to topology changes.

Finally, to demonstrate the effectiveness of the monitoring services, we develop a adaptive overlay streaming media system which leverages our monitoring services for real-time path congestion/failure information, and an overlay network for adaptive packet relaying and buffering within the delivery infrastructure. Traditional streaming media systems treat the underlying network as a best-effort black box, and adaptations are performed at the transmission end-points. However, our Internet experiments show that overlay routing can often improve the loss rate and/or TCP throughput for lossy paths, and our system typically adapts to network congestions within five seconds, achieving skip-free streaming media playback.

Professor Randy H. Katz
Dissertation Committee Chair

To my parents and my wife,

without whom this dissertation would be impossible.

Contents

List of Figures

List of Tables

Acknowledgements

I am extremely grateful to Prof. Randy H. Katz, my thesis advisor, for giving me the chance to work with him in the SAHARA project about three years ago. The trust he had in me gave me the courage, confidence and strength to overcome the challenges I faced and the frustration I experienced as a beginner in this field. His unprecedented support led me through my Ph.D. study, and has always upheld me when such support was most needed. He taught me how to find important problem to investigate, not just for academic research, but also for potential impact on the industry - how will my research eventually influence the society and people's life. He also kept reminding me that the quality/impact of research is much more important than the number of publications. Randy not only taught me how to become a better researcher and a better scholar, but also taught me how to become a better teacher and a better mentor. His engaging arguments and strong feedback have contributed greatly to this dissertation. I hope and look forward to continued collaboration with him in the future.

I thank my dissertation committee members, Profs. Ion Stoica and John Chuang, for their comments and suggestions during the course of developing my research agenda and completing this dissertation. I also thank Prof. John D. Kubiatowicz, for guiding me in the beginning of my system research and chairing my qualifying exam committee.

My thesis work was significantly benefited from collaborating with both industry and academic researchers: Prof. John D. Kubiatowicz (U.C. Berkeley), Drs. Chris Overton (Crazy Tulip Inc., used to be affiliated with Keynote Inc.), Lili Qiu (Microsoft Research), and Wai-tian Tan (HP Labs). Being treated as a peer and a friend, I thoroughly enjoyed working with them, and look forward to continuing the collaborations in the future.

I am grateful for valuable feedback received from Dr. Peter Danzig (consultant), Prof. Anthony Joseph (U.C. Berkeley), Prof. Eugene Ng (Rice) and Dr. Vern Paxon (ICIR/LBL). I have learnt a lot through interacting with them. I also thank Prof. Tony McGregor and Dr. Jing Zhi for providing NLANR and Keynote measurement data.

During my thesis study, I was fortunate to work with many excellent undergraduate and graduate students in Berkeley and other institutes: Alvin AuYoung (now graduate student at U. C. San Diego), Adam Bargteil, David Bindel, Weiyu Chen, Brian Chavez (U. C. Santa Cruz), Chris Karlof, Johnny Lam, Yaping Li, Khian Hao Lim (now graduate student at Stanford), Luan Nguyen (now graduate student at U. C. Los Angeles), Jacob Scott, Hanhee Song, Albert Wang, and Yingying Wei. The collaboration (sometimes mentoring) experience with them was indeed pleasant, which motivated me to join academia after graduation.

I interned at AT&T Shannon Labs - Research in the summer of 2002. I thank my mentors Drs. Balachander Krishnamurthy, Shubho Sen and Yin Zhang for their guidance as I learned high-speed network measurement and monitoring, and statistical traffic analysis.

My wonderful memories of graduate school were most enriched by my day-to-day interactions with fellow graduate students. I thank the members of Soda 465, Soda 473, and other EECS comrades - Sharad Agarwal, Matthew Caesar, Hao Chen, Chen-Nee Chuah, Weidong Cui, Matt Denny, Yunfei Deng, Yitao Duan, Patrick Eaton, Alyosha Efros, Ling Huang, Hoon Kang, Karthik Lakshminarayanan, Tal Lavian, Yong Liu, Z. Morley Mao, Sridhar Machiraju, George Porter, Bhaskaran Raman, Sylvia Ratnasamy, Xiaofeng Ren, Mukund Seshadri, Chen Shen, Jimmy Shih, Wilson So, Zhendong Su, Lakshminarayanan Subramanian, Helen Wang, Hakim Weatherspoon, Victor Wen, Bo Wu, Eric Po Xing, Li Yin, Fang Yu, Yizhou Yu, Lei Yuan, Hao Zhang, Ben Zhao, Shelley Zhuang, and Yan Zhuang.

I thank Damon Hinson, Bob Miller, Veronique Richard and Gretchen Sanderson who managed our research group matters. They were very friendly to work with and always responded promptly to my requests and questions regarding administrative issues. I sincerely acknowledge Keith Sklower for maintaining our network and computers, and being very responsive whenever I ran into problems and called for help.

I would like to express my earnest gratitude to my parents and my brother for their love and support, and for raising me and educating me. My deepest gratitude goes to my dear wife Fei for her love and understanding through these years. She was always behind me and gave her unconditional support even when that meant to sacrifice the time we spent together. I especially appreciate my parents' and my wife's patience for this boundless journey. They have always given me the freedom to make my own choices and have always wanted my happiness above their own. I also thank my parents-in-law, my sister-in-law, Peng Znou, and her husband, Xiaolan Peng. Both my academic career and family life have been greatly benefited from their constant care and support.

I give thanks to the members of the Chinese for Christ Berkeley Church, especially the Young Adult Fellowship, for their prayers, support and care. Special thanks to Amy Jie and Aray Ge who helped me grow in Christ. Last but the most, I want to give thanks to the One and Only God for His unconditional love, and for being my shepherd, my shield and my refuge, all the times.

Chapter 1

Introduction

1.1 Motivation

The Internet has evolved to become a critical commercial infrastructure for service delivery. However, due to the IP addressing scheme, routing paradigm, and other historical reasons, the current Internet is not well suited for the purpose of emerging new service delivery contemplated by today's applications. In response to these challenges, various forms of overlay networks, introducing new functionality within the network near the edges, have been proposed and some deployed in the Internet. For instance, commercial Content Distribution Networks (CDN) use overlay technology to bring content closer to the end users. The Internet being an enormous, highly-dynamic, heterogeneous, and untrusted environment raises several challenges for building Internet-scale services (such as content delivery) with good scalability, efficiency, agility and security.

To build an Internet-scale services, we need to address the following challenges:

- **Scalability** It is estimated to be 581 million Internet users in May 2002, and the number will reach one billion by 2005 [86]. The Internet carries 2,000TB traffic a day [89]. Despite such huge scale and fast growth, most existing Internet services rely on centralized architecture, *e.g.*, a cluster of local machines connected with high speed Ethernet. Such design is fundamentally unscalable to the number of clients/requests, will cause hot spots and single point of failure.

- **Efficiency** Given the rapid spread of portable computing devices, and wireless and sensor networks, end users have increasingly heterogeneous connectivity and devices (CPU, memory, *etc.*), desiring different quality of service (QoS). Current Internet applications treat all the clients equally with the best-effort services. To support different QoS of massive heterogenous clients requires distributed, efficient and balanced resource consumption of the underlying infrastructure.

  We have been enjoying Moore's law for many years, *i.e.*, the computational power, DRAM capacity, and the disk storage capacity have been doubling every 18 - 24 months. However, according to the recent "International Technology Roadmap for Semiconductors 2002 Update" report, the present chip-size model will slow the Moore's

Law rate of on-chip transistors to 2× every three years. Another important metric, DRAM chip size reduction rate, is also estimated to stretch to 2×/2.5 to three years [117]. Meanwhile, the speed of information explosion outpaced that of the technology improvement - the annual digital information growth is about 50% [73]. Thus it is crucial to have Internet service providers to be efficient for bandwidth, storage, and management cost.

- **Agility** All the entities in the system, from clients, servers to network, change their status continuously. For instance, clients may change their interests for different contents and change their access patterns, servers may have various loads or even crash, network may have congestion or failures. Meanwhile, it is crucial for mission-critical applications to provide 24×7×365 availability. Obviously, Internet services need to monitor these changes and adapt to them.

  However, most existing Internet services ignore these dynamics and only provide best-effort services. Recent CDNs like Akamai provide user-level agility in an inefficient manner (see Chapter 5.4), and provide limited system/network-level agility with inaccurate and unscalable monitoring (use clients' DNS server as clients representative, see Chapter 6.1.1).

- **Security** Various forms of denial-of-service (DoS) attacks and viruses/worms are plaguing the Internet. For instance, Yankee Group, an Internet research firm, estimated that DoS attacks cost $1.2 billion in lost revenues in 2000 [101]. However, even well-known Internet service/application providers, such as Yahoo, Amazon, eBay and Microsoft's DNS, suffered attacks.

In this thesis, I address these issues (especially the first three) by developing a suite of techniques to build a CDN, namely Scalable Content Access Network (SCAN), on top of a peer-to-peer location system. Content delivery is important because it has been dominating the Internet traffic. There are more than 3.3 billion Web pages [55] and a growth of 7 million pages every day [95]. The convergence in the digital world of voice, data and video is expected to lead to a compound growth rate of Web traffic.

Essentially, CDN improves content delivery performance by replicating content to multiple locations in the Internet, and by having users obtain data from the "best" data repository. Here, "best" is defined by the desired quality of service of clients, e.g., latency, bandwidth, and/or load. However, it remains an open issue how to provision, manage, and monitor such a replication infrastructure efficiently. In particular, we seek to address the following questions.

1. How to dynamically deploy optimal number number of replicas to meet clients' QoS (*e.g.*, latency) constraints without overloading servers?

2. How to maintain the coherence of the replicas when the original contents are changed?

3. Given any client's request, how to find its nearby replica with good scalability?

4. How to reduce the replica management overhead (*e.g.*, the amount of replica index states to maintain) without sacrificing the performance for end users (*e.g.*, latency)?

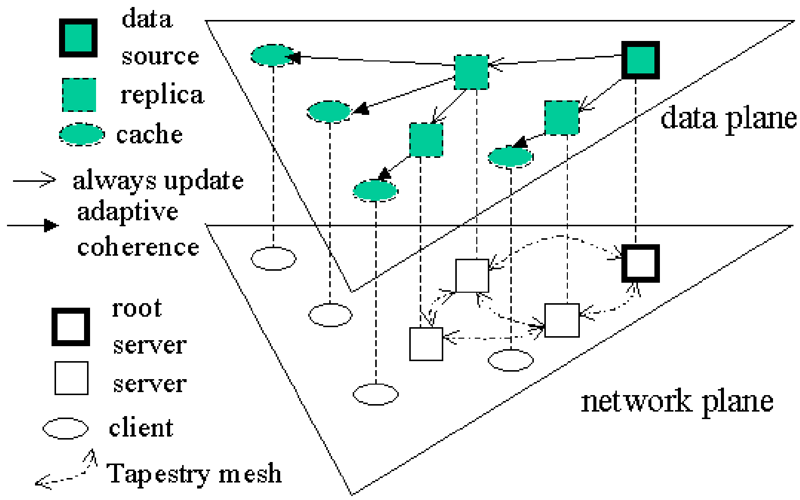


Figure 1.1: A sample SCAN system.

5. How to estimate the network end-to-end distance, such as latency and loss rate, in a scalable and accurate manner to provide proximity and adaptation to network congestions and failures?

6. How to leverage network monitoring services to build adaptive applications?

Next, we will discuss the SCAN architecture and our contributions for solving the questions above.

1.2 SCAN Architecture

SCAN is a self-organizing soft-state replication system, as illustrated in Figure 1.1. There are two classes of physical nodes shown in the network-plane of this diagram: *SCAN servers* (squares) and *clients* (circles). We assume that SCAN servers are placed in Internet Data Centers (IDC) of major ISPs with good connectivity to the backbone. Each SCAN server may contain replicas for a variety of data items. One novel aspect of the SCAN system is that it assumes SCAN servers participate in a distributed routing and location (DOLR) system (*a. k. a.* distributed hash table (DHT)), Tapestry [137]. Tapestry permits clients to locate nearby replicas without global communication. Note that Tapestry is shared across objects, while each object has a application-level multicast tree for disseminating updates.

There are three types of data illustrated in Figure 1.1: Data *sources* and *replicas* are the primary topic of this thesis. They reside on SCAN servers and are always kept up-to-date. *Caches* are the images of data that reside on clients and are beyond our scope[1].

[1]Caches may be kept coherent in a variety of ways (for instance [111]).

1.3 Dissertation Overview and Contributions

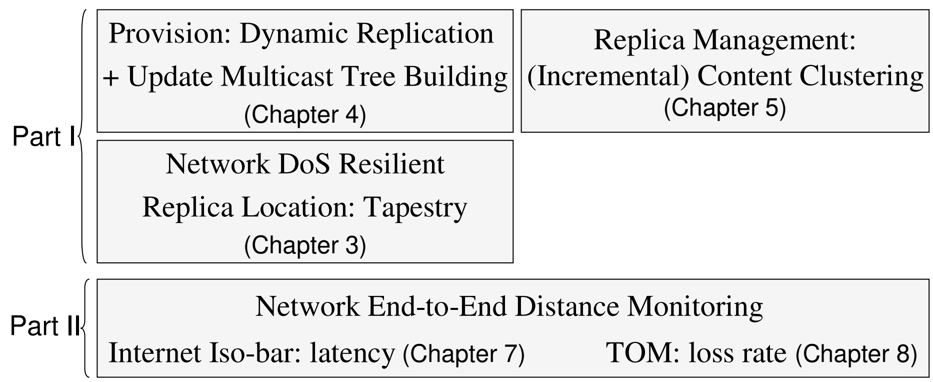


Figure 1.2: Layered architecture of the SCAN system.

As shown in Figure 1.2, SCAN leverages a peer-to-peer distributed hash table (DHT) system for scalable object location, applies dynamic replica placement for provisioning, achieves scalable replica management through (incremental) clustering of objects, and is monitored with scalable overlay network monitoring.

We summarize our contributions in the four components as follows:

1.3.1 Network DoS-resilient Object Location with DHT (Chapter 3)

Networked applications are extending their reach to a variety of devices, and *locating objects* is an important problem. Existing approaches can be roughly categorized into three groups: *Centralized Directory Services (CDS)*, *Replicated Directory Services (RDS)*, and the recently emerged *Distributed Directory Services (DDS)*. Given the proliferation of network DoS attacks, we wish to select SCAN a directory service which is resilient to DoS attacks.

However, although the field of "security metrics" has at least a 20-year history involving the production of evaluation criteria, Information Assurance (IA) quantification, risk assessment/analysis methodology development, *etc.*, it has provided neither generally acceptable nor reliable measures for rating IT security or requisite security assurance. For example, there lacks a general methodology to measure the resilience of a system or service to network DoS attacks. As the first step towards this ambitious goal, we propose a simulation-based network DoS resilience benchmark, and apply it to compare the three object location services. We find that DHT-based DDS has the best resilience in practice. Thus we leverage it as the location services for SCAN.

1.3.2 Dynamic Replica Placement and Update Tree Construction (Chapter 4)

Basically, the challenge of replication is to provide good Quality-of-Service (QoS, such as certain latency bound) to clients while retaining efficient and balanced resource consumption of the underlying infrastructure. We proposes a novel algorithm which combines dynamic replica placement with a self-organizing application-level multicast tree to meet client QoS and server resource constraints. Each node (including the root) in the tree only needs to maintain states for its parent and direct children. Simulation results on both flash-crowd-like synthetic workloads and real Web server traces show that SCAN deploys close to an optimal number of replicas, achieves good load balance, and incurs a small delay and bandwidth penalty for update multicast relative to static replica placement on IP multicast.

1.3.3 Scalable Replica Management using Content Clustering (Chapter 5)

SCAN is efficient on the number of replicas deployed. But given the large scale of the World Wide Web, the per-URL-based replica placement computation and management overhead is still overwhelming. Traditional Web caching systems and CDNs do not keep track of the replicas deployed to avoid the management problem, but have to suffer inefficient replication, and consequently poor performance (see Chapter 5.4).

SCAN reduces the replica management overhead by investigating several clustering schemes to group the Web documents and to replicate them in the units of clusters. Evaluations based on various topologies and Web server traces show that without sacrificing users' retrieval cost, the management overhead (in terms of amount of states to maintain and replica location computational cost) is reduced by a factor of 50 - 100. Furthermore, we propose a unique online Web object popularity prediction algorithm based only on hyperlink structures, and applied it for online incremental clustering and replication to adapt to changes in users' access patterns. With this scheme, we pushes new content to the appropriate existing clusters even before accessed. It cuts down the retrieval cost by 4.6 times compared with random replication, and by 8 times compared with no replication. Therefore it is especially useful to improve document availability during flash crowds.

1.3.4 Scalable Overlay Network Monitoring (Chapters 7, 8 and 9)

End-to-end distance monitoring can enable Internet services, like CDN, to construct an efficient overlay mesh, detect network failures, and facilitate dynamic server selection. Existing systems either lack scalability or accuracy, or face problems of deployment. To overcome these shortcomings, we developed a scalable overlay network monitoring system with two components: Internet Iso-bar for latency estimation and TOM (Tomography-based Overlay network Monitoring) for loss rate estimation.

Internet Iso-bar clusters hosts based on the similarity of their perceived network distance to a small number of landmark sites, and chooses the center of each cluster as a monitor site. The distance between two hosts is estimated using inter- or intra-cluster distances. Evaluation using real Internet measurements shows that Internet Iso-bar yields much higher estimation accuracy [2] and stability than the traditional latency estimation systems which are based on network/geographical proximity clustering. Compared with the best known prior work, GNP [83], our accuracy is similar, but the Internet Iso-bar further detect 78% of congestion/failures [3] in real time with 32% false positives because

[2] The accuracy is the relative error between predicted latency and real latency as defined in [136].

[3] See the definition of congestion/failures in Chapter 7.3.2.

GNP has the landmark sites as bottleneck for real-time coordinate update. Such accuracy is suitable for non-business P2P file sharing type of applications.

For mission-critical applications that desire accurate loss rate estimation, previous work require measurement of $O(n^2)$ paths for an overlay network with n end hosts [40]. In contrast, TOM finds a minimal basis set of k linearly independent paths that can fully describe all the $O(n^2)$ paths. It selectively monitors and measures the loss rates of these paths, and then applies them to estimate the loss rates of all other paths. By extensively studying synthetic and real topologies, we find that for reasonably large n (e.g., 100), k is only in the range of $O(n \log n)$. This is explained by the moderately hierarchical nature of Internet routing. Besides, TOM has good measurement load balancing, tolerates topology measurement inaccuracies, and is adaptive to topology changes, such as end host join/leave and routing changes. Both extensive simulation and experiments on the Internet show that we achieve high path loss rate estimation accuracy. We can also continuously update the loss rate estimates online. For example, in the Internet experiments, the average update time is 0.16 second for all 2550 paths, the absolute error of loss rate estimation is 0.0027 and the average error factor is 1.1.

We further demonstrate the effectiveness of the monitoring services with a adaptive overlay streaming media system. In contrast to traditional schemes which treat the underlying network as a best-effort black box and perform adaptations at the transmission end-points, our live streaming media system leverages scalable monitoring services (such as Internet Iso-bar and TOM) for real-time path congestion/failure information, and an overlay network for adaptive packet relaying and buffering within the delivery infrastructure. Specifically, streaming clients in our system employs overlay routing to bypass faulty or slow links and re-establish new connection to streaming servers. Using PlanetLab for Internet testbed, we show that overlay routing can often improve the loss rate and/or TCP throughput for lossy paths, and our system can typically adapt to network congestions within five seconds, achieving skip-free streaming media playback.

The rest of the thesis is organized as the follows. We discuss the CDN-specific techniques, including replica placement, location and management schemes, in Part I, and study the more general overlay measurement and monitoring services in Part II. In each part, we first examine previous work and their limitations, then present our own design, implementation and evaluation.

Part I

Replica Placement, Location and Management

Although quite a few commercial CDNs, like Akamai [4], Digital Island [44] Mirror Image [79] and Speedera [121], have been deployed, several challenges remain as follows.

1. Relying on centralized location services (CDN name servers), the current CDNs cannot keep track of the replicas deployed, have poor scalability and are vulnerable to network DoS attacks.

2. Inefficient replica placement makes it unaffordable to provide replica coherence.

3. Pull-based replica placement causes network congestion and source content server overloading when visited by flash crowds.

In Part I, we first examine related work on these issues (Chapter 2), then present our solutions as three major components of SCAN: replica location (Chapter 3), replica placement and coherence maintenance (Chapter 4) and replica management (Chapter 5).

Chapter 2

Previous Work

In this chapter, we first survey existing content distribution systems, namely Web caching (Chapter 2.1), uncooperative pull-based CDNs (Chapter 2.2), and cooperative push-based CDNs (Chapter 2.3). We compare these systems with SCAN, and summarize this in Table 2.1. Then we discuss the previous work on three building blocks of CDN: object location services (Chapter 2.4), multicast techniques for update dissemination (Chapter 2.5), and content clustering (Chapter 2.6). Finally, we summarize the limitations of previous work in Chapter 2.7.

2.1 Web Caching

Caching can be *client-initiated* or *server-initiated*. Most caching schemes in wide-area, distributed systems are *client-initiated*, such as used by current Web browsers and Web proxies [72]. The problems with both of these solutions are myopic. A client cache does nothing to reduce traffic to a neighboring computer, and a web proxy does not help neighboring proxies. Thus the effectiveness of caching is ultimately limited to the low level of sharing of remote documents among clients of the same site [14]. A possible solution, *server-initiated caching*, allows servers to determine when and where to distribute objects [13, 14, 59]. Essentially, Content Distribution Networks (CDNs, including our approach) are server-initiated caching *with dedicated edge servers*. Previous server-initiated caching systems rely on unrealistic assumptions. Bestavros, *et al.* model the Internet as a hierarchy and any internal node is available as a service proxy [13, 14]. This assumption is not valid because internal nodes are routers, unlikely to be available as service proxies. Geographical push-caching autonomously replicate HTML pages based on the global knowledge of the network topology and clients' access patterns [59]. More recently, adaptive web caching [77] and summary cache [48] are proposed to enable the sharing of caches among Web proxies. Caches exchange content state periodically with other caches, eliminating the delay and unnecessary use of resources of explicit cache probing. However, each proxy server needs to send index update of cached contents to *all* other proxy servers, and needs to store the content indices of *all* other proxy servers. Thus even with compact content index summary like the Bloom filter [48], the state maintenance and exchange overhead is still overwhelming and unscalable with the number of documents and number of cache servers. For instance, the target

Properties	Web caching (client initiated)	Web caching (server initiated)	Uncooperative pull-based CDNs	Cooperative push-based CDNs	**SCAN**
Cache/replica sharing for efficient replication	No, uncooperative	Yes, cooperative	No, uncooperative	Yes, cooperative	**Yes, cooperative**
Scalability for request redirection	No redirection	OK, use Bloom filter [48] to exchange replica locations	Bad, centralized CDN name server	Bad, centralized CDN name server	**Good, decentralized DHT location services**
Granularity of replication	Per URL	Per URL	Per URL	Per Website	**Per cluster**
Distributed load balancing	No	No	Yes	No	**Yes**
Replica coherence	No	No	No	No	**Yes**
Network monitoring for fault-tolerance	No	No	Yes, but unscalable monitoring	No	**Yes, scalable monitoring**

Table 2.1: Comparison of various Internet content delivery systems

number of proxy servers in [48] is only in the order of 100. Furthermore, without dedicated infrastructure like CDN, caching proxies can not adapt to network congestion/failures or provide distributed load balancing.

2.2 Un-cooperative Pull-based CDNs

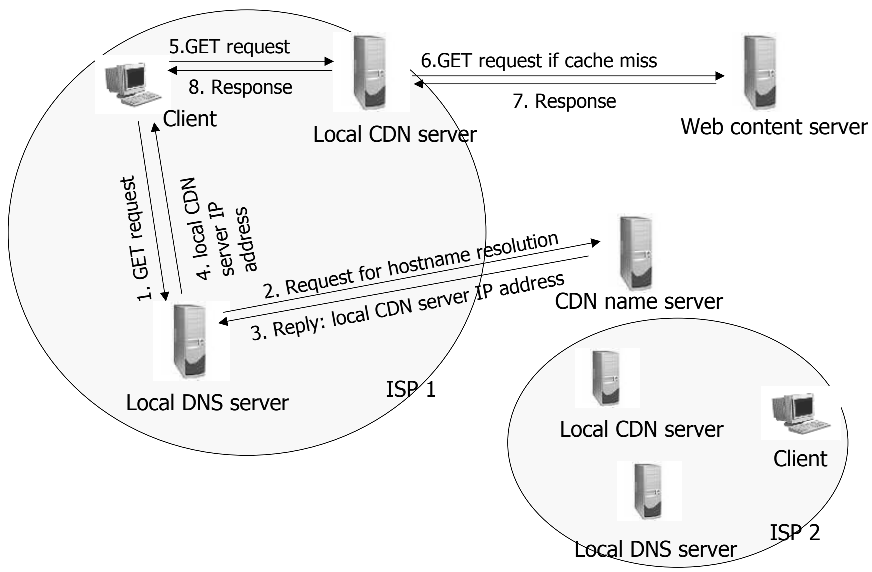


Figure 2.1: Un-cooperative pull-based CDN architecture

Recently, CDNs have been commercialized to provide Web hosting, Internet content and streaming media delivery. Basically, the contents are pulled to the edge servers upon clients' requests. Various mechanisms, such as DNS-based redirection, URL rewriting, HTTP redirection, *etc.*[9], have been proposed to direct client requests for objects to one of the CDN servers (*a. k. a.* CDN nodes or edge servers). Most of the commercial CDN providers, such as Akamai [4], Digital Island [44] Mirror Image [79] and Speedera [121] use DNS-based redirection due to its transparency [68]. Figure 2.1 shows the CDN architecture using DNS-based redirection. Given the rapid growth of CDN service providers, such as Akamai (which already has more than 13,000 servers in about 500 networks around the world [4]), we assume that for each popular clients cluster, there is a CDN server as well as a local DNS server. The client cluster is the group of clients that are topologically close. The clients can be grouped by their BGP prefix [67] or by their local DNS servers. The latter is simple and adopted in practice, but it is not very accurate [74].

Figure 2.1 gives the sequence of operations for a client to retrieve a URL. The hostname resolution request is sent to the CDN name server via local DNS server. Due to the nature of centralized location service, the CDN name server cannot afford to keep records for the locations of each URL replica. Thus it can only redirect the request based on network proximity, bandwidth availability and server load. The CDN server that gets the redirected request may not have the replica. In that case, it will pull a replica from the

Web content server, then reply to the client.

Due to the uncooperative nature, current CDNs often places many replicas than necessary and consumes lots of resources for storage and update. Simulations in Chapter 5.4 reveals that with reasonable latency guarantees, cooperative push-based CDN (defined in Chapter 2.3) only uses a small fractional number of replicas (6-8%) and less than 10% of the update dissemination bandwidth than the uncooperative schemes.

As a research effort, Rabinovich and Aggarwal propose RaDaR, a global Web hosting service with dynamic content replication and migration [105]. However, it requires the DNS to give the complete path from the client to the server, which in practice is often unavailable.

2.3 Cooperative Push-based CDNs

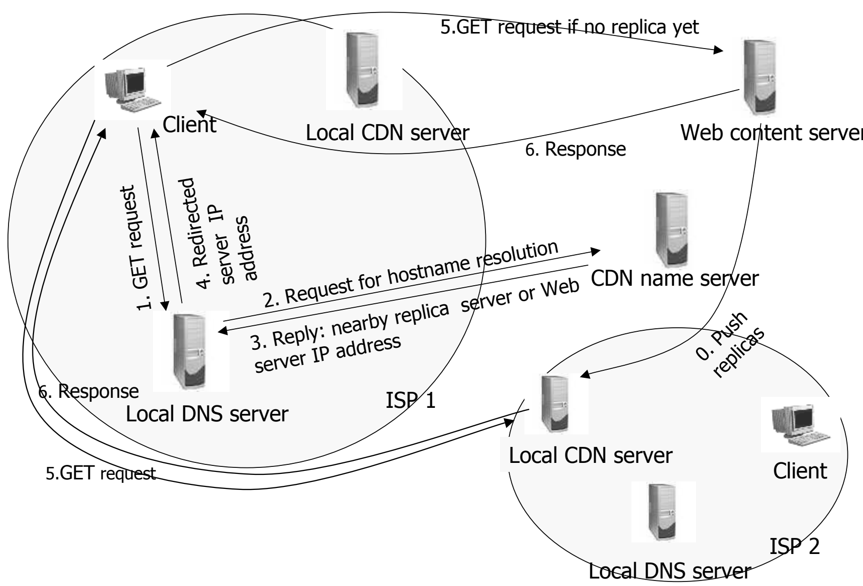


Figure 2.2: Cooperative push-based CDN architecture

Several recent works proposed to pro-actively push content from the origin Web server to the CDN edge servers or proxies according to users' access patterns and global network topology, and have the replicas cooperatively satisfy clients' requests [71, 65, 104, 128].

The key advantage of this cooperative push-based replication scheme over the conventional one does not come from the fact that we use push instead of pull (which only saves compulsory miss), but comes from the cooperative sharing of the replicas deployed. This cooperative sharing significantly reduces the number of replicas deployed, and consequently reduces the replication and update cost, as shown in Chapter 5.4.

We can adopt a similar CDN architecture as shown in Figure 2.2 to support such a cooperative push-based content distribution. First, the Web content server incrementally pushes contents based on their hyperlink structures and/or some access history collected by

CDN name server (Chapter 5.9). The content server runs a "push" daemon, and advertises the replication to the CDN name server, which maintains the mapping between content, identified by the host name in its (rewritten) URL, and their replica locations. The mapping can be coarse (e.g., at the level of Web sites if replication is done in units of Web sites), or fine-grained (e.g., at the level of URLs if replication is done in units of URLs).

With such replica location tracking, the CDN name server can redirect a client's request to its closest replica. Note that the DNS-based redirection allows address resolution on a per-host level. We combine it with content modification (e.g., URL rewriting) to achieve per-object redirection [9]. References to different objects are rewritten into different host names. To reduce the size of the domain name spaces, objects can be clustered as in Chapter 5, and each cluster shares the same host name. Since the content provider can rewrite embedded URLs *a priori* before pushing out the objects, it does not affect the users' perceived latency and the one-time overhead is acceptable. In both models, the CDN edge servers are allowed to execute their cache replacement algorithms. That is, the mapping in cooperative push-based replication is soft-state. If the client cannot find the content in the redirected CDN edge server, either the client will ask the CDN name server for another replica, or the edge server pulls the content from the Web server and replies to the client.

Li, *et al.* approach the proxy placement problem with the assumption that the underlying network topologies are trees, and model it as a dynamic programming problem[71]. While an interesting first step, this approach has an important limitation in that the Internet topology is not a tree. More recent studies [104, 65], based on evaluating real traces and topologies, have independently reported that a greedy placement algorithm can provide content distribution networks with performance that is close to optimal. To simplify the assumption about detailed knowledge of global network topology and clients' distribution, topology-informed Internet replica placement was proposed to place replicas on the routers with big fanout [107]. They show that the router-level topology based replica placement can achieve average client latencies within a factor of 1.1 - 1.2 of the greedy algorithm, but only if the placement method is carefully designed.

2.4 Object Location Systems

Networked applications are extending their reach to a variety of devices and services over the Internet. Applications expanding to leverage these network resources find that *locating objects* on the wide-area is an important problem. Further, the read-mostly model of shared access, widely popularized by the World-Wide-Web, has led to extensive object replication, compounding the problem of object location. Work on location services has been done in a variety of contexts [39, 58, 61, 137]. These approaches can be roughly categorized into the following three groups: *Centralized Directory Services (CDS)*, *Replicated Directory Services (RDS)*, and *Distributed Directory Services (DDS)*.

Extensive work on these directory services have been proposed as we will discuss in more detail in this subsection. However, to the best of our knowledge, there is no attempt to benchmark and contrast their performance.

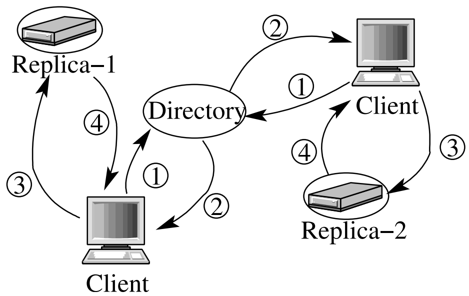


Figure 2.3: A Centralized Directory Service (CDS): Clients contact a single directory to discover the location of a close replica. Clients subsequently contact the replica directly. A Replicated Directory Service (RDS) provides multiple directories.

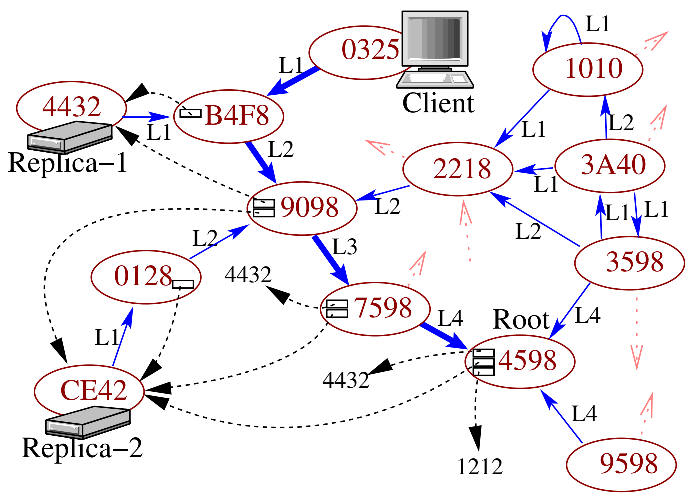


Figure 2.4: A Distributed Directory (Tapestry): Nodes connected via links (solid arrows). Nodes route to nodes one digit at a time: e.g. 1010 → 2218 → 9098 → 7598 → 4598. Objects are associated with one particular "root" node (e.g. 4598). Servers publish replicas by sending messages toward root, leaving back-pointers (dotted arrows). Clients route directly to replicas by sending messages toward root until encountering pointer (e.g. 0325 → $B4F8$ → 4432).

2.4.1 Centralized and Replicated Directory Services

A *centralized directory service* (CDS) resides on a single server and provides location information for every object on the network. See Figure 2.3. Because it resides on a single server, it is extremely vulnerable to DoS attacks. A variant of this is the *replicated directory service* (RDS) which provides multiple directory servers. An RDS provides higher availability, but suffers consistency overhead. Here we do not consider the partitioned directory service because it often requires extra meta directory server for maintaining the partitioning information, such as the root server of DNS.

2.4.2 Distributed Directory Services: the Tapestry Infrastructure

Networking researchers have begun to explore decentralized peer-to-peer location services with distributed hash table (DHT), such as CAN [108], Chord [124], Pastry [112] and Tapestry [137]. Such services offer a distributed infrastructure for locating objects quickly with guaranteed success. Rather than depending on a single server to locate an object, a query in this model is passed around the network until it reaches a node that knows the location of the requested object. The lack of a single target in decentralized location services means they provide very high availability even under attack; the effects of successfully attacking and disabling a set of nodes is limited to a small set of objects.

In addition, Tapestry exploits locality in routing messages to mobile endpoints such as object replicas; this behavior is in contrast to other structured peer-to-peer overlay networks [108, 124, 112]. Thus we leverage on Tapestry to build SCAN.

Tapestry is an IP overlay network that uses a distributed, fault-tolerant architecture to track the location of objects in the network. It has two components: a *routing mesh* and a *distributed location services*.

Tapestry Routing Mesh Figure 2.4 shows a portion of Tapestry. Each node joins Tapestry in a distributed fashion through nearby surrogate servers and set up *neighboring* links for connection to other Tapestry nodes. The neighboring links are shown as solid arrows. Such neighboring links provide a route from every node to every other node; the routing process resolves the destination address one digit at a time. (e.g., ***8 $\Longrightarrow$ **98 $\Longrightarrow$ *598 $\Longrightarrow$ 4598, where *'s represent wildcards). This routing scheme is based on the hashed-suffix routing structure originally presented by Plaxton, Rajaraman, and Richa [103].

Tapestry Distributed Location Service Tapestry assigns a globally unique name (GUID) to every object. It then deterministically maps each GUID to a unique *root* node. Storage servers *publish* objects by sending messages toward the roots, depositing *location pointers* at each hop. Figure 2.4 shows two replicas and the Tapestry root for an object. These mappings are simply pointers to the server s where o is being stored, and not a copy of the object itself. Thus for nearby objects, client search messages quickly intersect the path taken by publish messages, resulting in quick search results that exploit locality. It is shown in [103] that the average distance travelled in locating an object is *proportional* to the distance from that object.

2.5 Multicast for Disseminating Updates

For update dissemination, IP multicast has *fundamental* problems as the architectural foundation for Internet distribution. For instance, it works only across space, not across time, while most content distribution on the Internet works across both [50]. Further, there is no widely available inter-domain IP multicast.

As an alternative, many application-level multicast systems have been proposed [50, 32, 52, 98, 23, 138]. Among them, [32, 23, 98] target small group, multi-source applications, such as video-conferencing, while [50, 52, 138] focus on large-scale, single-source applications, such as streaming media multicast. Bayeux [138] is also built on top of Tapestry. It uses the Tapestry location service to find the multicast root(s), and then uses Tapestry routing to route both the control (e.g., "join") and data messages. In contrast, we only use the Tapestry location mechanism to find the nearby replica.

Most ALM systems have scalability problems, since they utilize a central node to maintain states for all existing children [32, 52, 98, 23], or to handle all "join" requests [138]. Replicating the root is the common solution [52, 138], but this suffers from consistency problems and communication overhead. On the other hand, Scribe [113] and the update multicast system of SCAN (namely dissemination tree, see Chapter 4) leverage peer-to-peer routing and location services, and do not have the scalability problem. Scribe is a large-scale event notification system, using overlay DHT for both subscription and dissemination. The dissemination tree is more efficient because we use overlay DHT only for subscription, and use IP for dissemination directly.

2.6 Content Clustering

There is considerable work done in data clustering, such as K-means [66], HAC [129], CLANRNS [82], *etc.*. In the Web research community, there have been many interesting research studies on clustering Web content or identifying related Web pages for various purposes, such as pre-fetching, information retrieval, and Web page organization, *etc.*. Cohen, *et al.* [37] investigated the effect of content clustering based on temporal access patterns and found it effective in reducing latency, but they considered a single server environment and didn't study the more accurate spatial clustering. Padmanabhan and Mogul [93] proposed a pre-fetching algorithm using a dependency graph. When a page A is accessed, clients will pre-fetch a page B if the arc from A to B has a large weight in the dependency graph. Su, *et al.* proposed a recursive density-based clustering algorithm for efficient information retrieval on the Web [125]. As in the previous work, our content clustering algorithms also try to identify groups of pages with similar access pattern. Unlike many previous works, which are based on analysis of individual client access patterns, we are interested in aggregated clients' access patterns, since content is replicated for aggregated clients. Also, we quantify the performance of various cluster-based replications by evaluating their impact on replication.

Moreover, we examine the stability of content clusters using *incremental clustering*. Incremental clustering has been studied in previous work, such as [21] and [135]. However, to the best our knowledge, none of the previous work looks at incremental clustering as a way to facilitate content replication and improve the access performance perceived by clients. We are one of the first to examine clustering Web content for efficient replication, and use both replication performance and stability as the metrics for evaluation of content clustering (Chapter 5).

2.7 Summary

In summary, we find that previous work on CDN and its related techniques have the following limitations.

1. Client-initiated web caching is myopic, while the server-initiated web caching has unscalable content state exchange overhead. Neither can adapt to network congestion/failures or provide distributed load balancing.

2. CDNs rely on centralized location services, thus they have to either apply inefficient and pull-based replication (uncooperative CDN), or replicate at the granularity of per Website and sacrifice the performance to clients (cooperative CDN).

3. There is no performance or DoS attack resilience benchmark for existing location services. This makes it difficult to compare the alternative proposals.

4. No coherence to replicas/caches: IP multicast doesn't exist in the Internet, while the existing application-level multicast has scalability problem.

5. Content clustering is based on individual client access patterns, and thus not suitable for shared, cooperative access of CDNs.

In SCAN, the first two limitations are addressed with distributed location services, Tapestry, and we propose a network DoS resilience benchmark to contrast its performance with other alternatives (Chapter 3). For limitation 4, we dynamically place replicas and self-organize them into a scalable application-level multicast tree to disseminate updates (Chapter 3). The last limitation is tackled with (incremental) clustering of Web contents based on aggregated access patterns for better replica management scalability (Chapter 5).

Chapter 3

Performance Comparison of Object Location Systems: A Case Study of Network Denial of Service (DoS) Attack Resilience

Traditional CDNs redirect clients' requests only based on the network proximity, regardless of where the replicas of requested object have been deployed. This is because their centralized CDN name servers cannot afford to keep track of the replica locations, which leads to inefficient replication (see Chapter 5.4). Furthermore, it is a common belief that such centralized approach has poor scalability and is vulnerable to network DoS attacks.

On the other hand, there are emerging replicated and distributed directory services (Chapter 2.4). In this chapter, we seek to compare their performance and choose the best one for SCAN. In particular, we focus on contrasting the network DoS attack resilience, which also sheds light on the scalability performance of these services. We will start with motivation (Chapter 3.1), then the threat model (Chapter 3.2) and benchmark setup (Chapter 3.3). Finally, We will contrast the DoS resilience of these architectures (Chapter 3.4).

3.1 Motivation

Network DoS attacks are increasing in frequency, severity and sophistication, making it desirable to measure the resilience of systems to DoS attacks. From 1989-1995 the number of DoS attacks increased 50% per year [60]. Additionally, a 1999 CSI/FBI survey reported that 32% of respondents detected DoS attacks directed at their systems [62]. To make things worse, automatic attack tools (such as Tribal Flood Network(TFN), TFN2K, Trinoo and stacheldrant) allow teenagers to launch widely distributed DoS attack with a few keystrokes (so called "script kiddies") [38]. Given the proliferation of DoS attacks, many mission-critical applications claim DoS resilience. To test these claims, there is a desire for a general methodology to measure the resilience of a system or service to network DoS attacks.

DoS attacks are difficult to analyze because they are system-wide phenomena. Viewing components or attackers in isolation often fails to expose interesting behavior. As a consequence, we choose to observe a simulation of a complete system, including realistic network topology, client workloads, server architecture, and attack profiles. Chapter 3.3 will describe the simulation environment in detail. Here we wish to understand the *types of attacks* that might be mounted against object location services and how we can *assess their impact*.

Our work is the first attempt towards benchmarking DoS attacks for arbitrary network services. In [116], the authors investigated several approaches to fighting TCP SYN attacks and developed a tool which actively monitored the network for suspicious attack behavior and terminated dangling connections left by the attacker. In [120], the authors describe the use of an end-to-end resource accounting in the Scout operating system to protect against resource-based DoS attacks. Both these works present microbenchmarks testing the effectiveness of the proposed countermeasure. Our approach differs partly in that we investigate attacks on availability of a service, rather than on a particular server.

Brown and Patterson [17] investigate the use of fault injection to benchmark availability and apply their methodology to software RAID systems. Our work is similarly based on injecting faults into a workload and investigating the effect, but our faults are malicious in nature.

3.2 Threat Models

Denial of Service attacks come in many shapes and sizes. In fact, the CERT Coordination Center [20] has proposed the following taxonomy:

- Consumption of network connectivity and/or bandwidth
- Consumption of other resources, *i.e.*, CPU cycles or kernel data structures
- Destruction or alteration of configuration information
- Physical destruction or alteration of network components

Specializing this set for object location services, we identify two general classes of attack: *Flooding Attacks* and *Corruption Attacks*:

3.2.1 Flooding Attacks

The most popular network DoS attack is the flooding attack, in which the attacker sends superfluous requests at a high rate. Flooding attacks overload the victim's resources (such as queues and CPU), and also swamp the local routers, gateways and links. These DoS attacks can be classified as *point-to-point* or *distributed*. There are four major point-to-point DoS attacks: TCP SYN flooding, UDP flooding, ICMP flooding and Smurf attacks [43].

Distributed DoS (DDoS) attacks combine point-to-point DoS attacks with distributed and coordinated control. Figure 3.1 shows the structure of a DDoS attack, with

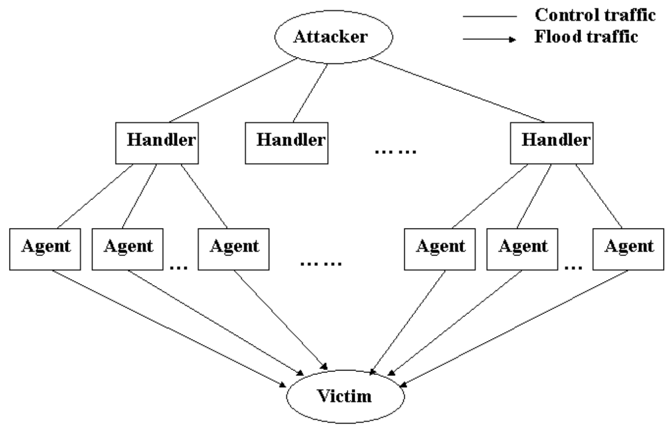


Figure 3.1: Structure of a distributed DDoS attacks

one or more *attackers* controlling *handlers*, with each handler controlling multiple *agents*[1]. Handlers and agents are extra layers introduced to increase the rate of packet traffic as well as hide the attackers from view. Each agent can choose the size and type of packets as well as the duration of flooding. While the victim may be able to identify some agents and have them taken off-line, the attacker can monitor the effects of the attack and create new agents accordingly [43]. In general, attack simulation parameters should be chosen to cover a sufficient spectrum of attack traffic versus legitimate traffic to show interesting results.

3.2.2 Corruption Attacks

When an attacker corrupts or destroys information, we call this a *corruption attack*. There are numerous variants on this type of attack. For instance an attacker might alter configuration information to prevent the use of a computer or network. Or, an attacker might corrupt routing tables, causing victim nodes to redirect traffic toward the attacker, which would subsequently drop or deny requests. It is not possible to test all attacks, so typical examples of this category should be simulated and measured.

3.2.3 Measuring Resilience

DoS attacks reduce resource *availability*. Here, availability refers to a spectrum of service quality, not simply "up" versus "down". Though the choice of Quality of Service (QoS) metrics depends on the system or service being studied, Brown and Patterson have suggested *performance, completeness, accuracy* and *capacity* as starting points [17]. For our particular study, we consider metrics of *response latency*, *request throughput*, and *time to recover*[2]. We examine the level degradation of a service under attack to assess the *resilience* of that service.

[1] Compromised hosts responsible for generating packet streams directed at the victim.

[2] A corrupted directory service could prevent service entirely, but this is beyond the scope of the current study.

Of course, Denial of Service is *multidimensional* in that system A may be more resilient than system B for one type of attack but less resilient for another. Usually, the particular threat-model under consideration defines a set of dimensions, one for each class of threat. Combining these dimensions to yield a particular *resilience ranking* is a very system-specific task and hard to generalize. Our solution is to be sufficiently specific in the definition of the threat model and only quantify the resilience in that model.

3.3 Experimental Setup

We built a complete system on top of *ns*2 simulator [16]. All of our nodes function as both clients and hosts with a subset providing the directory service. Clients send lookup requests to the directory service, which either returns the location of a replica or forwards the request directly to the replica. We selected some nodes to be attackers and measured changes in the availability of system resources.

We used 1000 node network topologies generated by GT-ITM [134] using a transit-stub model. Each graph is made up of five transit domains. These domains are guaranteed to be connected. Each transit domain consists of an average of eight stub networks. The stub networks contain edges amongst themselves with a probability of 0.5. Each stub network consists of an average of 24 nodes, in which nodes are once again connected with a probability of 0.5. We then extended these topologies with common network bandwidths as recommended in [52]. Our routers use simple drop-tail queuing with the default NS queue size 50 (we assumed attackers will spoof their IP addresses, defeating any filtering done by more complicated queuing policies).

3.3.1 Client Operation

We generated synthetic client workloads using both Zipf's law [5] and hot-cold [106] models. Zipf's law states that if objects are ranked according to their access frequency, then the number of requests of the object with rank i is proportional to $1/i$. In a hot-cold model, a small portion of the objects (10%) receive the majority (90%) of the requests. Our network has 500 objects, each with three replicas placed on three randomly chosen nodes. The sizes of objects were chosen randomly from the interval 5KB - 50KB. Nodes request a data object, wait for the data and then request another, such as when a user is following a series of web links.

3.3.2 Directory Server Operation

We used five different directory services in our simulations:

CDSr The simplest directory service is the *Centralized Directory Server*(CDS). Here, one non-transit node is chosen to be the directory server. Object requests are made in two stages. First, the directory server is queried and returns the location of a random replica of the object. Second, the requesting node communicates directly with the node hosting the replica and the data is returned.

CDSo Same as above, except that the directory server returns the location of the replica which is closest to the requesting node.

RDSr The *Replicated Directory Service*(RDS) is placed on four random, widely-distributed, non-transit nodes. Queries are made as above, except that a node must choose one of the servers to fulfill its request. Here, the choice is made randomly for each request. The replica is also randomly chosen by the directory server as in the CDSr.

RDSo Same as the RDSr, except that each node sends requests to the nearest directory server. (Replica choice is still random).

DDS For the DDS, we implemented a simplified version of Tapestry as an extension to *ns*. All messages between nodes are passed by *ns*'s TCP/IP agent. Messages route through the object's tree to the statistically closest object replica, and the replica responds by sending the data contents directly to the requesting node. Our Tapestry data structures are statically built at the start of the simulation using full knowledge of the topology, and using hop count as the network distance metric. It should also be noted that our implementation is un-optimized and is likely slower than a real implementation would be. On the other hand, the static configuration of Tapestry may overlook the effect of dynamic node failures on its DDoS resilience. It is our future work to address that issue.

3.3.3 The Attacks

We modeled two types of attacks in our simulations:

Flooding Attacks

The first attacks we simulated flood some important node(s) and overload their queues to reduce the number of legitimate requests that get through. We randomly designated some nodes "agents"; the agents then stream a constant bit rate at the victim. We varied the number of agents as well as the severity (bit rate) of flooding. The life time of each agent was randomly chosen from 0 - 200 seconds with new agents immediately replacing those taken off-line.

For the CDS and RDS, we attacked the directory server(s). We attacked the closest analogy in Tapestry, the root of a hot object. For comparison with the CDS (RDS), we flood the root of one (four) hot object(s), keeping the number of attacked nodes the same.

Corruption Attacks

As these attacks are system/service-specific, we only simulated two attacks here as examples. This is by no means to be an exhaustive list of such attacks. We just discuss two examples as representatives.

The first attack forces an important node to believe there is a link with negligible latency between the nodes which are actually the farthest apart. We attack the directory server of the CDS, a random directory server of the RDS and the Tapestry root node of a hot object for comparison.

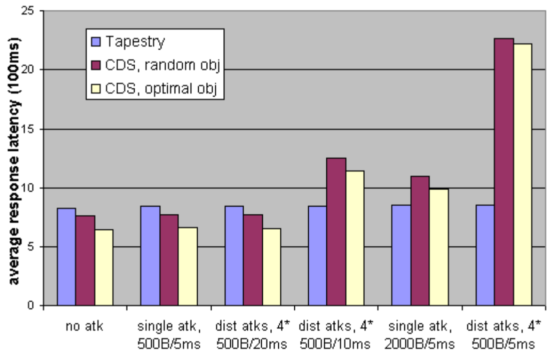


Figure 3.2: Average response latency of CDS vs. Tapestry under DoS flooding attacks

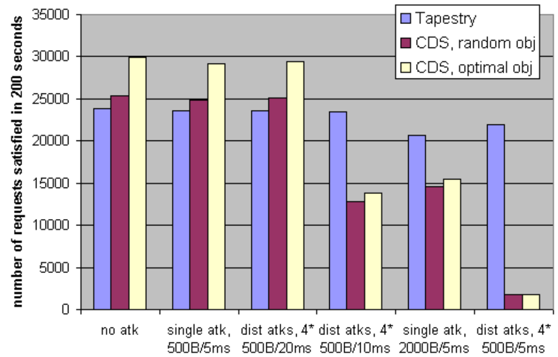


Figure 3.3: Throughput of CDS vs. Tapestry under DoS flooding attacks

The second attack is specific to Tapestry; a malicious Tapestry node claims to be the root node of all objects. By replying with a negative result to any request it receives, this node can potentially convince clients that requested objects do not exist, denying them access to an existing resource. The question we ask here is "how many nodes are affected?"

3.4 Results

3.4.1 Flooding Attacks

We performed simulations of flooding attacks on the CDS, RDS, and Tapestry with hot-cold and Zipf's law workloads. The results were similar for both workloads, so we present only hot-cold results.

Comparison of CDS and Tapestry

First, we compare the performance of CDS with Tapestry. We simulated *one attacker* at a rate of 500 or 2000 bytes every 5 ms or *four attackers* at rates between 500 bytes every 20ms and 500 bytes every 5ms. The results are shown in Figures 3.2 and 3.3. These figures reveal that a single attacker does not significantly influence performance, while distributed attackers, each flooding at the same high rate, cause severe denial of service.

While a CDS suffers greatly under severe attacks, Tapestry shows some resistance. This can be explained by the distributed nature of Tapestry. For each simulation, we normalize the throughput under various attacks vs. normal throughput, and plot it as Figure 3.4. The normalized graph shows that the throughput of CDS drops to 6-7%, while Tapestry remains over 90% of throughput under the most severe attacks. This is because Tapestry satisfies many requests even before they reach the root.

One interesting observation, as shown by the two rightmost sets of data in Figures 3.2 and 3.3, is that distributed attackers cause more severe DoS than a single attacker, *even when injecting the same amount of flood traffic*. The reason for this is that the DoS

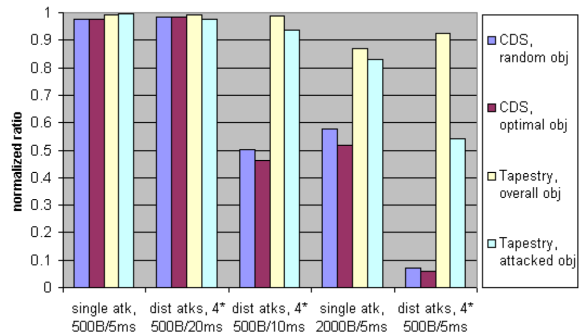


Figure 3.4: Normalized throughput of CDS vs. Tapestry under DoS flooding attacks

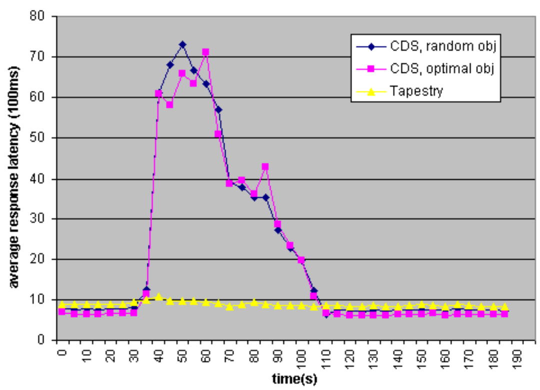


Figure 3.5: Dynamics of average response latency of CDS vs. Tapestry under DoS flooding attacks

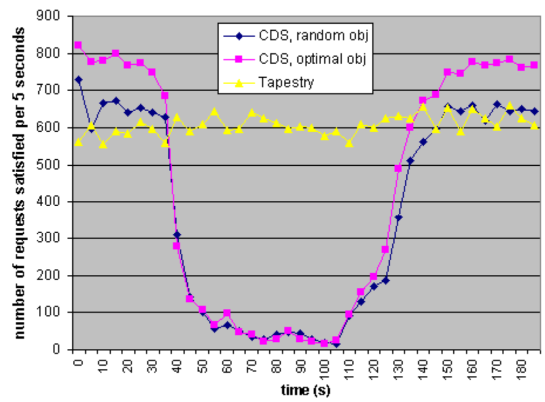


Figure 3.6: Dynamics of throughput of CDS vs. Tapestry under DoS flooding attacks

attack power of single attacker is limited by the bottleneck bandwidth from the attacker to the victim.

Figures 3.5 and 3.6 show the dynamics of the most severe flooding attacks on CDS and Tapestry. The attack(s) start at 40 seconds and end at 110 seconds. Given our simulation setup, the *time to recover* for CDS with both policies is 40 seconds. As Tapestry is not really affected much, its *time to recover* is negligible.

Comparison of RDS and Tapestry

To explore a replicated directory service, we put four servers on widely-distributed, non-transit nodes. We investigated two policies: either the client contacts a random directory server (RDSr) or the closest one (RDSo). We did not simulate consistency traffic between directories.

Again, the single flooding attack has little effect, so we only present results of DDoS attacks in Figure 3.7 and 3.8. We randomly selected four non-transit nodes as attackers. Each of these nodes attacks a directory server in a different subnet or the DDS root of a hot object; these attacks have little effect. We also randomly selected sixteen non-transit

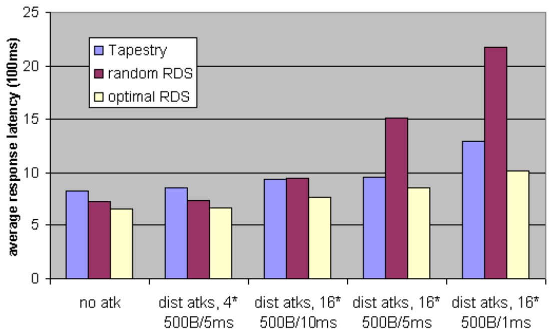


Figure 3.7: Average response latency of RDS vs. Tapestry on DDos flooding attacks

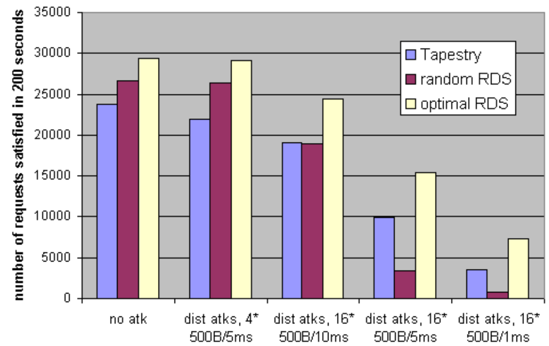


Figure 3.8: Throughput of RDS vs. Tapestry on DDos flooding attacks

attack agents in groups of four, each from different subnets. Each group attacked one RDS directory server or the DDS root of a hot object. The attack rate varied from 500 bytes every 10ms to 500 bytes every 1ms, with each agent set to the same rate.

Both forms of RDS and Tapestry are far more resilient to DoS than CDS (observe the difference in flooding rates along the X-axes). Thus, replication and topology-aware locality can significantly increase resilience to DoS attacks. In our simulations, the optimal RDS always performs better than Tapestry. This is because Tapestry may be forced to make traverse bottleneck links multiple times, whereas the clients in the same subnet as an RDS directory server can avoid the bottlenecks entirely. A more interesting observation, however, is that Tapestry comes very close to optimal RDS; as the number of objects and size of network increases, the number of replicated directory servers required to compete with the self-organizing nature of Tapestry is likely to increase, making Tapestry a better overall choice. Meanwhile, Tapestry outperforms the random RDS on severe attacks, lending credence to the locality properties of Tapestry.

3.4.2 Corruption Attacks

When we compromised routing information at important nodes, the CDS and RDS, which access a random replica, are not affected[3]. The performance of the CDS which returns the optimal replica was degraded to 85%. The impact to Tapestry is negligible, with overall performance reduced by only 2.2%. We also simulated the Tapestry-specific node spoofing attack. The effects of the attack are displayed in Figure 3.9. The attack affects 24% of the network.

3.4.3 Resiliency Ranking

How might we combine the results of previous sections into a single ranking? As Denial of Service is multidimensional (Chapter 3.2.3), we might assign weights to different types of attacks based on perceived severity or frequency. For instance, if we assign 80% weight to flooding attacks and 10% each to two "corruption" attacks, we can roughly rank

[3]We assume that the directory server(s) are not routers or gateways.

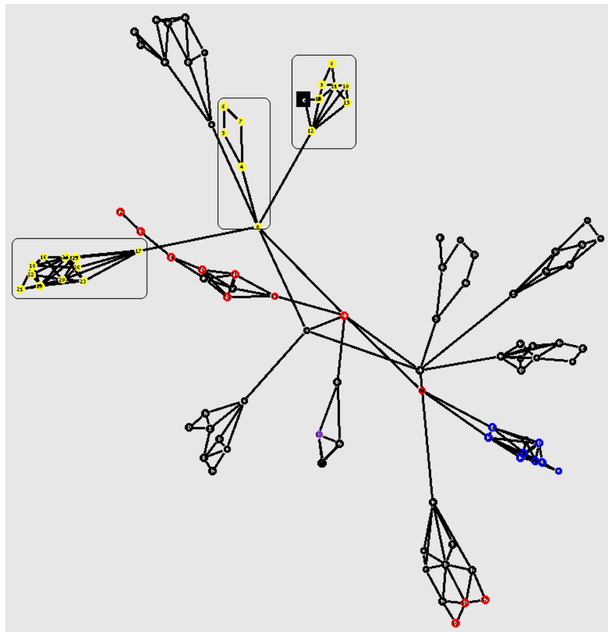


Figure 3.9: Nodes accessing each replica of an attacked object. Neighbor table corruption at the black square node renders all nodes enclosed by round-corner rectangles unable to locate the object. Simulation of 100 nodes and 60 objects (15% hot).

the directory services as in Table 3.1. It is our future work to automate the weight generation more systematically (Chapter 10.2).

Here we simulate all eight attacks in Figures 3.2, 3.3, 3.7 and 3.8 for all three types of directory services and report a weighted sum of normalized throughputs. The weights are assigned in proportion to the amounts of flood traffic and the normalization is based on the corresponding directory service performance without attack; this will vary from system to system, but does give an idea how these services differ in terms of DoS resilience.

From Table 3.1, RDS with the optimal directory server has the best resilience. However, the optimal directory server requirement is not very realistic. Thus we choose DDS, and in particular, Tapestry, for our location services. In the next chapter, we will

Directory services	Flooding attack (80%)	Corruption attack (10%)	Node spoofing attack (10%)	Total score	Rank
CDS, random replica	0.027	N/A	N/A	0.2216	4
CDS, optimal replica	0.023	0.85	N/A	0.2034	5
RDS, random dir server	0.17	N/A	N/A	0.336	3
RDS, optimal dir server	0.48	N/A	N/A	0.584	1
DDS (Tapestry)	0.35	0.978	0.76	0.4538	2

Table 3.1: Attempting to rank the five different directory services. "N/A" means that the attack is not applicable to the service, and we give it a score of 1.0.

discuss how to leverage Tapestry to dynamically place replicas and build application-level multicast tree for update dissemination.

Chapter 4

Dynamic Replica Placement

As shown in Figure 1.2, replica placement is a key component of SCAN. According to users' requests, it dynamically places a minimal number of replicas while meeting client QoS and server capacity constraints. The location services discussed in last chapter are notified about the new replicas via Tapestry *PUBLISHOBJECT* API [137]. The replicas are grouped into clusters to reduce the management overhead, and network-level adaptation is enabled via overlay monitoring services (to be covered in the next few chapters).

In this chapter, we will study replica placement, and how these replicas are self-organized into a application-level multicast tree (termed as *dissemination tree*, or *d-tree*) with small delay and bandwidth consumption for update dissemination. The important intuition here is that the presence of the DOLR (distributed overlay location and routing) system enables simultaneous placement of replicas and construction of a d-tree without contacting the data source. As a result, each node in a d-tree must maintain state only for its parent and direct children. Simulation results on both flash-crowd-like synthetic workloads and real Web server traces show that SCAN meets the the design goals.

For the rest of this chapter, we will first formulate the replica placement problem in Chapter 4.1, then present our algorithms in Chapter 4.2, evaluation methodology in Chapter 4.3 and evaluation results in Chapter 4.4.

4.1 Problem Formulation

There is a large design space for modelling Web replica placement as an optimization problem and we describe it as follows. Consider a popular Web site or a CDN hosting server, which aims to improve its performance by pushing its content to some hosting server nodes. The problem is to dynamically decide where content is to be replicated so that some objective function is optimized under a dynamic traffic pattern and set of clients' QoS and/or resource constraints. The objective function can either minimize clients' QoS metrics, such as latency, loss rate, throughput, *etc.*, or minimize the replication cost of CDN service providers, e.g., network bandwidth consumption, or an overall cost function if each link is associated with a cost. For Web content delivery, the major resource consumption in replication cost is the network access bandwidth at each Internet Data Center (IDC) to the backbone network. Thus when given a Web object, the cost is linearly proportional to the number of replicas.

As Qiu, *et al.* tried to minimize the total response latency of all the clients' requests with the number of replicas as constraint [104], we tackle the replica placement problem from another angle: minimize the number of replicas when meeting clients' latency constraints and servers' capacity constraints. Here we assume that clients give reasonable latency constraints as it can be negotiated through a service-level agreement (SLA) between clients and CDN vendors. Thus we formulate the Web content placement problem as follows. Given a network G with C clients and S server nodes, each client c_i has its *latency constraint* d_i, and each server s_j has its load/bandwidth/storage *capacity constraint* l_j. The problem is to find a smallest set of servers S' such that the distance between any client c_i and its "parent" server $s_{c_i} \in S'$ is bounded by d_i. More formally, find the minimum K, such that there is a set $S' \subset S$ with $|S'| = K$ and $\forall\ c \in C$, $\exists\ s_c \in S'$ such that distance$(c,\ s_c) \leq d_c$. Meanwhile, these clients C and servers S' self-organize into an application-level multicast tree with C as leaves and $\forall\ s_i \in S'$, its fan-out degree (i.e., number of direct children) satisfies $f(s_i) \leq l_i$.

4.2 Replica Placement Algorithms

The presence of an underlying DOLR with routing locality can be exploited to perform simultaneous replica placement and tree construction. Every SCAN server is a member of the DOLR. Hence, new replicas are published into the DOLR. Further, each client directs its requests to its proxy SCAN server; this proxy server interacts with other SCAN servers to deliver content to the client.

Although we use the DOLR to locate replicas during tree building, we otherwise communicate through IP. In particular, we use IP between nodes in a d-tree – parents and children keep track of one another. Further, when a client makes a request that results in placement of a new replica, the client's proxy keeps a cached pointer to this new replica. This permits direct routing of requests from the proxy to the replica. Cached pointers are soft state since we can always use the DOLR to locate replicas.

4.2.1 Goals for Replica Placement

Replica placement attempts to satisfy both *client latency* and *server load* constraints. *Client latency* refers to the round-trip time required for a client to read information from the SCAN system. We keep this within a pre-specified limit. *Server load* refers to the communication volume handled by a given server. We assume that the load is directly related to the number of clients it handles and number of d-tree children it serves. We keep the load below a specified maximum. Our goal is to meet these constraints while minimizing the number of deployed replicas, keeping the d-tree balanced, and generating as little traffic during update as possible. Our success will be explored in Chapter 4.4.

4.2.2 Dynamic Placement

Our dynamic placement algorithm proceeds in two phases: *replica search* and *replica placement*. The replica search phase attempts to find an existing replica that meets the client latency constraint without being overloaded. If this is successful, we place a link in the client and cache it at the client's proxy server. If not, we proceed to the replica placement phase to place a new replica.

Replica search uses the DOLR to contact a replica "close" to the client proxy; call this the *entry* replica. The locality property of the DOLR ensures that the entry replica is a reasonable candidate to communicate with the client. Further, since the d-tree is connected, the entry replica can contact all other replicas. We can thus imagine three search variants: *Singular* (consider only the entry replica), *Localized* (consider the parent, children, and siblings of the entry replica), and *Exhaustive* (consider all replicas). For a given variant, we check each of the included replicas and select one that meets our constraints; if none meet the constraint, we proceed to place a new replica.

procedure DynamicReplicaPlacement_Naive(c, o)
1 c sends *JOIN* request to o through DOLR, reaches entry server s. Request collects $\mathrm{IP}_{s'}$, $dist_{overlay}(c, s')$ and $rc_{s'}$ for each server s' on the path.
2 **if** $rc_s > 0$ **then**
    if $dist_{overlay}(c, s) \leq d_c$ **then** s becomes parent of c, **exit**.
    else
3   s pings c to get $dist_{IP}(c, s)$.
4   **if** $dist_{IP}(c, s) \leq d_c$ **then** s becomes parent of c, **exit**.
    end
end
5 At s, choose s' on path with $rc_{s'} > 0$ *and* smallest $dist_{overlay}(t, c) \leq d_c$
if $\nexists$ *such* s' **then**
6   for each server s' on the path, s collects $dist_{IP}(c, s')$ and chooses s' with $rc_{s'} > 0$ *and* smallest $dist_{IP}(t, c) \leq d_c$.
end
7 s puts a replica on s' and becomes its parent, s' becomes parent of c.
8 s' publishes replica in DOLR, **exit**.

Algorithm 1: Naive Dynamic Replica Placement. Notation: Object o. Client c with latency constraint d_c. Entry Server s. Every server s' has remaining capacity $rc_{s'}$ (additional children it can handle). The overlay distance ($dist_{overlay}$(x,y)) and IP distance ($dist_{IP}$(x,y)) are the round trip time (RTT) on overlay network and IP network, separately.

We restrict replica placement to servers visited by the DOLR routing protocol when sending a message from the client's proxy to the entry replica. We can locate these servers without knowledge of global IP topology. The locality properties of the DOLR suggest that these are good places for replicas. We consider two placement strategies: *Eager* places the replica as close to the client as possible and *Lazy* places the replica as far from the client as possible. If all servers that meet the latency constraint are overloaded, we replace an old replica; if the entry server is overloaded, we disconnect the oldest link among its d-trees.

procedure DynamicReplicaPlacement_Smart(c, o)

c sends *JOIN* request to o through DOLR, reaches entry server s

From s, request forwarded to children (sc), parent (p), and siblings (ss)

Each family member t with $rc_t > 0$ sends rc_t to c. c measures $dist_{IP}(t, c)$ through TCP three-way handshaking.

if $\exists$ t *and* $dist_{IP}(t, c) \leq d_c$ **then**

  $c$chooses t as parent with biggest rc_t and $dist_{IP}(t, c) \leq d_c$, **exit**.

else

  c sends *PLACEMENT* request to o through DOLR, reaches entry server s

  Request collects IP$_{s'}$, $dist_{overlay}(c, s')$ and $rc_{s'}$ for each server s' on the path.

  At s, choose s' on path with $rc_{s'} > 0$ *and* largest $dist_{overlay}(t, c) \leq d_c$

  if $\nexists$ *such* s' **then**

    for each server s' on the path, s collects $dist_{IP}(c, s')$ and chooses s' with $rc_{s'} > 0$ *and* largest $dist_{IP}(t, c) \leq d_c$.

  end

  s puts a replica on s' and becomes its parent, s' becomes parent of c.

  s' publishes replica in DOLR, **exit**.

end

Algorithm 2: Smart Dynamic Replica Placement. Notation: Object o. Client c with latency constraint d_c. Entry Server s. Every server s' has remaining capacity $rc_{s'}$ (additional children it can handle). The overlay distance ($dist_{overlay}$(x,y)) and IP distance ($dist_{IP}$(x,y)) are the round trip time (RTT) on overlay network and IP network, separately.

Dynamic Techniques

We can now combine some of the above options for search and placement to generate dynamic replica management algorithms. Two options that we would like to highlight are as follows.

- *Naive Placement:* A simple combination utilizes *Singular* search and *Eager* placement. This heuristic generates minimal search and placement traffic.
- *Smart Placement:* A more sophisticated algorithm is shown in Algorithm 4. This algorithm utilizes *Localized* search and *Lazy* placement.

Note that we try to use the overlay latency to estimate the IP latency in order to save "ping" messages. Here the client can start a daemon program provided by its CDN service provider when launching the browser so that it can actively participate in the protocols. The locality property of Tapestry naturally leads to the locality of d-tree, i.e., the parent and children tend to be close to each other in terms of the number of IP hops between them. This provides good delay and multicast bandwidth consumption when disseminating updates, as measured in Chapter 4.4. The tradeoff between the naive and smart approaches is that the latter consumes more "join" traffic to construct a tree with fewer replicas, covering more clients, with less delay and multicast bandwidth consumption. We evaluate this tradeoff in Chapter 4.4.

Static Comparisons

The replica placement methods given above are unlikely to be optimal in terms of the number of replicas deployed, since clients are added sequentially and with limited knowledge of the network topology. In the static approach, the root server has complete knowledge of the network and places replicas *after* getting all the requests from the clients. In this scheme, updates are disseminated through IP multicast. Static placement is not very realistic, but may provide better performance since it exploits knowledge of the client distribution and global network topology.

The problem formulated in Chapter 4.1 can be converted to a special case of the capacitated facility location problem [64] defined as follows. Given a set of locations i at which facilities may be built, building a facility at location i incurs a cost of f_i. Each client j must be assigned to one facility, incurring a cost of $d_j c_{ij}$ where d_j denotes the demand of the node j, and c_{ij} denotes the distance between i and j. Each facility can serve at most l_i clients. The objective is to find the number of facilities and their locations yielding the minimum total cost.

To map the facility location problem to ours, we set f_i always 1, and set c_{ij} 0 if location i can cover client j or ∞ otherwise. The best approximation algorithm known today uses the primal-dual schema and Lagrangian relaxation to achieve a guaranteed factor of 4 [64]. However, this algorithm is too complicated for practical use. Instead, we designed a greedy algorithm that has a logarithmic approximation ratio.

Besides the previous notations, we define the following variables: set of covered clients by s: C_s, $C_s \subseteq C$ and $\forall\ c \in C_s$, $dist_{IP}(c, s) \leq d_c$; set of possible server parents for client c: S_c, $S_c \subseteq S$ and $\forall\ s \in S_c$, $dist_{IP}(c, s) \leq d_c$.

procedure ReplicaPlacement_Greedy_DistLoadBalancing(C, S)
input : Set of clients to be covered: C, total set of servers: S
output: Set of servers chosen for replica placement: S'
while *C is not empty* **do**
  Choose $s \in S$ which has the largest value of min(cardinality $|C_s|$, remaining capacity rc_s)
  $S' = S' \bigcup \{s\}$
  $S = S$ - $\{s\}$
  if $|C_s| \leq rc_s$ **then** $C = C$ - C_s
  else
    Sort each element $c \in C_s$ in increasing order of $|S_c|$
    Choose the first rc_s clients in C_s as $C_{sChosen}$
    $C = C$ - $C_{sChosen}$
  end
  recompute S_c for $\forall\ c \in C$
end
return S'.

Algorithm 3: Static Replica Placement with Load Balancing

We consider two types of static replica placement:

- *IP Static:* The root has global IP topology knowledge.
- *Overlay Static:* For each client c, the root only knows the servers on the Tapestry path from c to the root which can cover that client (in IP distance).

The first of these is a "guaranteed-not-to-exceed" optimal placement. We expect that it will consume the least total number of replicas and lowest multicast traffic. The second algorithm explores the best that we could expect to achieve gathering all topology information from the DOLR system.

4.2.3 Soft State Tree Management

Soft-state infrastructures have the potential to be extremely robust, precisely because they can be easily reconfigured to adapt to circumstances. For SCAN we target two types of adaptation: fault recovery and performance tuning.

To achieve fault resilience, the data source sends periodic *heartbeat* messages through the d-tree. Members know the frequency of these heartbeats and can react when they have not seen one for a sufficiently long time. In such a situation, the replica initiates a *rejoin* process – similar to the replica search phase above – to find a new parent. Further, each member periodically sends a *refresh* message to its parent. If the parent does not get the refresh message within a certain threshold, it invalidates the child's entry. With such soft-state group management, any SCAN server may crash without significantly affecting overall CDN performance.

Performance tuning consists of pruning and re-balancing the d-tree. Replicas at the leaves are pruned when they have seen insufficient client traffic. To balance the d-tree, each member periodically rejoins the tree to find a new parent.

4.3 Evaluation Methodology

We implement an event-driven simulator for SCAN because *ns*2 [16] can only scale up to one thousand nodes. This includes a packet-level network simulator (with a static version of the Tapestry DOLR) and a replica management framework. The soft-state replica layer is driven from simulated clients running workloads. Our methodology includes evaluation metrics, network setup and workloads.

4.3.1 Metrics

Our goal is to evaluate the replica schemes of Chapter 4.2.2. These strategies are dynamic naive placement (*od_naive*), dynamic smart placement (*od_smart*), overlay static placement (*overlay_s*), and static placement on IP network (*IP_s*). We compare the efficacy of these four schemes via three classes of metrics:

- *Quality of Replica Placement*: Includes number of deployed replicas and degree of load distribution, measured by the ratio of the standard deviation vs. the mean of the number of client children for each replica server.
- *Multicast Performance*: We measure the relative delay penalty (RDP) and the bandwidth consumption which is computed by summing the number of bytes multiplied by the transmission time over every link in the network. For example, the bandwidth consumption for 1K bytes transmitted in two links (one has 10 ms, the other 20 ms latency) is 1KB $\times$ (10+20)ms = 0.03(KB.sec).
- *Tree Construction Traffic:* We count both the number of application-level messages sent and the bandwidth consumption for deploying replicas and constructing d-tree.

In addition, we quantify the effectiveness of capacity constraints by computing the *maximal load* with or without constraints. The maximal load is defined as the maximal number of client cache children on any SCAN server. Sensitivity analysis are carried out for various client/server ratios and server densities.

4.3.2 Network Setup

We use the GT-ITM transit-stub model to generate five 5000-node topologies [134]. The results are averaged over the experiments on the five topologies. A packet-level, priority-queue based event manager is implemented to simulate the network latency. The simulator models the propagation delay of physical links, but does not model bandwidth limitations, queuing delays, or packet losses.

We utilize two strategies for placing SCAN servers. One selects all SCAN servers at random (labelled *random SCAN*). The other preferentially chooses transit and gateway

Table 4.1: Statistics of Web site access logs used for simulation

Web site	Period	# Requests total - simulated	# Clients	# Client groups total - simulated	# Objects simulated
MSNBC	10-11 am, 8/2/99	1604944 - 1377620	139890	16369 - 4000	4186
NASA	All day, 7/1/95	64398 - 64398	5177	1842 - 1842	3258

nodes (labelled *backbone SCAN*). This latter approach mimics the strategy of placing SCAN servers strategically in the network.

To compare with a DNS-redirection-based Web content distribution network (CDN), we simulate typical behavior of such a system. We assume that every client request is redirected to the closest CDN server, which will cache a copy of the requested information for the client. This means that popular objects may be cached in every CDN server. We assume that content servers are allowed to send updates to replicas via IP multicast.

4.3.3 Workloads

To evaluate the replication schemes, we use both a synthetic workload and access logs collected from real Web servers. These workloads are a first step toward exploring more general uses of SCAN.

Our synthetic workload is a simplified approximation of *flash crowds*. Flash crowds are unpredictable, event-driven traffic surges that swamp servers and disrupt site services. For our simulation, all the clients (not servers) make requests to a given hot object in random order.

Our trace-driven simulation includes a large and popular commercial news site, MSNBC [80], as well as traces from NASA Kennedy Space Center [81]. Table 4.1 shows the detailed trace information. We use the access logs in the following way. We group the Web clients based on BGP prefixes [67] using the BGP tables from a BBNPlanet (Genuity) router [12]. For the NASA traces, since most entries in the traces contain host names, we group the clients based on their domains, which we define as the last two parts of the host names (e.g., a1.b1.com and a2.b1.com belong to the same domain). Given the maximal topology we can simulate is 5000 (limited by machine memory), we simulate all the clients groups for NASA and 4000 top client groups (cover 86.1% of requests) for MSNBC. Since the clients are unlikely to be on transit nodes nor on server nodes, we map them randomly to the rest of nodes in the topology.

4.4 Evaluation Results

In this section, we evaluate the performance of the SCAN dynamic replica management algorithms. What we will show is that:

- For realistic workloads, SCAN places close to an optimal number of replicas, while providing good load balance, low delay, and reasonable update bandwidth consumption relative to static replica placement on IP multicast.

- SCAN outperforms the existing DNS-redirection based CDNs on both replication and update bandwidth consumption.

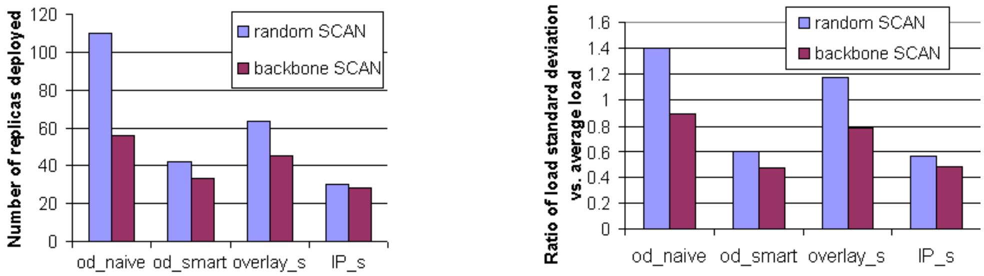


Figure 4.1: Number of replicas deployed (left) and load distribution on selected servers (right) (500 SCAN servers)

- The performance of SCAN is relatively insensitive to the SCAN server deployment, client/server ratio, and server density.
- The capacity constraint is quite effective at balancing load.

We will first present results on synthetic workload, and then the results of real Web traces.

4.4.1 Results for the Synthetic Workload

We start by examining the synthetic, flash crowd workload. 500 nodes are chosen to be SCAN servers with either "random" or "backbone" approach. Remaining nodes are clients and access some hot object in a random order. We randomly choose one non-transit SCAN server to be the data source and set as 50KB the size of the hot object. Further, we assume the latency constraint is 50ms and the load capacity is 200 clients/server.

Comparison Between Strategies

Figure 4.1 shows the number of replicas placed and the load distribution on these servers. *Od_smart* approach uses only about 30% to 60% of the servers used by *od_naive*, is even better than *overlay_s*, and is very close to the optimal case: *IP_s*. Also note that *od_smart* has better load distribution than *od_naive* and *overlay_s*, close to *IP_s* for both *random* and *backbone SCAN*.

Relative Delay Penalty (RDP) is the ratio of the overlay delay between the root and any member in d-tree vs. the unicast delay between them [32]. In Figure 4.2, *od_smart* has better RDP than *od_naive*, and 85% of *od_smart* RDPs between any member server and the root pairs are within 4. Figure 4.3 contrasts the bandwidth consumption of various replica placement techniques with the optimal IP static placement. The results are very encouraging: the bandwidth consumption of *od_smart* is quite close to *IP_s* and is much less than that of *od_naive*.

The performance above is achieved at the cost of d-tree construction (Figure 4.4). However, for both *random* and *backbone SCAN*, *od_smart* approach produces less than three times of the messages of *od_naive* and less than six times of that for optimal case: *IP_s*. Meanwhile, *od_naive* uses almost the same amount of bandwidth as *IP_s* while *od_smart* uses about three to five times that of *IP_s*.

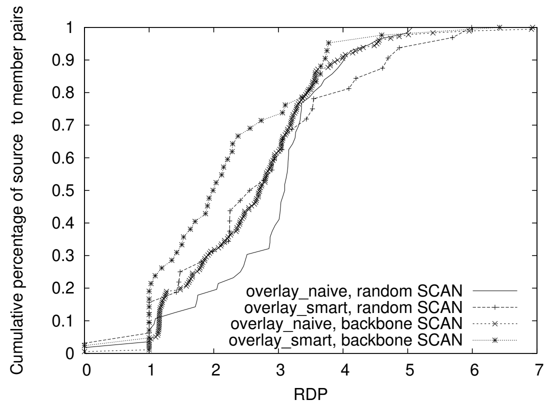


Figure 4.2: Cumulative distribution of RDP (500 SCAN servers)

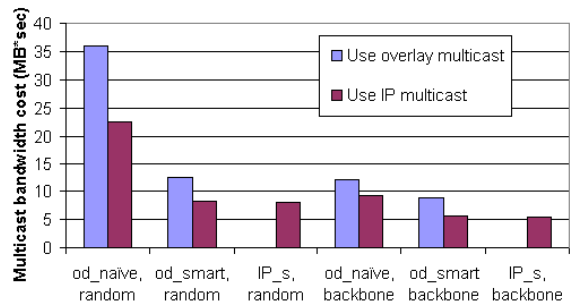
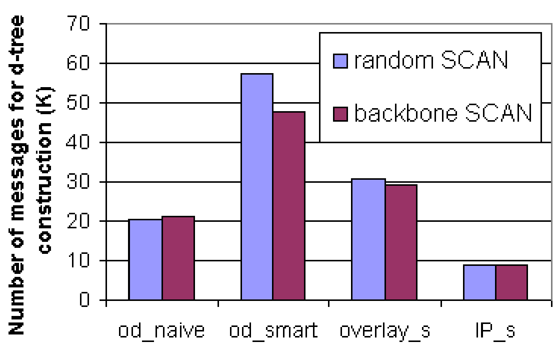


Figure 4.3: Bandwidth consumption of 1MB update multicast (500 SCAN servers)



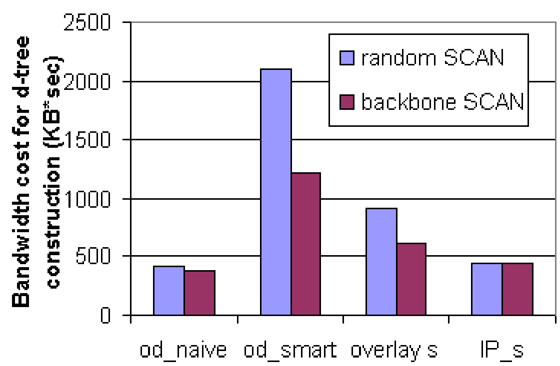


Figure 4.4: Number of application-level messages (left) and total bandwidth consumed (right) for d-tree construction (500 SCAN servers)

In short, the smart dynamic algorithm has performance that is close to the ideal case (static placement with IP multicast). It places close to an optimal number of replicas, provides better load distribution, and less delay and multicast bandwidth consumption than the naive approach – at the price of three to five times as much tree construction traffic. Since d-tree construction is a much less frequent than data access and update this is a good tradeoff.

Due to the limited number and/or distribution of servers, there may exist some clients who cannot be covered when facing the QoS and capacity requirements. In this case, our algorithm can provide hints as where to place more servers. Note that experiments show that the naive scheme has many more uncovered clients than the smart one, due to the nature of its unbalanced load. Thus, we remove it from consideration for the rest of synthetic workload study.

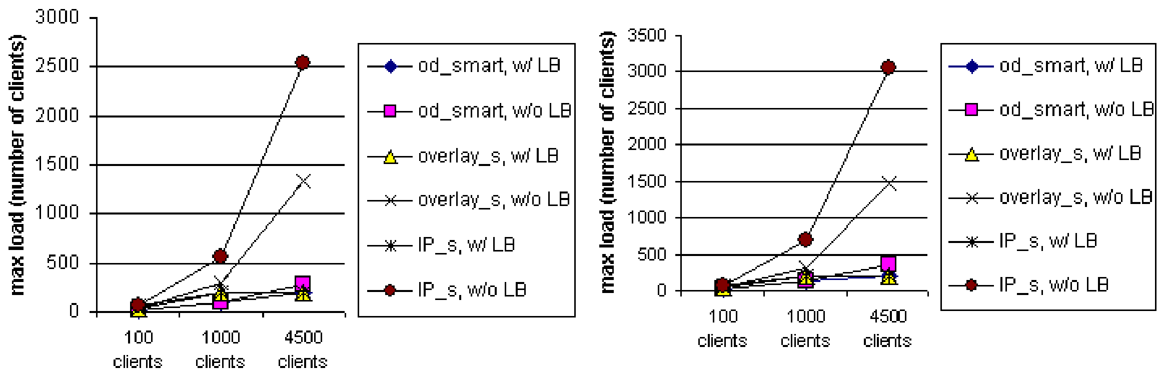


Figure 4.5: Maximal load measured with and without load balancing constraints (LB) for various numbers of clients (left: 500 random servers, right: 500 backbone servers)

Comparison with a CDN

As an additional comparison, we contrast the overlay smart approach with a DNS-redirection-based CDN. Compared with a traditional CDN, the overlay smart approach uses a fraction of the number of replicas (6-8%) and less than 10% of bandwidth for disseminating updates.

Effectiveness of Distributed Load Balancing

We study how the capacity constraint helps load balancing with three client populations: 100, 1000 and 4500. The former two are randomly selected from 4500 clients. Figure 4.5 shows that lack of capacity constraints (labelled *w/o LB*) leads to hot spot or congestion: some servers will take on about 2-13 times their maximum load. Performance with load balancing is labelled as *w/ LB* for contrast.

Performance Sensitivity to Client/Server Ratio

We further evaluate SCAN with the three client populations Figure 4.6 shows the number of replicas deployed. When the number of clients is small, *w/ LB* and *w/o LB* do not differ much because no server exceeds the constraint. The number of replicas required for *od_smart* is consistently less than that of *overlay_s* and within the bound of 1.5 for *IP_s*. As before, we also simulate other metrics, such as load distribution, delay and bandwidth penalty for update multicast under various client/server ratios. The trends are similar, that is, "od_smart" is always better than "overlay_s", and very close to "IP_s".

Performance Sensitivity to Server Density

Next, we increase the density of SCAN servers. We randomly choose 2500 out of the 5000 nodes to be SCAN servers and measure the resulting performance. Obviously, this configuration can support better QoS for clients and require less capacity for servers. Hence, we set the latency constraint to be 30 ms and capacity constraint 50 clients/server. The number of clients vary from 100 to 2500.

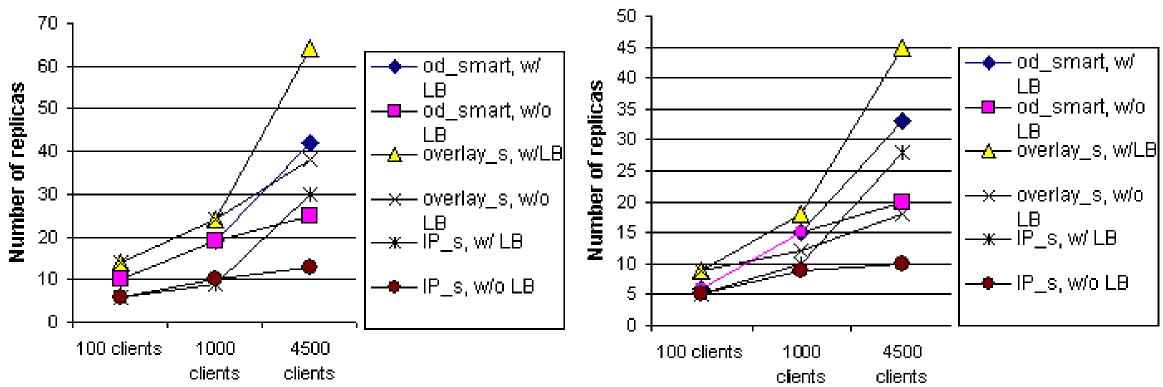


Figure 4.6: Number of replicas deployed with and without load balancing constraints (LB) for various numbers of clients (left: 500 random servers, right: 500 backbone servers)

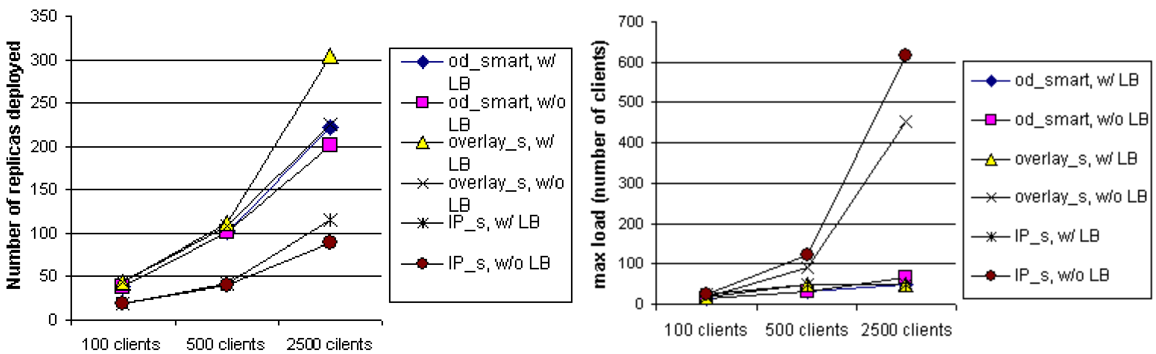


Figure 4.7: Number of replicas deployed (left) and maximal load (right) on 2500 random SCAN servers with and without the load balancing constraint (LB)

With very dense SCAN servers, our *od_smart* still uses less replicas than *overlay_s*, although they are quite close. *IP_s* only needs about half of the replicas, as in Figure 4.7. In addition, we notice that the load balancing is still effective. That is, overloaded machines or congestion cannot be avoided simply by adding more servers while neglecting careful design.

In summary, *od_smart* performs well with various SCAN server deployments, various client/server ratios, and various server densities. The capacity constraint based distributed load balancing is effective.

4.4.2 Results for Web Traces Workload

Next, we explore the behavior of SCAN for Web traces with documents of widely varying popularity. Figure 4.8.a characterizes the request distribution for the two traces used (note that the x-axis is logarithmic.). This figure reveals that the request number for different URLs is quite unevenly distributed for both traces.

For each URL in the traces, we compute the number of replicas generated with *od_naive*, *od_smart*, and *IP_s*. Then we normalize the replica numbers of *od_naive* and

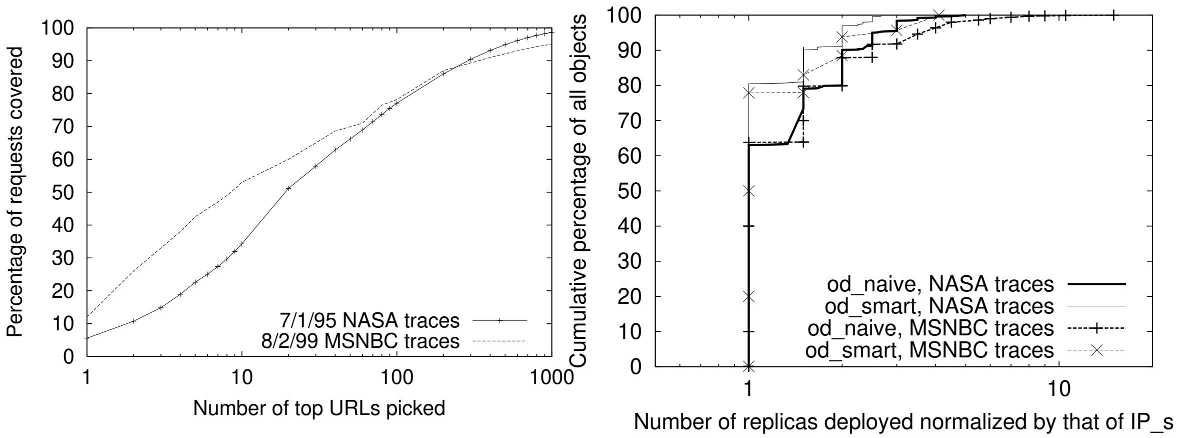


Figure 4.8: Simulation with NASA and MSNBC traces on 100 backbone SCAN servers. (a) Percentage of requests covered by different number of top URLs (left); (b) the CDF of replica number deployed with *od_naive* and *od_smart* normalized by the number of replicas using IP_s (right)

od_smart by dividing them with the replica number of IP_s. We plot the CDF of such ratios for both NASA and MSNBC in Figure 4.8.b. The lower percentage part of the CDF curves are overlapped and close to 1. The reasons are most of the URLs have very few requests, and we only simulate a limited period, thus the number of replicas deployed by the three methods are very small and similar. However, *od_smart* and *od_naive* differ significantly for popular objects, exhibited in the higher percentage part. *Od_smart* is very close to IP_s, for all objects, the ratio is less than 2.7 for NASA and 4.1 for MSNBC, while the ratio for *od_naive* can go as high as 5.0 and 15.0, respectively.

In addition, we contrast the bandwidth consumption for disseminating updates. Given an update of unit size, for each URL, we compute the bandwidth consumed by using (1) overlay multicast on an *od_naive* tree, (2) overlay multicast on an *od_smart* tree, and (3) IP multicast on an IP_s tree. Again, we have metric (1) and (2) normalized by (3), and plot the CDF of the ratios. The curves are quite similar to Figure 4.8.b.

In conclusion, although *od_smart* and *od_naive* perform similarly for infrequent or cold objects, *od_smart* outperforms dramatically over *od_naive* for hot objects which dominate overall requests.

4.4.3 Discussion

How does the distortion of topology through Tapestry affect replica placement? Notice that the overlay distance through Tapestry, on average, is about 2-3 times more than the IP distance. Our simulations in Chapter 4.4, shed some light on the resulting penalty: *Overlay_s* applies exactly the same algorithm as IP_s for replica placement, but uses the static Tapestry-level topology instead of IP-level topology. Simulation results show that *overlay_s* places 1.5 - 2 times more replicas than IP_s. For similar reasons, *od_smart* outperforms *overlay_s*. The reason is that *od_smart* uses "ping" messages to get the real IP distance between clients and servers. This observation also explains why *od_smart* gets similar performance to IP_s. One could imagine scaling overlay latency by an expected "stretch" factor to estimate real IP distance – thereby reducing ping probe traffic.

So far, we have discussed how to deploy, locate and maintain the coherence of the replicas. In the next chapter, we will cluster the contents to reduce the replica index management overhead without sacrificing the performance.

Chapter 5

Scalable Replica Management using Content Clustering

5.1 Introduction

In the previous chapters, we examined scalable replica locations and dynamic replica placement. As mentioned in Chapter 1, there are more than 3.3 billion Web pages [55] and a daily growth of 7 million pages [95]. Thus to manage the content replicas for such a gigantic scale remains a big challenge. In this chapter, we address the management overhead issues via content clustering. In other words, we focus on an orthogonal issue in Web replication: what content is to be replicated, and in what granularity.

We start by analyzing several access traces from large commercial and government Web servers. Our analysis shows that 10% of hot data can cover over 80% of requests, and this coverage can last for at least a week in all the traces under study. This suggests that it is cost effective to replicate only the hot data and leave the cold data at the origin Web server.

Then we compare the traditional un-cooperative pulling (Chapter 2.2) vs. cooperative pushing (Chapter 2.3). Simulations on a variety of network topologies using real Web traces show that the latter scheme can yield similar clients' latency with only about 4-5% of the replication and update cost compared to the former scheme.

Motivated by this observation, we explore how to efficiently push content to CDN nodes. Our study is based on cooperative-push based CDN, but the technique is also applicable to SCAN.

We compare the performance between per Web site-based replication (hot data only) versus per hot URL-based replication. We find per URL-based scheme yields a 60-70% reduction in clients' latency. However, it is very expensive to perform such a fine-grained replication: it takes 102 hours to come up with the replication strategy for 10 replicas per URL on a PII-400 server. This is clearly not acceptable in practice.

To address the issue, we propose several clustering algorithms that group Web content based on their correlation, and replicate objects in units of content clusters (i.e., all the objects in the same cluster are replicated together). We evaluate the performance of cluster-based replication by simulating their behavior on a variety of network topologies

Web Site	Period	Duration	# Requests avg - min - max	# Clients avg - min - max	# Client Groups avg - min - max
MSNBC	8/99 - 10/99	10 am-11 am	1.5M - 642K - 1.7M	129K - 69K - 150K	15.6K - 10K - 17K
NASA	7/95 - 8/95	All day	79K - 61K - 101K	5940 - 4781 - 7671	2378 - 1784 - 3011
WorldCup	5/98 - 7/98	All day	29M - 1M - 73M	103K - 13K - 218K	N/A

Table 5.1: Access logs used.

using the real traces. Our results show that the cluster-based replication schemes yield performance close to that of the URL-based scheme, but only at 1% - 2% of computation and management cost (The management cost includes communication overhead and the cost of keeping track of where content has been replicated). For instance, the computation time for 10 replicas per URL with 20 clusters is only 2.5 hours under the same platform.

Finally, as the users' access pattern changes over time, it is important to adapt content clusters to such changes. Simulations show that clustering and replicating content based on old access pattern does not work well beyond one week; on the other hand, complete re-clustering and re-distribution, though achieves good performance, has large overhead. To address the issue, we explore incremental clustering that adaptively add new documents to the existing content clusters. We examine both offline and online incremental clustering, where the former assumes access history is available while the latter predicts access pattern based on hyperlink structure. Our results show that the offline clustering yields close to the performance of the complete re-clustering with much lower overhead. The online incremental clustering and replication reduce the retrieval cost by 4.6 times compared with random replication, and by 8 times compared with no replication. So it is very useful to improve document availability during flash crowds.

The rest of the chapter is organized as follows. We describe our simulation methodology in Chapter 5.2, and study the temporal stability of popular documents in Chapter 5.3. In Chapter 5.4, we compare the performance of the pull-based vs. push-based replication. We formulate the push-based content placement problem in Chapter 5.5, and compare the Web site-based replication and the URL-based replication using trace-driven simulation in Chapter 5.6. We describe content clustering techniques for efficient replication in Chapter 5.7, and evaluate their performance in Chapter 5.8. In Chapter 5.9, we examine offline and online incremental clustering that take into account of changes in users' access pattern. Finally we conclude in Chapter 5.10.

5.2 Simulation Methodology

Throughout the chapter, we use trace-driven simulations to evaluate the performance of various schemes. In this section, we describe the network topologies and Web traces we use for evaluation.

5.2.1 Network Topology

In our simulations, we use three random network topologies generated from the GT-ITM internetwork topology generator [134]: pure random, Waxman, and Transit-Stub.

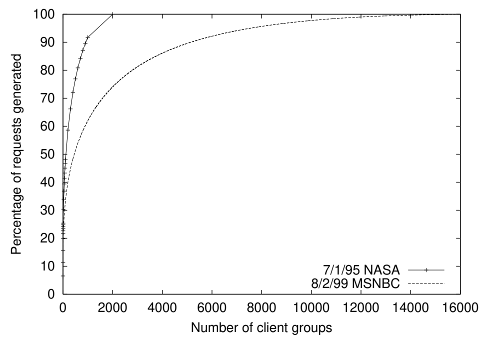


Figure 5.1: The CDF of the number of requests generated by the Web client groups defined by BGP prefixes for the MSNBC traces, and by domains for the NASA traces.

In the pure random model, nodes are randomly assigned to locations on a plane, with a uniform probability p of an edge added between a pair of nodes. The Waxman model also places nodes randomly, but creates an edge between a pair of node u and v with probability $P(u, v) = \alpha e^{(-d/\beta L)}$, where $d = |\overrightarrow{u} - \overrightarrow{v}|$, L is the maximum Euclidean distance between any two vertices, and $\alpha > 0$ and $\beta \leq 1$. The Transit-Stub model generates network topologies composed of interconnected transit and stub domains, and better reflects the hierarchical structure of real networks. We further experiment with various parameters in every topology model.

In addition to using synthetic topologies, we also construct an AS-level Internet topology using BGP routing data collected from a set of seven geographically-dispersed BGP peers in April 2000 [63]. Each BGP routing table entry specifies an AS path, AS_1, AS_2, ..., AS_n, *etc.*, to a destination address prefix block. We construct a graph using the AS paths, where individual clients and address prefix blocks are mapped to their corresponding AS nodes in the graph, and every AS pair has an edge with the weight being the shortest AS hop count between them. While not very detailed, an AS-level topology at least partially reflects the true topology of the Internet.

5.2.2 Web Workload

In our evaluation, we use the access logs collected at the MSNBC server site [80], as shown in Table 5.1. MSNBC is a popular news site that is consistently ranked among the busiest sites in the Web [75]. For diversity, we also use the traces collected at NASA Kennedy Space Center in Florida [81] during 1995 and the World Cup Web site in 1998 [8]. Table 5.1 shows the detailed trace information. The number of client groups is unavailable in the WorldCup traces because it anonymizes the clients' Internet Protocol (IP) addresses. As a result, we are unable to group clients to study their aggregated behavior for clustering. So we only use it to analyze stability of document popularity.

Both the NASA and WorldCup traces contain accesses to images and HTML content. For administrative reasons, MSNBC traces only record accesses to HTML content.

We use the access logs in the following way. When using the AS-level topology, we group clients in the traces based on their AS numbers. When using random topologies, we group the Web clients based on BGP prefixes [67] using the BGP tables from a BBNPlanet (Genuity) router [12]. For the NASA traces, since most entries in the traces contain host names, we group the clients based on their domains, which we define as the last two parts of the host names (e.g., a1.b1.com and a2.b1.com belong to the same domain). Figure 5.1 plots the CDF of the number of requests generated by Web client groups. As we can see, in August 2, 1999 MSNBC traces, the top 10, 100, 1000, 3000 groups account for 18.58%, 33.40%, 62.01%, and 81.26% of requests, respectively; in July 1, 1995 NASA traces, the top 10, 100, 1000 groups account for 25.41%, 48.02%, and 91.73% of requests, respectively.

We choose top 1000 client groups in the traces since they cover most of the requests (62-92%) and map them to 1000 nodes in the random topologies. Assigning a group C_i to a node P_i in the graph means that the weight of the node P_i is equal to the number of requests generated by the group C_i.

In our simulations, we assume that replicas can be placed on any node, where a node represents a popular IP cluster in the MSNBC traces, or a popular domain in the NASA traces. Given the rapid growth of CDN service providers, such as Akamai (which already has more than 15,000 servers in about 1,100 networks around the world [4]), we believe this is a realistic assumption. Moreover, for any URL, the first replica is always at the origin Web server (a randomly selected node), as in [71, 104]. However, including or excluding the original server as a replica is not a fundamental choice and has little impact on our results.

5.2.3 Performance Metric

We use the average retrieval cost as our performance metric, where the retrieval cost of a Web request is the sum of the costs of all edges along the path from the source to the replica from which the content is downloaded. In the synthetic topologies, the edge costs are generated by the Georgia Tech Internetwork Topology Models (GT-ITM) [134]. In the AS-level Internet topology, the edge costs are all 1, so the average retrieval cost represents the average number of AS hops that a request traverses.

5.3 Stability of Hot Data

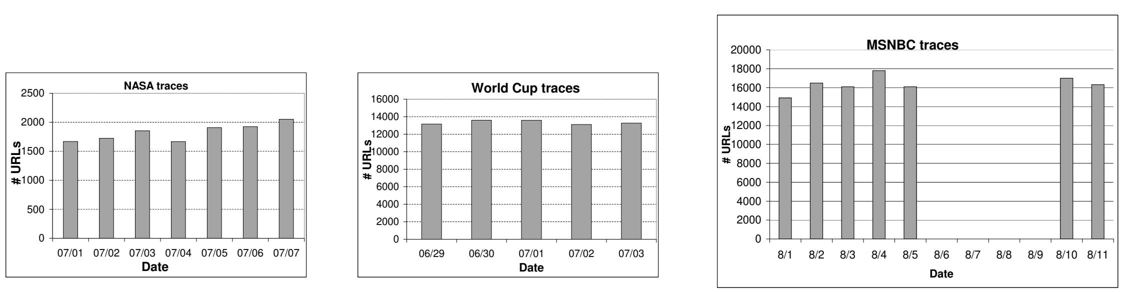
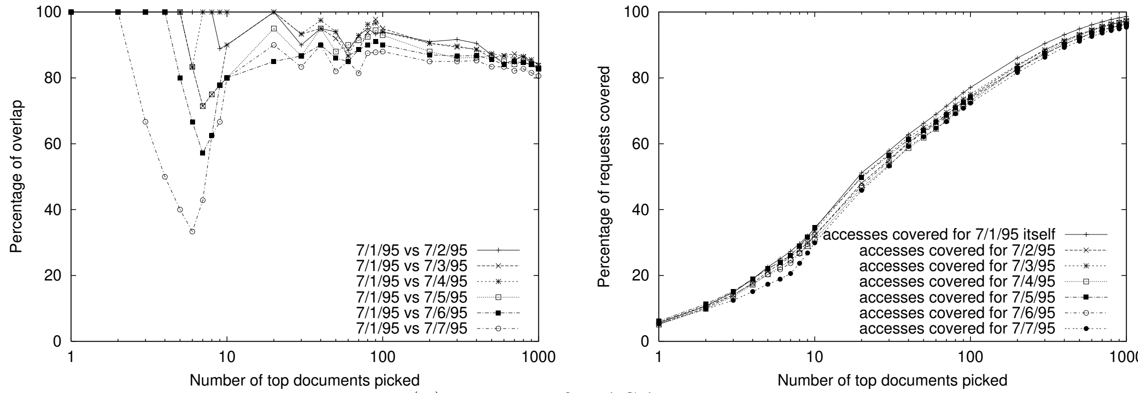
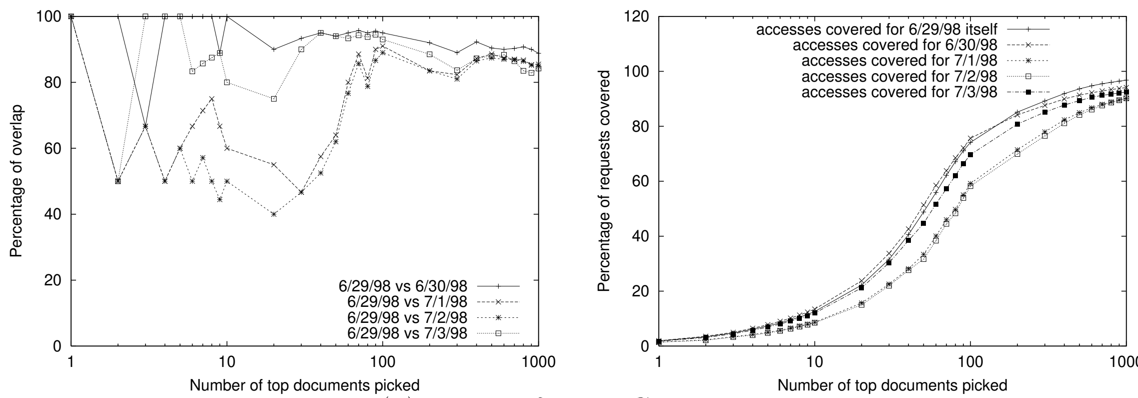


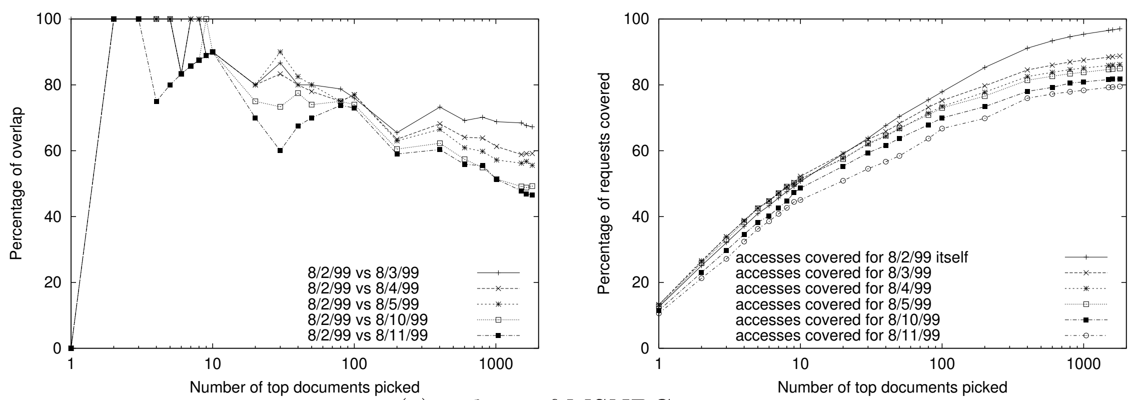
Figure 5.2: The number of URLs accessed in NASA, WorldCup, and MSNBC daily traces.



(a) 7 days of NASA traces

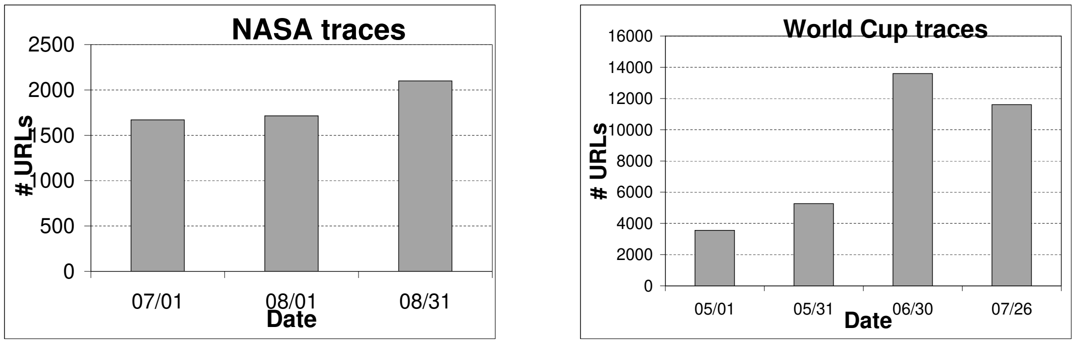


(b) 5 days of WorldCup Traces

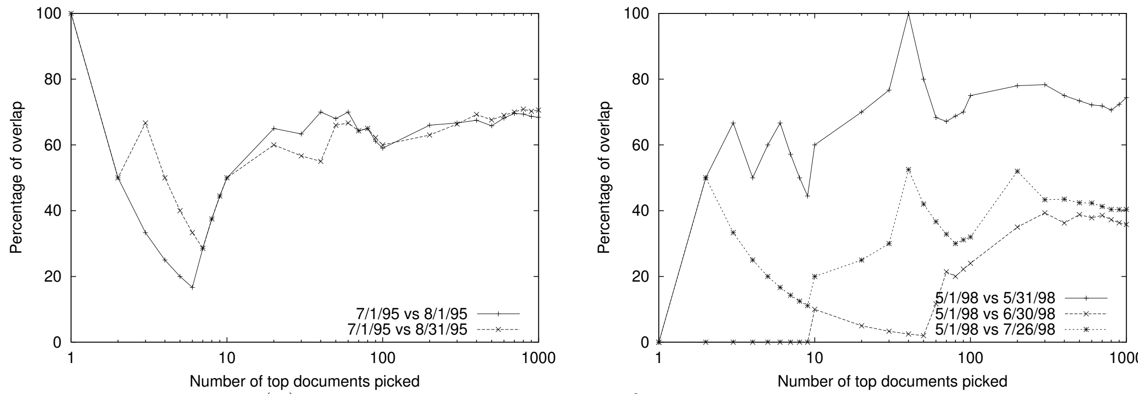


(c) 6 days of MSNBC traces

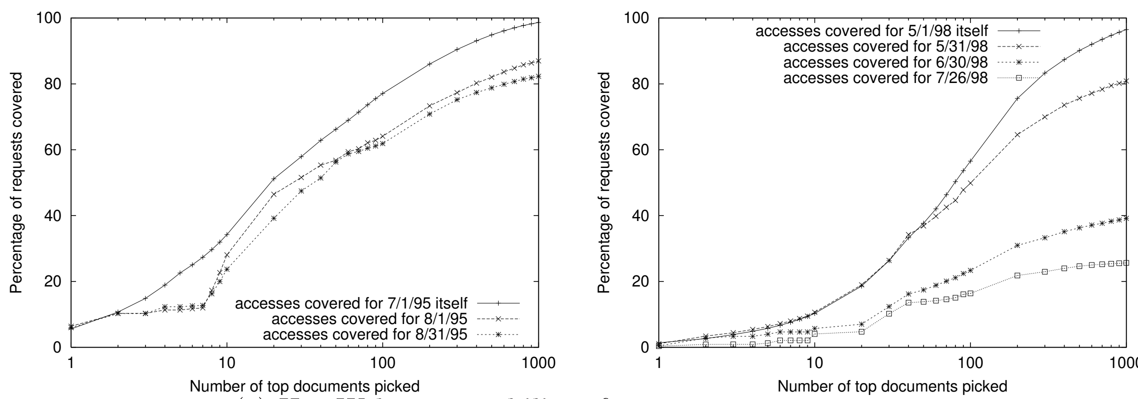
Figure 5.3: Hot Web page stability of popularity ranking (left column), and stability of access request coverage (right column) with daily intervals.



(a) The number of URLs accessed.



(b) Hot Web page stability of popularity ranking.



(c) Hot Web page stability of access request coverage.

Figure 5.4: Number of URLs and hot Web page statistics for NASA (left column) and WorldCup (right column) with monthly intervals.

Many studies report that Web accesses follow the Zipf-like distribution [15], which are also exhibited by our traces. This indicates that there is large variation in the number of requests received by different Web pages, and it is important to focus on popular pages when doing replication. For replicating popular pages to be an effective approach, the popularity of the pages needs to be stable. In this section, we investigate this issue.

We analyze the stability of Web pages using the following two metrics: (i) the stability of Web page popularity rankings, as used in [94], and (ii) the stability of request coverage from (previous) popular Web pages. The latter is an important metric to quantify the efficiency of pre-fetching/pushing of hot Web data. One of our interesting findings is that while the popularity ranking may fluctuate, the request coverage still remains stable.

We study the stability of both metrics in various time scales: within a month, a day, an hour, a minute, and a second. They show similar patterns, so we only present the results for the daily and monthly scale. Figures 5.2 and 5.4(a) show the number of unique URLs in the traces. Figures 5.3 and 5.4 show the stability results for the time gap of a few days and one month, respectively. In all the graphs on the top of Figures 5.3 and 5.4, the x axis is the number of most popular documents picked from each day, and the y axis is the percentage of overlap between the documents picked from the two days. As we can see, the overlap is mostly over 60%, which indicates many documents are popular on both days. On the other hand, for the WorldCup site, which is event-driven and frequently has new content added, the overlap sometimes drops to 40%.

A natural question arises: whether the old hot documents can continue to cover a majority of requests as time evolves. The graphs of Figures 5.3 and 5.4 shed light on this. Here we pick the hot documents from the first day, and plot the percentage of requests covered by these documents for the first day itself and for the following days. As we can see, the request coverage remains quite stable. The top 10% of objects on one day can continue to cover over 80% requests for at least the subsequent week. We also find that the stability of content varies across different Web sites. For example, the stability period of WorldCup site is around a week, while the top 10% objects at the NASA site can continue to cover the majority of requests for two months.

Based on the above observations, we conclude that when performing replica placement, we only need to consider the top few URLs (e.g., 10%), as they account for most of the requests. Furthermore, since the request coverage of these top URLs remains stable for a long period (at least a week), it is reasonable to replicate based on previous access patterns, and change the provision infrequently. This helps to reduce the cost of replication, computation, and management. We will examine this issue further in Chapter 5.9.

5.4 Replication Methods: Un-cooperative Pulling vs. Cooperative Pushing

As mentioned before, for simplicity, our scalable replica management study is based on cooperative push-based CDNs, another emerging paradigm shift from the conventional CDNs. In this section, we compare the performance between the un-cooperative pulling versus cooperative pushing (see Chapters 2.2 and 2.3) using trace-driven simulation as follows.

We apply the MSNBC trace during 10am - 11am of 8/2/1999 to a transit-stub topology. We choose the top 1000 URLs and top 1000 client groups with 964466 requests in total. In our evaluation, we assume that there is a CDN node located in every client group. In the un-cooperative pulling, we assume that a request is always redirected to the client's local CDN node and the latency between the client and its local CDN node is negligible (i.e., latencies incurred at step 5 and 8 shown in Figures 2.1 and 2.2 are zero.), since they both belong to the same client group. In the cooperative push-based scheme, we replicate content in units of URLs to achieve similar clients' latency. Requests are directed to the closest replicas. (If the content is not replicated, the request goes to the origin server.) The details of replication algorithm will be explained in Chapter 5.6. As shown in Figures 2.1 and 2.2, the resolution steps (1-4) to locate a CDN node are the same for both schemes. Therefore we only need to compare the time for the "GET" request (steps 5-8 in Figures 2.1 and 2.2).

Our results show that the un-cooperative pulling out-sources 121016 *URL replicas* (with a total size of 470.3MB) to achieve an average round trip time (RTT) of 79.2ms, where the URL replica is the total number of times URLs being replicated (e.g., URL_1 replicated 3 times, and URL_2 replicated 5 times, then the replication cost is 8 URL replicas). In comparison, the cooperative push-based scheme (per URL) distributes only 5000 URL replicas (with a total size of 18.5MB) to achieve a comparable average latency (i.e., 77.9ms) [1].

We also use the same access logs along with the corresponding modification logs to compare the cost of pushing updates to the replicas to maintain consistency. In our experiment, whenever a URL is modified, the Web server must notify all the nodes that contain the URL. Because the update size is unavailable in the trace, we use the total number of messages sent as our performance metric. With 11,509 update events in 8/2/1999, the un-cooperative pulling uses 1349655 messages (about 1.3GB if we assume average update size is 1KB), while the cooperative per-URL based pushing only uses 54564 messages (53.3MB), about 4% update traffic, to achieve comparable user latency.

The results above show that cooperative pushing yields much lower traffic, compared to the traditional un-cooperative pulling, which is currently in commercial use. The main reason for the traffic savings is that in the cooperative scheme, Web pages are strategically placed at selected locations, and a client's request is directed to the closest CDN node that contains the requested objects, while in the un-cooperative scheme, requests are directed to the closest CDN node that may or may not contain the requested objects. Therefore, while the analysis is based on one-hour trace, the performance benefit of cooperative pushing does not reduce over a longer period of time due to content modification and creation. Of course, the performance benefit of the cooperative push-based scheme comes at the cost of maintaining content directory information. However this cost is adjustable by changing the granularity of content clusters as shown in Chapter 5.7.

Another advantage of the cooperative scheme is the ability to smoothly trade off management and replication cost for better client performance due to the combination of informed request redirection and content clustering (Chapter 5.7). In comparison, in the

[1] Here we compare the number of URL replicas under two schemes. But these replicas are not generated within an hour (10-11am) because many of these URLs are old and have been requested earlier. See the stability of hot content in Chapter 5.3.

uncooperative scheme, requests can be satisfied either at a local replica or at the origin Web server, but not at a nearby replica due to lack of informed request redirection. As a result, the uncooperative scheme does not have much flexibility in adjusting replication and management cost.

Furthermore, for newly created content that has not been accessed, cooperative pushing is the only way to improve its availability and performance. We will study such performance benefits in Chapter 5.9.2.

Motivated by the observations, in the remainder of the chapter we explore how to effectively push content to CDN nodes.

5.5 Problem Formulation

We describe the Web content placement problem as follows. Consider a popular Web site or a CDN hosting server, which aims to improve its performance by pushing its content to some hosting server nodes. The problem is to decide what content is to be replicated and where to place the replicas so that some objective function is optimized under a given traffic pattern and a set of resource constraints. The objective function can minimize either clients' latency, or loss rate, or total bandwidth consumption, or an overall cost function if each link is associated with a cost.

For Web content delivery, the major constraint in replication is the network access bandwidth at each Internet Data Center (IDC) to the backbone network. Moreover, replication is not a one-time cost. Once a page is replicated, we need to spend additional resources to keep it consistent. Depending on the update and access rate, the cost of keeping replicas consistent can be high.

Based on the above observations, we formulate the Web content placement problem as follows. Given a set of URLs U and a set of locations L to which the URLs can be replicated, replicating a URL incurs a replication cost. A client j fetching a URL u from the ith replica of u located at $l_{u(i)}$ incurs a cost of $C_{j,l_{u(i)}}$, where $C_{j,l_{u(i)}}$ denotes the distance between j and $l_{u(i)}$. Depending on the metric we want to optimize, the distance between two nodes can reflect either the latency, or loss rate, or total bandwidth consumption or link cost. The problem is to find a replication strategy (i.e., for each URL u, we decide the set of locations $l_{u(i)}$ to which u is replicated) such that it minimizes

$$\sum_{j \in CL} \left(\sum_{u \in U_j} (min_{i \in \ R_u} C_{j,l_{u(i)}}) \right)$$

subject to the constraint that the total replication cost is bounded by R, where CL is the set of clients, U_j is the set of URLs requested by the j-th client, R_u is the set of locations to which URL u has been replicated. (The total replication cost is either $\sum_{u \in U} |u|$ assuming the replication cost of all URLs is the same, or $\sum_{u \in U} |u| * f(u)$ to take into account of different URL sizes, where $|u|$ is the number of different locations to which u is replicated, $f(u)$ is the size of URL u.)

5.6 Replica Placement: Per Web Site vs. Per URL

In this section, we examine if replication at a fine granularity can help to improve the performance for push-based scheme. One natural question arises: is it possible to replicate a modern Web site only partially? There are about 40% of HTTP requests to which the reply are generated dynamically and thus uncacheable [49]. However, the rest of requests can still be satisfied with partial cache. Many current CDN operators, such as Akamai [4], only cache individual URL based on clients' demand [41].

Our performance metric is the total latency incurred for all the clients to fetch their requested documents, as recorded in the traces. We compare the performance of replicating all the hot data at a Web site as one unit (*i.e.*, per Web site-based replication, see Algorithm 4) versus replicating content in units of individual URLs (*i.e.*, per URL-based replication, see Algorithm 5). For simplicity, we assume that all URLs are of the same size. The non-uniform nature of size distribution actually has little effect on the results as shown in Chapter 5.8.2.

For both algorithms below, $totalURL$ is the number of distinct URLs at the Web site to be replicated, $currRepCost$ is the current number of URL replicas deployed, and $maxRepCost$ is the total number of URL replicas that can be deployed.

procedure GreedyPlacement_Site($maxRepCost$, $totalURL$)
Initially, all the URLs reside at the origin Web server, $currRepCost = totalURL$
while $currRepCost < maxRepCost$ **do**
    foreach *node i without the Web site replica* **do**
        Compute the clients' total latency reduction if the Web site is replicated to i (denoted as $gain_i$)
    end
    Choose node j with maximal $gain_j$ and replicate the Web site to j
    $currRepCost += totalURL$
end

Algorithm 4: Greedy Replica Placement (Per Web site)

When replicating content in units of URLs, not all URLs have the same number of replicas. Given a fixed replication cost, we give a higher priority to URLs that yield more improvement in performance. Algorithm 5 uses a greedy approach to achieve it: at each step, we choose the $< object, location >$ pair that gives the largest performance gain.

We will compare the computational cost of the two algorithms with clustering-based approach in Chapter 5.7. Figure 5.5 shows the performance gap between per Web site-based replication and per URL-based replication. The first replica is always at the origin Web server for both schemes, as described in Chapter 5.2. In our simulation, we choose top 1000 URLs from August 2, 1999 MSNBC traces, covering 95% of requests, or top 300 URLs from July 1, 1995 NASA traces, covering 91% of requests. For the MSNBC traces, per URL-based replication can consistently yield a 60-70% reduction in clients' latency; for the NASA traces, the improvement is 30-40%. The larger improvement in the MSNBC traces is likely due to the fact that requests in the MSNBC traces are more concentrated

procedure GreedyPlacement_URL($maxRepCost$, $totalURL$)

1 Initially, all the URLs reside at the origin Web server, $currRepCost = totalURL$

2 **foreach** *URL* u **do**

3 **foreach** *node* i **do**

4 Compute the clients' total latency reduction for accessing u if u is replicated to i (denoted as $gain_{u_i}$)

end

5 Choose node j with maximal $gain_{u_j}$, $best\_site_u = j$ and $max\_gain_u = gain_{u_j}$

end

6 **while** $currRepCost < maxRepCost$ **do**

7 Choose URL v with maximal max_gain_v, replicate v to $best\_site_v$

8 Repeat steps 3, 4 and 5 for v

9 $currRepCost{+}{+}$

end

Algorithm 5: Greedy Replica Placement (Per URL)

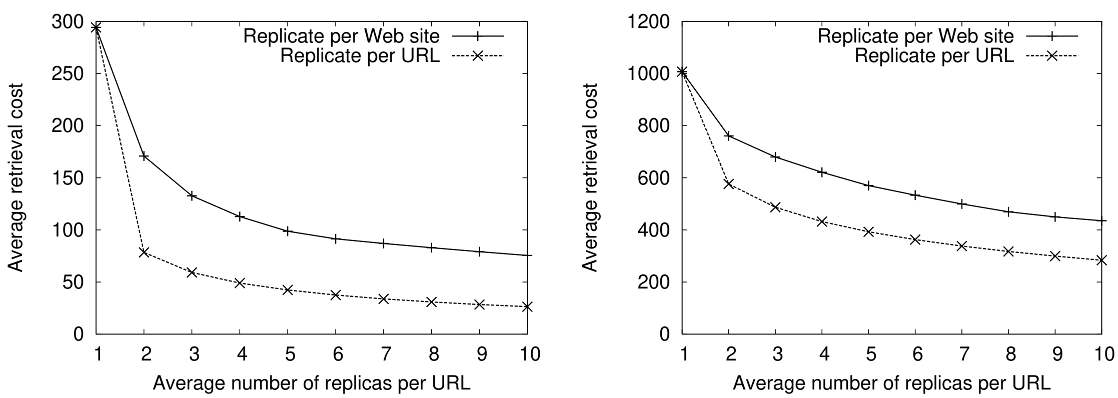


Figure 5.5: Performance of the per Web site-based replication vs. the per URL-based replication for August 2, 1999 MSNBC traces on a transit-stub topology (left) and July 1, 1995 NASA traces on a pure random topology (right).

on a small number of pages, as reported in [94]. As a result, replicating the very hot data to more locations, which is allowed in per URL-based scheme, is more beneficial.

One simple improvement to per Web-site based replication is to limit the set of URLs to be replicated to only the most popular ones. However it is crucial to determine the number of hot URLs to be replicated based on their popularity. This is essentially a simple variant of the popularity-based clustering discussed in Chapter 5.7.2, except that there are two clusters, and only the hot one is replicated. We found that once the optimum size of hot URL set is available, it can achieve the performance close to that of the popularity-based clustering. However, the greedy algorithm for choosing replica locations of the hot URL set (Algorithm 4) is still important, and much better than simply distributing the small set of hot URLs across all client groups. Simulations show that under the same replication cost, its average retrieval cost can be 2 - 5 times that of per URL based replication (Algorithm 5).

5.7 Clustering Web Content

In the previous section, we have shown that a fine-grained replication scheme can reduce clients' latency by up to 60-70%. However since there are thousands of hot objects that need to be replicated, searching over all the possible $< object, location >$ combinations is prohibitive. In our simulations, it takes 102 hours to come up with a replication strategy for 10 replicas per URL on a PII-400 low end server. This approach is too expensive for practical use even using high end servers. To achieve the benefit of the fine-grained replication at reasonable computation and management cost, in this section, we investigate clustering Web content based on their access pattern, and replicate content in units of clusters.

At a high level, clustering enables us to smoothly tradeoff the computation and management cost for better clients' performance. Per URL-based replication is one extreme of clustering: create a cluster for each URL. It can achieve good performance at the cost of high management overhead. In comparison, per Web site-based replication is another extreme: one cluster for each Web site. While it is easy to manage, its performance is much worse than the former approach, as shown in Chapter 5.6. We can smoothly tradeoff between the two by adjusting the number of clusters. This provides more flexibility and choices in CDN replication. Depending on the CDN provider's need, it can choose whichever operating point it finds appropriate.

Below we quantify how clustering helps to reduce computation and management cost. The notations are in Table 5.2. Suppose there are N objects, and M (roughly $N/10$) hot objects to be put into K clusters ($K < M$). Assume on average there are R replicas/URL that can be distributed to S CDN servers to serve C clients. In the per cluster-based replication, we not only record where each cluster is stored, but also keep track of the cluster to which each URL belongs. Note that even when R approaches hundreds and M approaches tens of thousands, it is quite trivial to store all the information. The storage cost of per URL based replication is also manageable.

On the other hand, the computation cost of the replication schemes is much higher, and becomes an important factor that determines the feasibility of the schemes in practice. The computational complexities of Algorithm 4 and Algorithm 5 are $O(RSC)$ [104] and

Symbols	Meanings
N	number of objects
M	number of hot objects
K	number of clusters
R	number of replicas per URL
S	number of CDN servers
C	number of clients
f_p	placement adaptation frequency
f_c	clustering frequency

Table 5.2: Table of notations

Replication Scheme	States to Maintain	Computation Cost
Per Web Site	$O(R)$	$f_p \times O(RSC)$
Per Cluster	$O(RK + M)$	$f_p \times O(KR \times (K + SC)) + f_c \times O(MK)$
Per URL	$O(RM)$	$f_p \times O(MR \times (M + SC))$

Table 5.3: Management overhead comparison for replication at different granularities, where $K < M$.

$O(MR \times (M + SC))$, respectively. In Algorithm 5, there are MR iterations and for each we have to choose the $< object, location >$ pair which will give the most performance gain from M candidates. After that, the next best location for that object and its cost gain need to be updated with $O(SC)$ computation cost. Similarly, the complexity of the cluster-based replication algorithm is $O(KR \times (K + SC))$. There is an additional clustering cost, which varies with the clustering algorithm that is used. Assuming the placement adaptation frequency is f_p and the clustering frequency is f_c, Table 5.3 summarizes the management cost for the various replication schemes. As we will show in Chapter 5.9, the content clusters remain stable for at least one week. Therefore f_c is small, and the computational cost of clustering is negligible compared to the cost of the replication.

One possible clustering scheme is to group URLs based on their directories. While simple, this approach may not capture correlations between URLs' access patterns for the following reasons. First, deciding where to place a URL is a complicated process, since it is difficult to predict without consulting the actual traces whether two URLs are likely to be accessed together. Even with full knowledge about the contents of URLs, the decision is still ad hoc since people have different interpretations of the same data. Also, most Web sites are organized with convenience of maintenance in mind, and such organization does not correspond well to the actual correlation of URLs. For example, a Web site may place its HTML files in one directory and images in another, even though a HTML file is always accessed together with its linked images. Finally, it can be difficult to determine the appropriate directory level to separate the URLs.

We test our hypothesis with the MSNBC traces of August 1, 1999: we cluster the top 1000 URLs using the 21 top level directories, and then use the greedy algorithms to place on average 3 replicas/URL (i.e., 3000 replicas in total). Compared with per Web site-based replication, it reduces latency only by 12% for pure random graph topology, and by 3.5% for transit-stub topology. Therefore the directory-based clustering only yields a marginal benefit over the Web site based replication for the MSNBC site; in comparison, as we will show later, clustering content based on access pattern can yield more significant performance improvement: 40% - 50% for the above configuration.

In the remaining of this section, we examine content clustering based on access patterns. We start by introducing our general clustering framework, and then describe the correlation distance metrics we use for clustering.

5.7.1 General Clustering Framework

Clustering data involves two steps. First, we define the correlation distance between every pair of URLs based on a certain correlation metric. Then given n URLs and their correlation distance, we apply standard clustering schemes to group them. We will describe our distance metrics in Chapter 5.7.2. Regardless of how the distance is defined, we can use the following clustering algorithms to group the data.

We explore two generic clustering methods. The first one aims to minimize the maximum diameter of all clusters while limiting the number of clusters. The diameter of cluster i is defined as the maximum distance between any pair of URLs in cluster i. It represents the worst-case correlation within that cluster. The second one aims to minimize the number of clusters while limiting the maximum diameter of all clusters.

1. Minimize the maximum diameter of all clusters while limiting the number of clusters. We use the classical K-split algorithm by T. F. Gonzalez [54]. It is a $O(NK)$ approximation algorithm, where N is the number of points and K is the number of clusters. It guarantees solution within twice the optimal.

2. Minimize the number of clusters while limiting the diameter of each cluster. This can be reduced to the problem of finding cliques in a graph using the following algorithm: Let N denote the set of URLs to be clustered, and d denote the maximum diameter of a cluster. Build graph $G(V, E)$ such that $V = N$ and edge $(u, v) \in E \Leftrightarrow dist(u, v) < d$, $\forall u, v \in V$. We can choose d using some heuristics, e.g., a function of average distance over all URLs. Under this transformation, every cluster corresponds to exactly one clique present in the generated graph. Although the problem of partitioning graphs into cliques is NP-complete, we adopt the best known approximation algorithm with time complexity $O(N^3)$ [45].

We have applied both clustering algorithms, and obtained similar results. To avoid redundancy, for the rest of this chapter, we only use the first clustering algorithm.

5.7.2 Correlation Distance

In this section, we describe the correlation distance we use. We explore three orthogonal distance metrics: one based on spatial locality, one based on temporal locality, and another based on popularity locality. The metrics can also be based on semantics, such as the hyperlink structures or XML tags in Web pages. We will examine the hyperlink structure for online incremental clustering in Chapter 5.9.2, and leave the clustering based on other metadata, such as XML tags, for future work.

Spatial Clustering

First, we look at clustering content based on their spatial locality in the access patterns. At a high level, we would like to cluster URLs that share similar access distribution across different regions. For example, two URLs that both receive the largest number of accesses from New York and Texas and both receive the least number of accesses from California may be clustered together.

We use BGP prefixes or domain names to partition the Internet into different regions, as described in Chapter 5.2. We represent the access distribution of a URL using a spatial access vector, where the ith field denotes the number of accesses to the URL from the i-th client group. Given L client groups, each URL is uniquely identified as a point in L-dimensional space. In our simulation, we use the top 1000 clusters (i.e., $L = 1000$), covering 70% - 92% of requests.

We define the correlation distance between URLs A and B in two ways: either (i) the Euclidean distance between the points in the L-dimension space that represent the access distribution of URL A and B, or (ii) the complement of *cosine vector similarity* of spatial access vector A and B.

$$correl\_dist(A, B) = 1 - vector\_similarity(A, B) = 1 - \frac{\sum_{i=1}^{k} A_i \times B_i}{\sqrt{\sum_{i=1}^{k}(A_i)^2 \times \sum_{i=1}^{k}(B_i)^2}} \quad (5.1)$$

Essentially, if we view each spatial access vector as an arrow in high-dimension space, the vector similarity gives the cosine of the angle formed by the two arrows.

Temporal Clustering

In this section, we examine temporal clustering, which clusters Web content based on temporal locality of the access pattern.

There are many ways to define temporal locality. One possibility is to divide the traces into n time slots, and assign a temporal access vector to each URL, where the element i is the number of accesses to that URL from the time slot i. Then we can use similar methods in spatial clustering to define the correlation distance. However, in our experiments we found that many URLs share similar temporal access vectors because of specific events, but they are not accessed together. One typical example is in the event-driven WorldCup trace, where the corresponding URLs in English and French have very similar temporal access patterns during game time, but as expected are almost never fetched together by any client groups.

To address this issue, we consider URLs are correlated only if they are requested in a short period by the same client. In particular, we extend the co-occurrence based clustering by Su, *et al.* [125]. At a high-level, the algorithm divides requests from a client into variable length sessions, and only considers URLs requested together during a client's session to be related. We make the following enhancements: (i) we empirically determine the session boundary rather than choose an arbitrary time interval; (ii) we quantify the similarity in documents' temporal locality using the co-occurrence frequency.

Determine session boundaries: First, we need to determine user sessions, where a session refers to a sequence of requests initiated by a user without pro-longed pauses in between. We apply the heuristic described in [3] to detect the session boundary: we consider a session has ended if it is idle for sufficiently long time (called *session-inactivity period*); and we empirically determine the session inactivity period as the knee point where the change in its value does not yield a significant change in the total number of sessions [3]. Both the MSNBC and NASA traces have the session-inactivity period as 10 - 15 minutes, and we choose 12 minutes in our simulations.

Correlation in temporal locality: We compute the correlation distance between any two URLs based on the co-occurrence frequency (see Algorithm 6). This reflects the similarity in their temporal locality and thus the likelihood of being retrieved together by a client. Assume that we partition the traces into p sessions. The number of co-occurrences of A and B in the session i is denoted as $f_i(A, B)$, which is calculated by counting the number of interleaving access pairs (not necessarily adjacent) for A and B.

```
   procedure TemporalCorrelationDistance()
1  foreach session with access sequence (s_1, s_2, ... s_n) do
2      for i = 1; i ≤ n-1; i++ do
3          for j = i+1; j ≤ n; j++ do
4              if s_i ≠ s_j then  f_i(s_i, s_j)++; f_i(s_j, s_i)++;
5              else exit the inner for loop to avoid counting duplicate pairs
           end
       end
   end
6  foreach URL A do  compute the number of occurrences o(A)
7  foreach pair of URLs (A, B) do
8      Co-occurrence values f(A,B) = Σ_{i=1}^{p} f_i(A,B)
9      Co-occurrence frequency c(A,B) = f(A,B) / (o(A)+o(B))
10     Correlation distance correl_dist(A,B) = 1 − c(A,B)
   end
```

Algorithm 6: Temporal Correlation Distance Computation

Steps 2 to 5 of Algorithm 6 computes $f_i(A, B)$. For example, if the access sequence is "*ABCCA*" in session i. The interleaving access pairs for A and B are AB and BA, so $f_i(A, B) = f_i(B, A) = 2$. Similarly, $f_i(A, C) = f_i(C, A) = 3$, $f_i(B, C) = f_i(C, B) = 2$. Note that in Step 8 and 9, since $f(A, B)$ is symmetric, so is $c(A, B)$. Moreover, $0 \leq c(A, B) \leq 1$ and $c(A, A) = 1$. The larger the $c(A, B)$, the more closely correlated the two URLs are, and the more likely they are to be accessed together. Step 10 reflects the property that distance decreases with the increase in the correlation.

Popularity-based Clustering

Finally, we consider the approach of clustering URLs by their access frequency. We consider two metrics. The first correlation distance metric is defined as

$$correl\_dist(A, B) = |access\_freq(A) - access\_freq(B)|$$

The second distance metric is even simpler. If N URLs are to be clustered into K

clusters, we sort them according to the total number of accesses, and place URLs 1 ... $\lfloor\frac{N}{K}\rfloor$ into cluster 1, and URLs $\lfloor\frac{N}{K}\rfloor$+1 ... $\lfloor\frac{2N}{K}\rfloor$ into cluster 2, and so on.

We tested both metrics on MSNBC traces and they yield very similar results. Therefore we only use the simpler approach for evaluation in the rest of the chapter.

Traces Collection for Clustering

The three clustering techniques all require access statistics, which can be collected at CDN name servers or CDN servers. The popularity-based clustering requires the least amount of information: only the hit counts of the popular Web objects. In comparison, the temporal clustering requires the most fine-grained information – the number of co-occurrences of popular objects, which can be calculated based on the access time, and source IP address for each request. The spatial clustering is in between the two: for each popular Web object, it needs to know how many requests are generated from each popular client group, where the client groups are determined using BGP prefixes collected over widely dispersed routers [67].

5.8 Performance of Cluster-based Replication

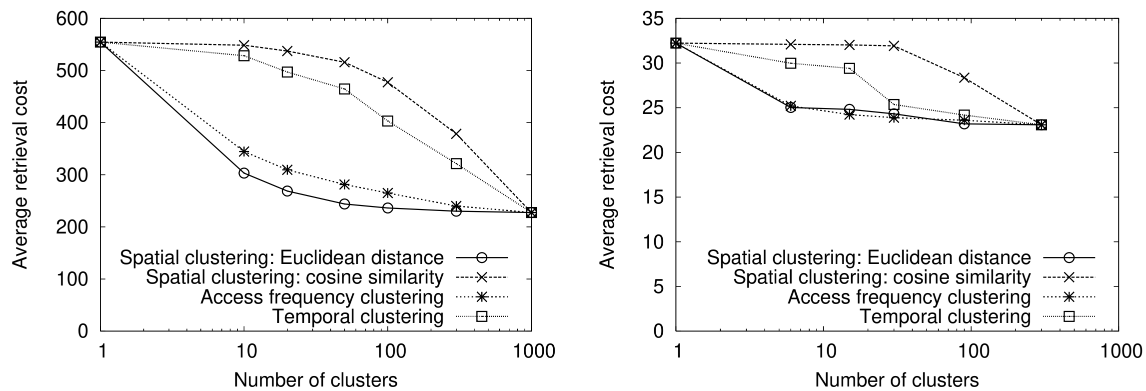
In this section, we evaluate the performance of different clustering algorithms on a variety of network topologies using the real Web server traces.

5.8.1 Performance Comparison of Various Clustering Schemes

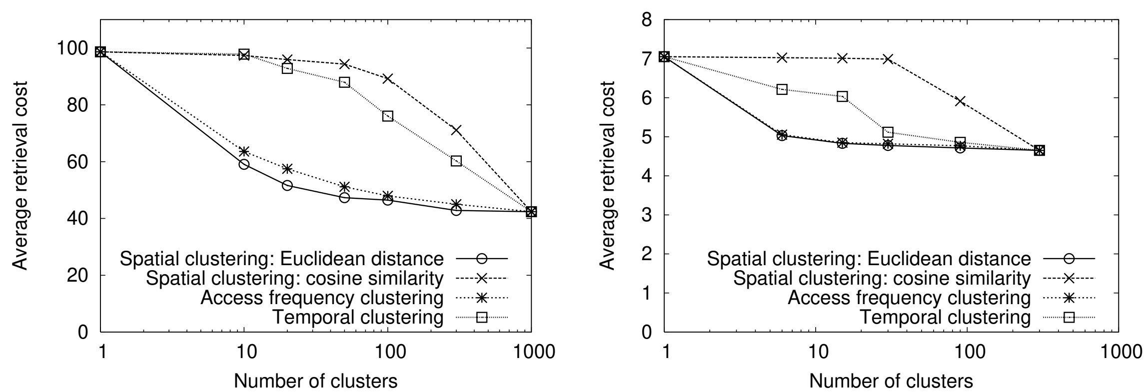
First we compare the performance of various cluster-based algorithms. In our simulations, we use the top 1000 URLs from the MSNBC traces covering 95% of requests, and the top 300 URLs from the NASA traces covering 91% of requests. The replication algorithm we use is similar to Algorithm 5 in Chapter 5.6. In the iteration step 7, we choose the $< cluster, location >$ pair that gives the largest performance gain per URL.

Figure 5.6 compares the performance of various clustering schemes. The starting points of all the clustering performance curves represent the single cluster case, which corresponds to per Web site-based replication. The end points represent per URL-based replication, another extreme scenario where each URL is a cluster.

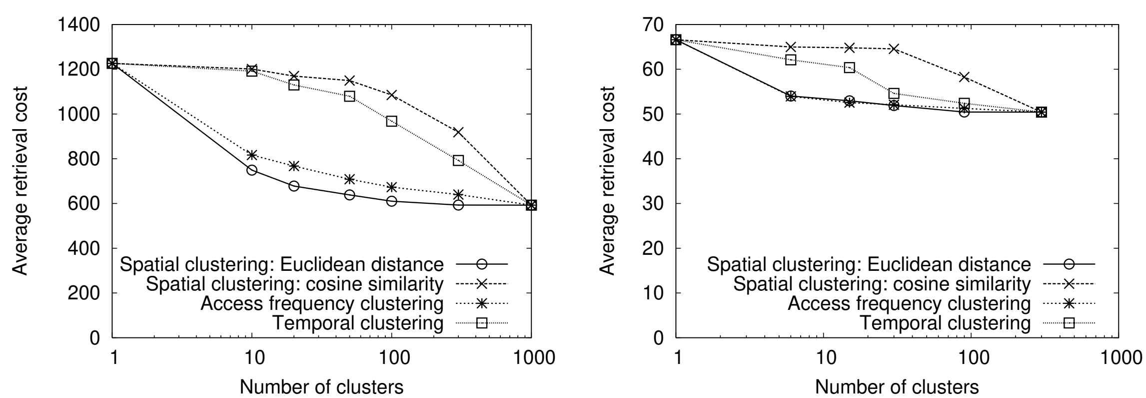
As we can see, the clustering schemes are efficient. Even with the constraint of a small number of clusters (i.e., 1% - 2% of the number of Web pages), spatial clustering based on Euclidean distance between access vectors and popularity-based clustering achieve performance close to that of the per URL-based replication, at much lower management cost (see Chapter 5.7). Spatial clustering with cosine similarity and temporal clustering do not perform as well. It is interesting that although the popularity-based clustering does not capture variations in individual clients' access patterns, it achieves comparable and sometimes better performance than the more fine-grained approaches. A possible reason is that many popular documents are globally popular [132], and access frequency becomes the most important metric that captures different documents' access pattern.



(a) On a pure random topology



(b) On a transit-stub topology



(c) On an AS-level topology

Figure 5.6: Performance of various clustering approaches for MSNBC August 2, 1999 traces with averagely 5 replicas/URL (left) and for NASA July 1, 1995 traces with averagely 3 replicas/URL (right) on various topologies.

The relative rankings of various schemes are consistent across different network topologies. The performance difference is smaller in the AS topology, because the distance between pairs of nodes is not as widely distributed as in the other topologies.

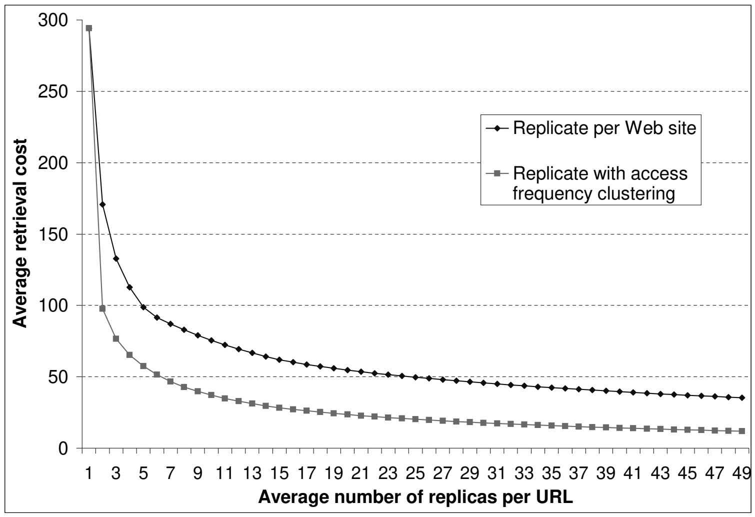


Figure 5.7: Performance of cluster-based replication for MSNBC August 2, 1999 traces (in 20 clusters) with up to 50 replica/URL on a transit-stub topology

We also evaluate the performance of cluster-based replication by varying the replication cost (i.e., the average number of replicas/URL). Figure 5.7 shows the performance results when we use the access frequency clustering scheme and 20 content clusters. As before, cluster-based scheme out-performs per Web site scheme by over 50%. As expected the performance gap between per Web site and per cluster replication decreases as the number of replicas per URL increases. Compared to per URL-based replication, the cluster-based replication is more scalable: it reduces running time by over 20 times, and reduces the amount of state by about 50 times.

5.8.2 Effects of Non-Uniform File size

So far, we assume each replicated URL consumes one unit of replication cost. In this section, we compute the replication cost by taking into account of different URL sizes. The cost of replicating a URL is its file size. We modify Algorithm 5 in Chapter 5.6 so that in iteration step 7, we choose the $< cluster, location >$ pair that gives the largest performance gain per byte.

We ran the experiments using the top 1000 URLs of the MSNBC traces on August 2, 1999. Table 5.4 shows the performance of the Euclidean distance based spatial clustering with the cost of 3 Website replicas on a transit stub topology. The results exhibit a similar trend as those obtained under the assumption of uniform URL size: per URL-based replication out-performs per Web site-based replication by 40%, and the cluster-based schemes (50 clusters) achieve similar performance as per URL-based replication (1000 clusters) with only about 5% management cost if we ignore the cost for clustering.

Per site	10 clusters	50 clusters	300 clusters	Per URL
132.7	108.1	84.7	81.3	80.4

Table 5.4: Average retrieval cost with non-uniform file size

Methods	Static 1	Static 2	Optimal
Traces used for clustering	training	training	new
Traces used for replication	training	new	new
Traces used for evaluation	new	new	new

Table 5.5: Static and optimal clustering schemes

5.9 Incremental Clustering

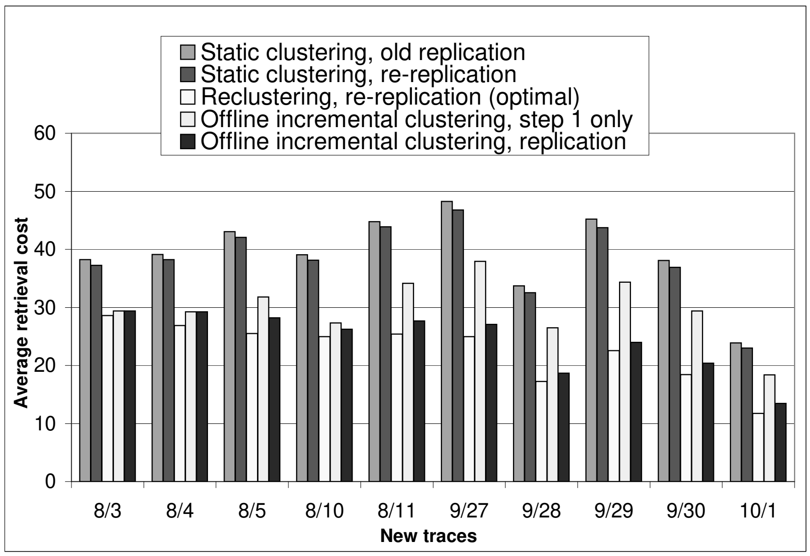
In the previous sections, we have presented cluster-based replication, and showed it is flexible and can smoothly trade off replication cost for better user performance. In this section we examine how the cluster-based replication scheme adapts to changes in users' access patterns. One option is to re-distribute the existing content clusters without changing the clusters. We call it *static clustering* (Chapter 5.9.1). A better alternative, termed *incremental clustering*, gradually adds new popular URLs to existing clusters and replicates them (Chapter 5.9.2). We can determine new popular URLs either by observing users' accesses (offline) or by predicting future accesses (online). Below we will study different ways to adapt to changes in user workload, and compare their performance and cost.

5.9.1 Static Clustering

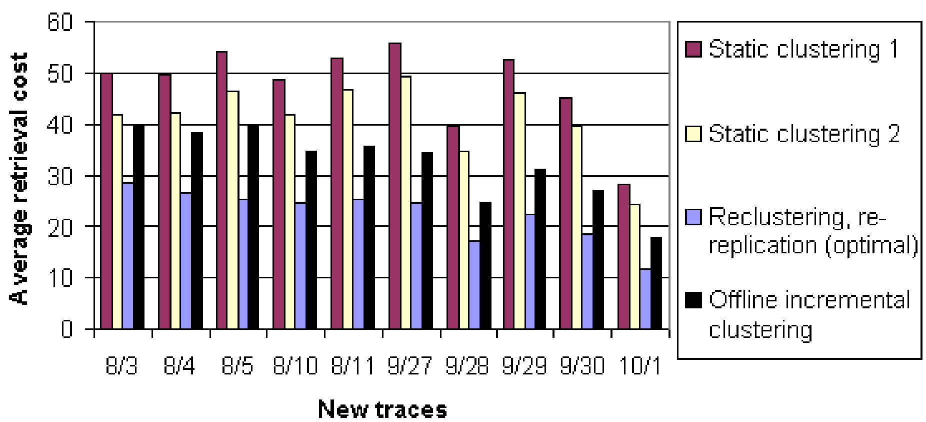
It is important to determine the frequency of cluster perturbation and redistribution. If the clients' interested URLs and access patterns change very fast, a fine-grained replication scheme that considers how a client retrieves multiple URLs together may require frequent adaptation. The extra maintenance and clustering cost may dictate that per Web site replication approach should be used instead. To investigate whether this would be a serious concern, we evaluate three methods, as shown in Table 5.5 using MSNBC traces: *training traces* and *new traces*, where the training traces and new traces are access traces for Day 1 and Day 2, respectively (Day 2 follows Day 1 either immediately or a few days apart).

Note that in the static 1 and static 2 methods, accesses to the URLs that are not included in the training traces have to go to the origin Web server, potentially incurring a higher cost. We consider the spatial clustering based on Euclidean distance (referred as SC) and popularity (i.e., access frequency) based clustering (referred as AFC), the two with the best performance in Chapter 5.8. We simulate on pure random, Waxman, transit-stub, and AS topologies. The results for different topologies are similar, and below we only present the results from transit-stub topologies.

We use the following simulation configuration throughout this section unless otherwise specified. We use 8/2/99 MSNBC traces as the training traces, and use 8/3/99, 8/4/99, 8/5/99, 8/10/99, 8/11/99, 9/27/99, 9/28/99, 9/29/99, 9/30/99 and 10/1/99 traces as the new traces. We choose the top 1000 client groups from the training traces, and they



(a) Cluster based on the Euclidean distance of spatial vector.



(b) Cluster based on the access frequency.

Figure 5.8: Stability analysis of the per cluster replication for MSNBC 1999 traces with 8/2/99 as training traces (averagely 5 replicas/URL).

have over 70% overlap with the top 1000 client groups in the new traces. So we use the same set of client groups for our performance evaluation in this section. To study the dynamics of content, we choose the top 1000 URLs from each daily traces. We use SC or AFC to cluster them into 20 groups when applicable.

As shown in Figure 5.8, using the past workload information performs significantly worse than using the actual workload. The average retrieval cost almost doubles when the time gap is more than a week. The performance of AFC is about 15-30% worse than that of SC for static 1 method and 6-12% worse for static 2 method. In addition, as we would expect, the difference in the performance gap increases with the time gap. The redistribution of old clusters based on the new traces does not help for SC, and helps to improve 12-16% for AFC. The increase in the clients' latency is largely due to the creation of new contents, which have to be fetched from the origin site according to our assumption. (The numbers of new URLs are shown in row 1 of Table 5.6.) In the next section, we will use various incremental clustering to address this issue.

5.9.2 Incremental Clustering

In this section, we examine how to incrementally add new documents to existing clusters without much perturbation. First, we formulate the problem, and set up framework for generic incremental clustering. Then we investigate both online and offline incremental clustering. The former predicts access patterns of new objects based on hyperlink structures, while the latter assumes such access information is available. Finally, we compare their performance and management cost with the complete re-clustering and re-distribution.

Problem Formulation

We define the problem of *incremental clustering* for distributed replication system as follows. Given N URLs, initially they are partitioned into K clusters and replicated to various locations to minimize the total cost of all clients' requests. The total number of URL replicas created is T. After some time, V of the original objects become cold when the number of requests to them drops below a certain threshold, while W new popular objects emerge, and need to be clustered and replicated to achieve good performance. To prevent the number of replicas T from increasing dramatically, we can either explicitly reclaim the cold object replicas or implicitly replace them through policies such as LRU and LFU. For simplicity, we adopt the latter approach. The replication cost is defined as the total number of replicas distributed for new popular objects.

One possible approach is to completely re-cluster and re-replicate the new $(N - V + W)$ objects, as the optimal scheme in Table 5.5. One possible approach is to completely re-cluster and re-replicate the new $(N - V + W)$ objects as the third scheme described in Chapter 5.9.1. However this approach is undesirable in practice, because it requires reshuffling the replicated objects and re-building the content directory, which incurs extra replication traffic and management cost. Therefore our goal is to find a replication strategy that balances the tradeoff between replication and management cost versus clients' performance.

Incremental clustering takes the following two steps:

Row	Date of new traces in 1999	8/3	8/4	8/5	8/10	8/11	9/27	9/28	9/29	9/30	10/1
1	# of new popular URLs	315	389	431	489	483	539	538	530	526	523
2	# of cold URL replicas freed	948	1205	1391	1606	1582	1772	1805	1748	1761	1739
3	# of orphan URLs when using $\lvert\overrightarrow{v_{new}} - \overrightarrow{v_c}\rvert > r$	0	0	2	1	1	6	4	6	8	6
4	# of orphan URLs when using $\lvert\overrightarrow{v_{new}} - \overrightarrow{v_c}\rvert > r_{max}$	0	0	2	0	1	6	4	6	7	5
5	# of new URL replicas deployed for non-orphan URLs ($\lvert\overrightarrow{v_{new}} - \overrightarrow{v_c}\rvert \leq r$)	983	1091	1050	1521	1503	1618	1621	1610	1592	1551
6	# of new clusters generated for orphan URLs ($\lvert\overrightarrow{v_{new}} - \overrightarrow{v_c}\rvert > r$)	0	0	2	1	1	3	3	3	3	3
7	# of URL replicas deployed for orphan URLs ($\lvert\overrightarrow{v_{new}} - \overrightarrow{v_c}\rvert > r$): row 2 - row 5 if row 2 > row 5	0	0	341	85	79	154	184	138	169	188
8	# of new URL replicas deployed (Access frequency clustering)	1329	1492	1499	1742	1574	2087	1774	1973	1936	2133

Table 5.6: Statistics and cost evaluation for offline incremental clustering. Using MSNBC traces with 8/2/99 as training traces, 20 clusters, and averagely 5 replicas/URL. Results for clustering based on SC (row 3 - 7) and AFC (row 8).

Crawled time on 5/3/2002	8am	10am	1pm
# of crawled URLs (non image files)	4016	4019	4082
# of URL clusters (clustering with the same parent URL)	531	535	633

Table 5.7: Statistics and clustering of crawled MSNBC traces

STEP 1: If the correlation between the new URL and an existing content cluster exceeds a threshold, add the new URL to the cluster that has the highest correlation.

STEP 2: If there are still new URLs left (referred as *orphan URLs*), create new clusters and replicate them.

Online Incremental Clustering

Pushing newly created documents are useful during unexpected flash crowds events, such as disasters. Without clients' access information, we predict access pattern of new documents using the following two methods based on hyperlink structures.

1. Cluster URLs based on their parent URLs, where we say URL a is URL b's parent if a has a hyperlink pointing to b. There is one exception: many URLs point back to the root index page, but the root page should not be included in any children cluster because its popularity differs significantly from other pages.

2. Cluster URLs based on their *hyperlink depth*. The hyperlink depth of URL o is defined as the *smallest* number of hyperlinks needed to traverse before reaching o, starting from the root page of the Web server.

In our evaluation, we use WebReaper 9.0 [130] to crawl http://www.msnbc.com/ at 8 a.m., 10 a.m. and 1 p.m. (PDT time), respectively, on May 3, 2002. Given a URL, the WebReaper downloads and parses the page. Then it recursively downloads the pages pointed by the URL until a pre-defined hyperlink depth is reached. We set the depth to be 3 in our experiment. We ignore any URLs outside www.msnbc.com except the outsourced images. Since we also consider the URLs pointed by all the crawled documents, our analysis includes all pages within 4 hyperlink distance away from the root page. Clustering based on the hyperlink depth generates 4 clusters, e.g., depth = 1, 2, 3, and 4 (exclusive of the root page). The access logs do not record accesses to image files, such as .gif and .jpg. We have the access information for the remaining URLs, whose statistics are shown in Table 5.7. In general, about 60% of these URLs are accessed within the next two hours after crawling.

To measure the popularity correlation within a cluster, we define *access frequency span* (in short, af_span) as follows.

$$af\_span = \frac{standard\ deviation\ of\ access\ frequency}{average\ access\ frequency}$$

We have MSNBC access logs from 8 a.m. - 12 p.m. and 1 p.m. - 3 p.m. on May 3, 2002. For every hour during 8 a.m. - 12 p.m. and 1 p.m. - 3 p.m., we use the most recently crawled files to cluster content, and then use the access frequency recorded in the corresponding access logs to compute af_span for each cluster. We also compute the average, 10 percentile and 90 percentile of af_span in all clusters, and show the results in Figure 5.9.

In Figure 5.9, both clustering methods show much better popularity correlation (i.e., smaller af_span) than treating all URLs (except the root) as a single cluster. Method 1 consistently out-performs method 2. Based on the observation, we design an online incremental clustering algorithm as follows. We assign each new URL o to the existing cluster that has the largest number of URLs sharing the same parent URL with o (i.e., the largest number of sibling URLs). If there are ties, we are conservative, and pick the cluster that has the largest number of replicas. Note that o may have multiple parents, so we consider all the children of its parents as its siblings except the root page. When a new URL o is assigned to cluster c, we replicate o to all the replicas to which cluster c has been replicated.

We simulate the approach on a 1000-node transit-stub topology as follows. First, among all the URLs crawled at 8am, 2496 of them were accessed during 8am - 10am. We use AFC to cluster and replicate them based on the 8am - 10am access logs, with 5 replicas per URL on average. Among those new URLs that appear in the 10am crawling, but not in the 8am crawling, 16 of them were accessed during 10am - 12pm. Some of them were quite popular, receiving 33262 requests in total during 10am - 12pm. We use the online incremental clustering algorithms above to cluster and replicate the 16 new URLs with a replication cost of 406 URL replicas. This yields an average retrieval cost of 56. We also apply the static AFC by using 10am - 12pm access logs, and completely re-clustering and re-replicating all these 2512 (2496 + 16) URLs, with 5 replicas per URL on average. As it requires information about future workload and complete re-clustering and re-replication of content, it serves as the optimal case, and yields an average retrieval cost of 26.2 for the 16 new URLs. However, if the new URLs are not pushed but only cached after it is accessed, the average retrieval cost becomes 457; and if we replicate the 16 new URLs to random places using the same replication cost as in the online incremental clustering (406 URL replicas), the average retrieval cost becomes 259.

These results show that the online incremental clustering and replication cuts the retrieval cost by 4.6 times compared to random pushing, and by 8 times compared to no push. Compared to the optimal case, the retrieval cost doubles. But since it requires no access history nor complete re-clustering or replication, such performance is quite good.

Offline Incremental Clustering

Now we study offline incremental clustering, which uses access history as input.

STEP 1: In the SC clustering, when creating clusters for the training traces, we record the center and diameter of each cluster. Given a cluster U with p URLs, each URL u_i is identified by its spatial access vector $\overrightarrow{v_i}$ and $correlation\_distance(u_i, u_j) = |\overrightarrow{v_i} - \overrightarrow{v_j}|$. We define the center c as $\frac{\sum_{i=1}^{p} \overrightarrow{v_i}}{p}$. The radius r is $max_i(|\overrightarrow{v_c} - \overrightarrow{v_i}|)$, which is the maximum Euclidean distance between the center and any URL in U. For each new URL $\overrightarrow{v_{new}}$, we add it to an existing cluster U whose center c is closest to $\overrightarrow{v_{new}}$, if either $|\overrightarrow{v_{new}} - \overrightarrow{v_c}| < r$ or $|\overrightarrow{v_{new}} - \overrightarrow{v_c}| < r_{max}$ is satisfied, where r is the radius of cluster U, and r_{max} is the maximum radius of all clusters.

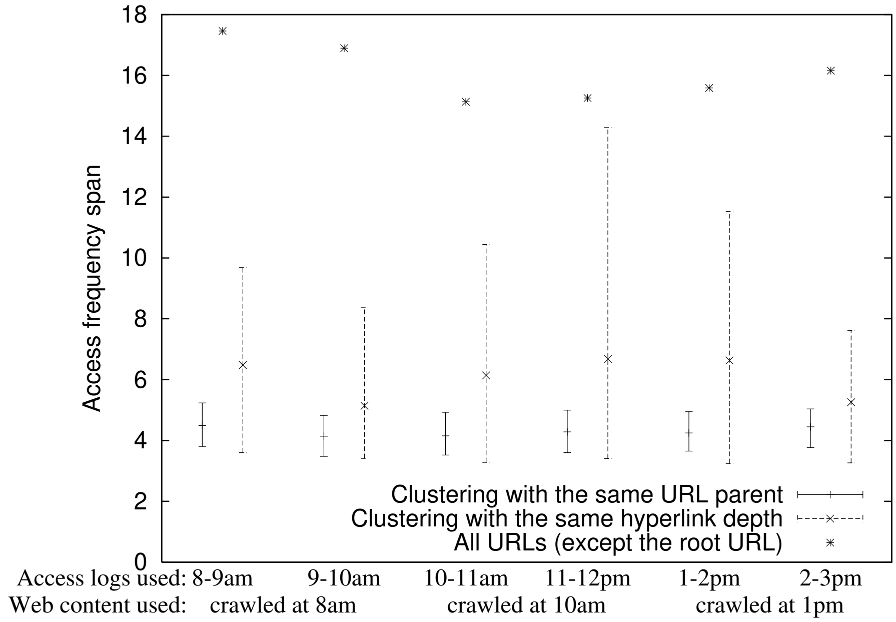


Figure 5.9: Popularity correlation analysis for semantics-based clustering. The error bar shows the average, 10 and 90 percentile of af_span.

Our analysis of MSNBC traces shows that most of the new URLs can find their homes in old clusters (as shown in rows 3 and 4 of Table 5.6); this implies the spatial access vector of most URLs are quite stable, even after about two months. Furthermore, the difference between using $|\overrightarrow{v_{new}} - \overrightarrow{v_c}| < r$ and $|\overrightarrow{v_{new}} - \overrightarrow{v_c}| < r_{max}$ is insignificant. So we consider the former in the remaining of this section. Once a new URL is assigned to a cluster, the URL is replicated to all the replicas to which the cluster has been replicated. Row 5 of Table 5.6 shows the number of new URL replicas.

procedure IncrementalClusteringReplication_OrphanURLs()

1 Compute and record the biggest diameter d of original clusters from the training traces
2 Use limit diameter clustering (Chapter 5.7) to cluster the orphan URLs into K' clusters with diameter d
3 l = number of cold URL replicas freed - number of URL replicas deployed for non-orphan URLs in Step 1.
4 **if** $l > 0$ **then** replicate the K' clusters with l replicas
5 **else** Replicate the K' clusters with l' replicas, where l' = number of orphan URLs $\times$ average number of replicas per URL

Algorithm 7: Incremental Clustering and Replication for Orphan URLs (Spatial Clustering)

In the AFC clustering, the correlation between URLs is computed using their ranks in access frequency. Given K clusters sorted in decreasing order of popularity, a new

URL of rank i (in the new traces) is assigned to $\lceil \frac{i}{K} \rceil$th cluster. In this case, all new URLs can be assigned to one of the existing clusters, and step 2 is unnecessary.

Figure 5.8 shows the performance after the completion of Step 1. As we can see, incremental clustering has improvement over static clustering by 20% for SC, and 30-40% for AFC. At this stage, SC and AFC have similar performance. But notice that AFC has replicated all the new URLs while SC still has orphan URLs for the next step. In addition, AFC deploys more new URL replicas (row 8 of Table 5.6) than SC (row 5 of Table 5.6) at this stage.

STEP 2: We further improve the performance by clustering and replicating the orphan URLs. Our goal is (1) to maintain the worst-case correlation between existing clusters after adding new ones, and (2) to prevent the total number of URL replicas from increasing dramatically due to replication of new URLs. Step 2 only applies to SC, and we use Algorithm 7.

Row 6 and 7 in Table 5.6 show the number of new clusters generated by orphan URLs and the number of URL replicas deployed for the orphan URLs. As Figure 5.8 (top) shows, SC out-performs AFC by about 20% after step 2, and achieves comparable performance to complete re-clustering and re-replication, while using only 30 - 40% of the replication cost compared to the complete re-clustering and re-replication. (The total replication cost of the latter scheme is 4000 URL replicas: 1000 URLs $\times$ 5 replicas/URL, except 1000 URL replicas residing at the origin Web server.)

To summarize, in this section, we study online and offline incremental clustering, and show they are very effective in improving users' perceived performance with small replication cost.

5.10 Summary

In this chapter, we explore how to efficiently push content to CDN nodes for cooperative access. Using trace-driven simulations, we show that replicating content in units of URLs out-performs replicating in units of Web sites by 60 - 70%. To address the scalability issue of such a fine-grained replication, we examine several clustering schemes to group the Web documents and replicate them in units of clusters. Our evaluations based on various topologies and Web server traces show that we can achieve performance comparable to per URL-based replication at only 1% - 2% of the management cost. To adapt to changes in users' access patterns, we consider both offline and online incremental clustering. Our results show that the offline clustering yields the performance close to that of the complete re-clustering at much lower overhead; the online incremental clustering and replication reduce the retrieval cost by 4.6 - 8 times compared to no replication and random replication.

In particular, for the content without access history (e.g., newly created URLs), we can incrementally add them to existing content clusters based on hyperlink structures, and push them to the locations to which the cluster has been replicated. This online incremental cluster-based replication is very useful to improve document availability during flash crowds.

In conclusion, our main contributions of this chapter include (i) cluster-based replication schemes to smoothly trade off management and computation cost for better clients' performance in a CDN environment, (ii) an incremental clustering framework to adapt to changes in users' access patterns, and (iii) an online popularity prediction scheme based on hyperlink structures.

In summary, in Part I, we present how to locate, replicate and manage contents scalably, efficiently, and adaptively to users' dynamics. In the next part, we will discuss overlay network measurement and monitoring services for providing adaptation on the network and system levels.

Part II

Overlay Network Measurement and Monitoring

In the first part of the thesis, we built a CDN which is scalable, efficient, and adaptive to users' dynamics. In this part, we will shift our focus to network dynamics. Our technique is generally applicable to the emerging class of large-scale globally overlay network services and applications, such as CDN, overlay routing and location, application-level multicast, virtual private network (VPN) management, and peer-to-peer file sharing. As these systems have flexibility in choosing their communication paths and targets, they can benefit significantly from network distance (*e.g.*, latency and loss rate) prediction. For instance, CDN can achieve proximity and fault tolerance through selecting or switching to the "best" CDN edge server to satisfy each client's request.

Unfortunately, the infrastructure ossification and lack of understanding of the Internet artifact remain to be key challenges for network distance estimation. For the first challenge, the ability to deploy innovative disruptive technologies in the core infrastructure (which is operated mainly by businesses) is extremely limited. Consequently, various forms of overlay networks, which introduce new functionality within the network near the edges, have been proposed and some deployed in the Internet. Thus we design our network measurement and monitoring services on the overlay network. Even though, it has proved difficult to characterize, understand, and model the enormous volume and great variety of Internet traffic in terms of large-scale behaviors.

In the second part, we will explore these challenges to build scalable and accurate end-to-end network distance monitoring systems. We will first survey previous work and their limitations in Chapter 6. Then to address their shortcomings, we present a latency monitoring system, Internet Iso-bar, in Chapter 7 and a loss rate monitoring system, TOM, in Chapter 8. To demonstrate how Internet applications can benefit from such monitoring services, we build and evaluate a monitoring-based adaptive overlay streaming media system in Chapter 9.

Chapter 6

Previous Work

Among the distance metrics, latency is a common one, and the easiest one to measure and scale, thus there is much prior work [4, 51, 127, 83]. Measurement of other general metrics, *e.g.*, loss rate, bandwidth, jitter, *etc.* are much more challenging and harder to scale. Next, we group existing systems into these two categories, and compare them with the network measurement and monitoring systems of our own.

6.1 Latency Estimation Systems

A latency estimation system can be built as an infrastructure service [4, 51, 127] or a peer-to-peer service [83]. The former assumes dedicated infrastructure support, and can provide continuous dynamic monitoring, while the latter assumes that the end clients measure the distance with little support from the service providers. We compare them with our Internet Isobar (Chapter 7) in Table 6.1.

6.1.1 Infrastructure-based Latency Estimation Systems

Both industry [4] and academia [51, 127] have built infrastructure-based latency estimation systems. But they are either not scalable [4], or based on triangulation inequality which doesn't really hold in today's Internet [51], or not very accurate for latency estimation [51, 127], or assuming the wide existence of measurement servers [127]. Next, we discuss these systems in more details.

Content Distribution Network (Akamai's Network Operations Command Center (NOCC))

For each client address prefix (subnet), Akamai performs traceroute from all CDN servers to find a few core routers (close to clients) that are always on the path to client clusters. They constantly monitor the distance from every Internet Data Center (IDC) which hosts the CDN edge servers, to these routers to decide the relative distance to clients [70] [1]. Although working in real operation, this approach has a potential scalability problem.

[1] As a proprietary technique, there is no white paper available regarding this subject, to the best of our knowledge.

Properties	Akamai NOCC	IDMaps	Network Distance Maps	GNP	**Internet Isobar**
Type	Infrastructure	Infrastructure	Infrastructure	Peer-to-peer	**Infrastructure**
Scalability	Traceroute from each edge server farm to all client subnets	Proximity-based clustering of APs, $O(K^2 + AP)$ measurements	Proximity-based hierarchical clustering, $O(N)$ measurements	NL measurements (each landmark takes $O(N)$ of them)	**Similarity-based clustering, $K^2 + N$ measurements**
Estimation technique	Use distance between edge server farms to nearby router of clients	Based on triangulation inequality and proximity-based clustering	Proximity-based clustering, dist(h_1, h_2)=maximum (dist($m(h_1)$, $m(h_2)$), dist($m(h_1)$, h_1), dist($m(h_2)$, h_2))	Based on high-dimension coordinates, symmetric distance	**Similarity-based clustering, different inter- & intra- cluster estimation**
Monitors /landmarks deployment	CDN edge servers	Transit AS's	Traceroute servers	End hosts	**End hosts**

Table 6.1: Comparison of various Internet latency estimation systems, assuming there are N end hosts, AP address prefixes, L landmarks and K clusters, $m(h_i)$ is the monitor of the cluster to which host h_i belongs.

There are more than 8800 Internet Service Providers (ISPs). Suppose every ISP has an IDC for hosting CDN servers and clients are grouped by autonomous systems (AS's), we need 114 million traceroute measurements to build distance maps among 13,000 existing AS's. A similar amount of measurements is needed to maintain the distance maps. Though they divide the maps into regions, and only measure the distance between the IDCs and core routers of AS's in each region [70], the amount of measurements is only reduced by the factor of the number of regions (assume the ISPs and AS's are evenly distributed in the regions). While helpful when the number of regions is big, this approach will lose agility because the CDN servers in one region cannot serve the clients in other regions.

Internet Distance Maps (IDMaps)

IDMaps has special HOPS servers (called *Tracers*) that measure the distances among themselves and to other end hosts [51]. The distance between two end hosts A and B is estimated as the sum of 3 distances: distance between A and its nearest Tracer T_1, distance between B and its nearest Tracer T_2 and the shortest path distance between T_1 and T_2. To achieve scalability, they group the clients by Address Prefix (AP) which is a consecutive address range of IP addresses within which all hosts are equidistant (with some tolerance) to the rest of Internet. This grouping scheme leads to hundreds of thousands of APs, which are further clustered based on network proximity. The problem is that such a clustering requires pair-wise distance between *all* AP pairs. As a heuristic, Tracers are placed on transit AS's. However, this requires the cooperation of network providers. IDMaps also faces problems with prediction accuracy. First, their estimation is based on the triangulation inequality, which does not hold unless Tracers are placed on or very close to the shortest path between clients. Second, this proximity-based clustering is not as accurate as the similarity-based clustering used by Internet Iso-bar as we will show in Chapter 7.4.

Network Distance Maps

To tackle the scalability problem, Theilmann and Rothermel have proposed *network distance maps* [127]. Their approach creates a hierarchical decomposition of the network into regions, assigns each host to its closest measurement server (mServers) and estimates the distance between two hosts by the distance between their two assigned closest mServers. The maps can also be adapted to the changing network conditions.

However, there are two key features that distinguish our Internet Isobar (Chapter 7) from theirs. Firstly, they assume the existence of measurement servers (mServers) and do not consider the problem of placing the monitoring sites. Secondly, they assign the client to its closest mServer, which is equivalent to proximity-based clustering. Our analysis based on real measurement data shows that the similarity-based clustering has much better estimation accuracy and stability than the proximity-based clustering does (Chapter 7.4).

6.1.2 Peer-to-peer Latency Estimation Systems

Peer-to-peer latency estimation systems associate each node with a coordinate, and use the Euclidean distance between the coordinates to estimate their network distance between the nodes. They use either geographical coordinates [59] or coordinates computed based on the network distance to a few landmark nodes [83, 99]. The fundamental problem of these schemes is the assumption of symmetric distance.

Geographical-based distance estimation

Geographical distance was used to predict the real network distance in [59]. However, results show that the correlation between geographical distances and network distances is quite poor, especially between different network backbones [59, 83].

Global Network Positioning (GNP)

GNP is based on absolute coordinates computed from modelling the Internet as a D-dimensional geometric space [83]. Every end host maintains its own coordinates, and network distances to other hosts are predicted by evaluating a *distance function* (e.g., Euclidean distance) over their coordinates. For static estimation, it is much more accurate than IDMaps. However, the landmark sites are potential bottlenecks because every host has to measure its distance to the landmarks to compute and update its coordinates. Recently, a lighthouse system is proposed to overcome that limitation by calculating the coordinates with arbitrary set of reference points, then transforming the local coordinates to the global ones [99].

Moreover, it is hard to achieve real-time distance estimation – both source and destination have to obtain measurements to $D + 1$ landmarks (D is the dimension used in estimation), re-compute the coordinates, and exchange the coordinates to get the estimation. Thus it is not suitable as an online monitoring system. Moreover, GNP can only model symmetric distance, and its optimization algorithms are expensive to run.

6.2 Other Metrics Estimation Systems

The systems above provide good latency estimation under normal conditions, but they cannot estimate other metrics, such as loss rate, bandwidth, *etc.*. The fundamental idea for infrastructure-based services is to cluster the end hosts, and then use one from each cluster as the representative. Unless all the hosts in the same cluster take the same route as their representative monitor to reach other hosts, the representative cannot detect the congestion/failures for the hosts in its cluster.

Similarly for the peer-to-peer services like GNP, the coordinates of an end host are computed based on the latency to some landmark sites, and cannot really reflect the path conditions to other hosts.

In contrast, there is another class of overlay network distance estimation systems which can detect path congestion, outages and periods of degraded performance within seconds and have applications bypass them. In general, such systems are much harder to scale than the latency estimation systems because each path to be estimated may take different routes. Typical example is resilient overlay network (RON) [40]. People also proposed to leverage the underlying network topology to improve the measurement efficiency. But none solves the problem of monitor deployment for constructing an overlay network monitoring service. We propose our own system TOM in Chapter 8.

6.2.1 RON

A RON is an architecture that allows distributed Internet applications to detect and recover from path outages and periods of degraded performance within several seconds. They keep monitoring the functioning and quality of Internet paths between *every* pair of RON nodes. Thus it has very high accuracy for many metrics, like latency, loss rate and throughput, but also has serious scalability problem.

6.2.2 Topology-based Efficient Measurement

Shavitt, *et al.* use algebraic tools to compute the distances that are not explicitly measured [119]. Given certain "Tracer" stations deployed and some direct measurements among the Tracers, they search for path or path segments whose loss rates can be inferred from these measurements. Thus their focus is not on Tracer/path selection. Neither do they examine the topology measurement errors or the topology change problems.

Recently, Ozmutlu, *et al.* selected a minimal subset of paths to cover all links for monitoring, assuming link-by-link latency is available via end-to-end measurement [90]. Their approach has the following three limitations.

- Traceroute cannot give accurate link-by-link latency. Many routers in the Internet hide their identities.
- It is not applicable for loss rate, because it is difficult to estimate link-by-link loss rates from end-to-end measurements. Many applications, like streaming media and multiplayer gaming, are more sensitive to loss rate than latency.
- It assumes static routing paths and does not consider topology changes.

To address these problems, in the next two chapters, we will present our latency estimation system, Internet Iso-bar, and loss rate estimation system, TOM.

Chapter 7

Internet Iso-bar: A Scalable Overlay Network Latency Estimation System

7.1 Introduction

In this part, the general problem we want to study is as follows: Given N end hosts that may belong to different administrative domains and $O(N^2)$ paths among them, how to monitor and estimate the properties of these paths with small measurement overhead? Besides accurate, efficient, and scalable, a network distance monitoring system should also be incrementally deployable, and easy to use.

We will focus on the monitoring and estimation of latency in this chapter, and loss rate for the next chapter. The difference between these two metrics is: latency is easy to measure and easier to achieve measurement scalability, while the loss rate is more important for some applications. We will show why latency estimation scheme is not good enough for loss rate or congestion/failure estimation, then provide an efficient scheme of measuring the loss rate without sacrificing accuracy.

For latency estimation, we propose a clustering-based scheme: *Internet Iso-bar*, which clusters the end hosts based on the similarity in their distance to a small set of sites. The cluster centers are selected as monitoring sites for active and continuous probing. The distance between any pair of hosts is estimated using inter- or intra-cluster distances. No knowledge of network topology is needed.

We evaluate the estimation accuracy and stability of our scheme with real Internet measurements from NLANR [85]. Stability is examined on various time scales: daily, weekly, monthly, *etc.*. Our results show that Iso-bar has high estimation accuracy and stability, comparable to the state-of-the-art work, GNP [83]. In contrast to GNP, the distributed small measurement overhead of Iso-bar enables online monitoring which detects the majority of congestion/failures in real time with certain false positive. To the best of our knowledge, our work is the first attempt to evaluate the *stability* of various distance estimation schemes using real Internet measurements.

The rest of the chapter is organized as follows. The clustering and distance estima-

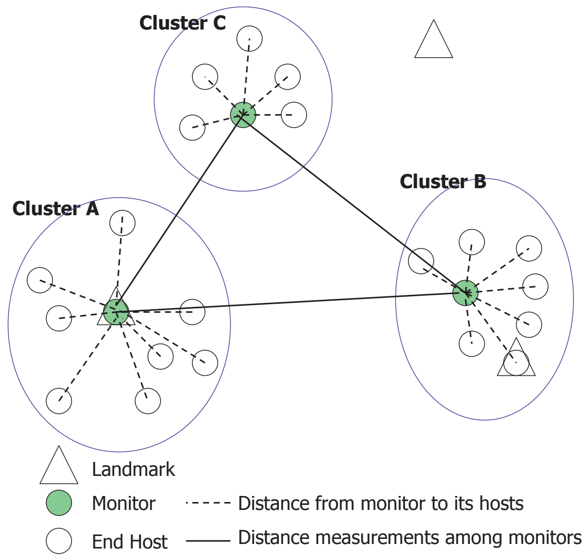


Figure 7.1: Internet Iso-bar architecture for a peer-to-peer system

tion techniques of Internet Iso-bar are presented in Chapter 7.2. We describe our simulation methodology in Chapter 7.3, and evaluate the accuracy and stability of different schemes in Chapter 7.4. Finally, we conclude in Chapter 7.5.

7.2 Architecture and Algorithms

The key idea of *Internet Iso-bar* (referred to as *Iso-bar* hereafter) is as follows. We group N hosts into K clusters based on several distance metrics (Chapter 7.2.1 and Chapter 7.2.2), and choose cluster centers as monitors (Chapter 7.2.2). The monitors continuously measure the distance among themselves as well as to hosts belonging to its cluster so that the distance between any pair of hosts can be estimated using intra- and inter-cluster distance (Chapter 7.2.3), as shown in Figure 7.1. Meanwhile, the measurement overhead is negligible (Chapter 7.2.4).

7.2.1 Distance Metrics

First we define the distance metrics for clustering. There are many ways to define Internet distance, including hop count, latency, loss rate, or bandwidth. In this chapter, we define the Internet distance to be the round trip time (RTT). Given asymmetric routing, RTT may not reflect the absolute network distance between two hosts. But in many cases, it still serves as a good approximation of the end-to-end distance we aim to estimate.

Using network proximity

Let $net\_dist(i,j)$ denote the measured network distance between host i and host j, and $cor\_dist(i,j)$ denote the correlation distance between them, i.e., the smaller $cor\_dist(i,j)$ is, the more i and j are correlated. We can directly use $net\_dist(i,j)$ for clustering (i.e.,

$cor\_dist(i, j) = net\_dist(i, j)$). This is adopted by IDMaps [51] and Network Distance Maps [127]. However, although it requires all pair-wise distance measurements, this scheme is not very accurate, as shown in Chapter 7.4. Next, we introduce the network similarity based clustering, which can significantly reduce the amount of measurements.

Using network similarity (Iso-bar)

As in GNP, each host i measures distance to landmarks and forms a *network distance vector* (referred as $netV_i$). However, unlike GNP, $netV_i$ is only used for initial clustering; and once clusters are formed, we use significantly fewer measurement probes (as will be shown in Chapter 7.2.4) for distance estimation.

$netV_i$ is a m-dimensional vector, where m is the number of landmarks. The landmarks can be all or a subset of the end hosts plus any outside servers that accept measurements.

Let w_k denote the k-th landmark, and $w_k(netV_i)$ denote the k-th element of the vector $netV_i$, which stands for the distance between host i and the k-th landmark. The correlation distance between two hosts can be defined as the Euclidean distance between their network distance vectors:

$$cor\_dist(i, j) = |netV_i - netV_j| = \sqrt{\sum_{k=1}^{m}(w_k(netV_i) - w_k(netV_j))^2} \tag{7.1}$$

In addition, cosine vector similarity has been widely used to measure the similarity between two vectors. An alternative distance metric is to use the complement of vector similarity.

$$cor\_dist(i, j) = 1 - \frac{netV_i \cdot netV_j}{|netV_i||netV_j|} = 1 - \frac{\sum_{k=1}^{m} w_k(netV_i) \cdot w_k(netV_j)}{\sqrt{\sum_{k=1}^{m}[w_k(netV_i)]^2 \cdot \sum_{k=1}^{m}[w_k(netV_j)]^2}} \tag{7.2}$$

Essentially, if we view each vector as an arrow in high-dimension space, the formula above gives the sine of the angle between the arrows of $netV_i$ and $netV_j$.

7.2.2 Generic Clustering Methods

Given a distance metric, we apply generic clustering algorithms to group hosts. We define the radius of cluster i as the maximum distance between the monitor (center node) and any host in cluster i, denoted as C_i. Our clustering seeks to minimize the maximum radius of all clusters, or to minimize the sum of distances between every host and its monitor. The former aims to optimize the worst case prediction, while the latter tries to optimize the average prediction.

Clustering to optimize the worst case (the maximal radius)

The goal is to place a given number of monitors so that the maximum radius is minimized (denoted as *limit_num_minRmax* clustering). This problem is known as the minimum K-center problem [51]. We use the algorithm in [54] to achieve an approximation within a factor of 2 in $O(NK)$ time, where N is the number of end hosts, and K is the number of clusters. More formally, find the minimum K, such that there is a set $N' \subset N$ with $|N'| = K$ and $\forall\ h \in N$, $\exists\ c_h \in N'$ such that distance(h, c_h) $\leq d$.

Clustering to optimize the average case (the average radius)

We formulate the clustering problem that minimizes the sum of intra-cluster distance as follows. Given N points, we select K of them to be centers, and assign each point j to the closest center. If point j is assigned to center i, we incur a cost c_{ij}, where c_{ij} is the correlation distance between point i and j. The goal is to select K centers so as to minimize the sum of assignment costs (denoted as *limit_num_minDistSum* clustering). This is an NP-hard minimum K-median problem. We use a 4-approximation algorithm with running time of $O(N^3)$ [22].

7.2.3 Distance Estimation

We need to consider congestion estimation for accurate latency estimation. Congestion is not common - our analysis of one-week NLANR 105 million ping measurements shows that only 0.96% of probes were congested (see the definition of congestion in Chapter 7.3.2). Of these, about 20% happen at the last mile. We infer that a node has last mile congestion when the majority ($> 90\%$) of its measurements are simultaneously congested. The rest of the congestion exhibits strong spatial correlation, which suggests that a few congested links dominate the end-to-end congestion. Currently, we examine congestion estimation without any knowledge of network topology, and apply a conservative heuristic to Iso-bar, which estimates the distance between host i and j as follows.

- i and j belong to the same cluster with monitor m. If either $path(m, i)$ or $path(m, j)$ is congested, report $path(i, j)$ as congested. Otherwise, $predicted\_dist(i, j) = (dist(m, i) + dist(m, j))/\ 2$.

- i belongs to cluster of monitor m_i and j belongs to the cluster of m_j. If one of $path(m_i, i)$, $path(m_i, m_j)$, and $path(m_j, j)$ is congested, report $path(i, j)$ as congested. Otherwise $predicted\_dist(i, j) = dist(m_i, m_j)$. Note that $path(m_i, i)$ and $path(m_j, j)$ are used for monitoring of last mile congestion, and they are not included in the latency estimation of normal cases.

To reduce false positive congestion alerts, each monitor checks whether it has last mile congestion. If so, the monitor will not use its measurements for congestion inference.

Evaluation based on real Internet measurements show that with only 15 monitors for 106 widely dispersed sites, Iso-bar can capture 78% of congestion with a 32% false positive ratio (Chapter 7.4). This suggests that the monitoring paths used by Iso-bar encompass most of the congested links. However, such congestion estimation accuracy is by no means satisfactory. In the next Chapter, we will discuss how to have good loss rate estimation, though at a bit more expensive measurement cost ($O(N \log N)$ amount of measurements instead of $O(K^2 + N)$ here).

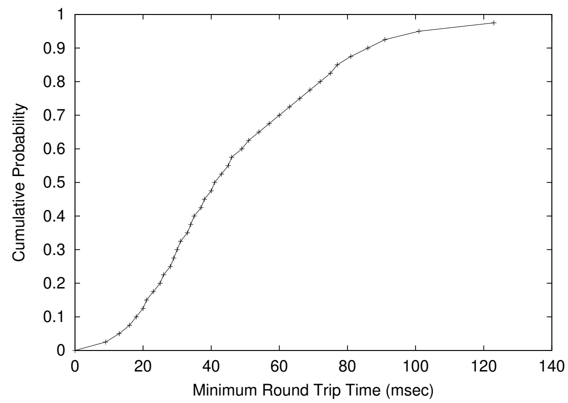


Figure 7.2: CDF of 06/25/01 daily minimum RTT on between each pair of NLANR AMP hosts

Note that we need many more inter-cluster estimates than intra-cluster estimates. For example, given N hosts evenly divided into K clusters, and considering all pairs of hosts for distance estimation, the ratio of inter-cluster estimates to intra-cluster estimates will be approximately $K-1$.

7.2.4 Measurement Traffic

The measurement traffic of Iso-bar is negligible compared with other normal traffic in a P2P system. Given K clusters, N hosts, and measurement frequency f, the total number of measurement packets is $f \times (K^2 + N)$. FastTrack, a very popular P2P system, has 2 million unique IP addresses per day, and most of them have less than 15 minutes on-time. Its total daily traffic is 1.78TB [118]. Assume Iso-bar has 1000 clusters for such a system, and uses 32-byte UDP packet to measure RTT every minute, the total measurement traffic is $(2M + 1K^2) \times 15 \times (32B \times 2) = 2.9$GB, around 0.16% of total normal traffic.

7.3 Evaluation Methodology

7.3.1 Internet Measurement Data

We evaluate the performance of different schemes for distance estimation using the real Internet measurement data from the National Laboratory of Applied Network Research (NLANR) Active Measurement Project (AMP) data [85]. The RTT was measured by ICMP packets in the NLANR traces. There are 119 sites participating in the NLANR AMP project, each of which uses a dedicated machine to measure the RTT between itself and the other participating sites by sending ICMP packets. Measurements are made every minute, resulting in a total of 1440 measurements for each day. We collected one month of raw data (6/25/01 to 7/24/01), and one day of more recent data (12/6/01). After filtering out sites with incomplete measurements, we have 106 hosts (i.e., $N = 106$ in our experiments) and a total of 15 million measurements each day. Figure 7.2 shows the cumulative distribution function (CDF) of minimum RTT between every pair of hosts.

To have more accurate latency estimation through ping measurement, we use a sliding window of 10 samples, and choose the minimum RTT value in the sliding window as the *latency sample*. If all the 10 ping measurements are lost [1], we denote the latency sample as "loss". Based on one day's trace, we compute the geometric mean [2] of all latency samples between every pair of sites i and j as $net\_dist(i, j)$, and use them for clustering or coordinate calculation in GNP. We refer to this day as the *birth date*. We then use the raw measurements of a future day to evaluate the estimation accuracy from each technique. The future day is referred as the *estimation date*.

7.3.2 Estimation Accuracy Metric

At each window slot, we define the measurement to be congested if 1) the latency sample is loss; or 2) the latency sample $>$ ($geometric\ mean \times geometric\ standard\ deviation$). The *geometric mean* and *geometric standard deviation* are calculated based on the measurements of birth date. With the techniques in Chapter 7.2.3, we examine the amount of real congestion captured and the false positive ratio, i.e., $\frac{number\ of\ false\ positives}{total\ congested\ measurements\ reported}$.

For uncongested latency samples, the following relative prediction error defined in [136] is applied as a measure for estimation accuracy:

$$relative\ error = Avg[|log_2(\frac{predicted\ distance}{measured\ distance})|]$$

where Avg refers to the average value computed over each of the measurement events (i.e., 1440 in the NLANR daily traces except the congested samples). Both the predicted distance and measured distance are latency samples. The relative error reflects how much the estimation deviates from the target value.

Given different estimation techniques, the predicted distance could be either *static* or *dynamic* (see Chapter 7.3.3).

GNP is not scalable enough for online update of the coordinates. Therefore we use each site's coordinates obtained from the birth date for distance estimation. For any pair of hosts i and j, the same estimated value is used consistently as predicted distance when we calculate the daily relative error.

For Iso-bar, we assume the monitors actively measuring the distance among themselves, as well as to the other hosts in its cluster. We use these measurements to predict the distance between clients as in Chapter 7.2.3.

7.3.3 Analysis of Estimation Accuracy

Static analysis

For static analysis, we use the measurement data of the same day both for offline setup (clustering or coordinates computation) and for online estimation. That is, the birth date and the estimation date are the same. This will help give a sense of the absolute accuracy of each estimation technique without temporal variation.

[1] Each ping sends at most 4 ICMP packets over 4 seconds before it reports a loss.

[2] We use the geometric mean because Internet distance measurements obey a heavy-tailed distribution, as do our datasets.

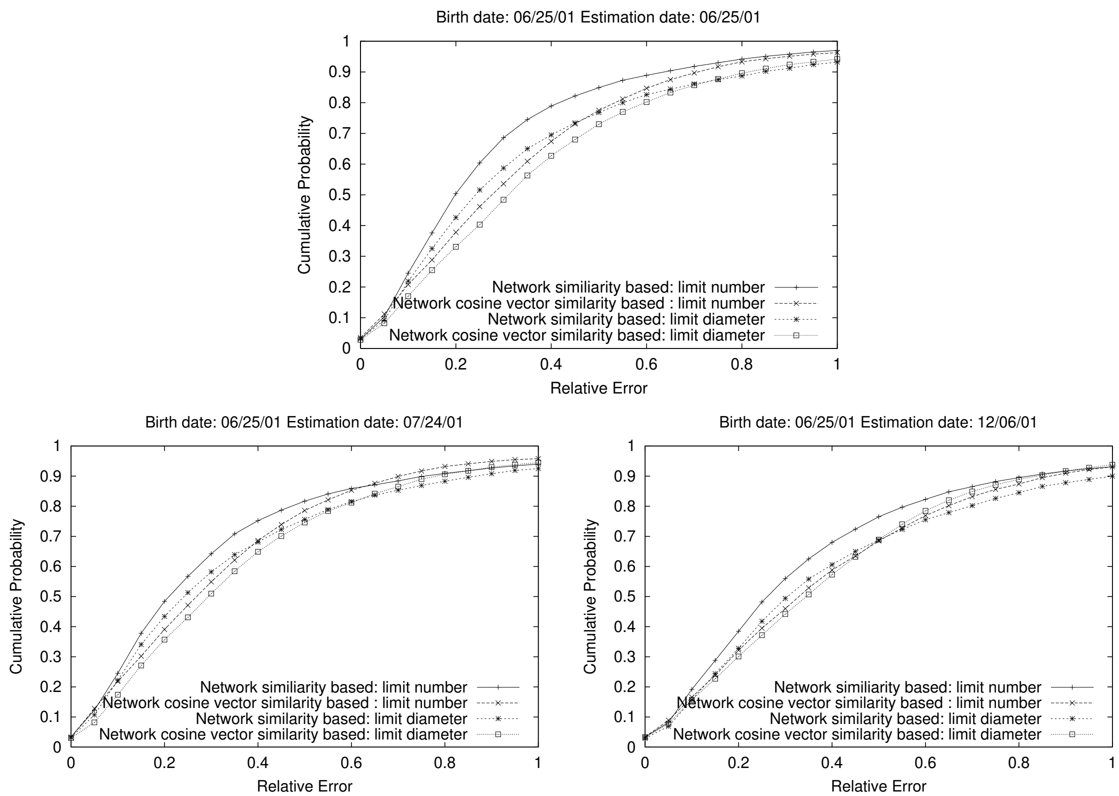


Figure 7.3: Evaluation of generic clustering methods with static analysis (top) and stability analysis (bottom).

Stability analysis

It is even more interesting to examine how well estimations derived from a single day's data perform over multiple time intervals. We evaluate the stability of various distance estimation schemes over a six-month period. The birth date is 06/25/01, and the estimation dates are 6/25/01 - 7/01/01, 7/8/01, 7/24/01, and 12/06/01, respectively.

7.4 Evaluation Results

In this section, we first describe different Internet distance estimation techniques to be used for comparison, and then evaluate the sensitivity of Iso-bar with varying number of landmarks. Finally, we present the accuracy and stability of various estimation schemes.

7.4.1 Internet Distance Estimation Techniques Evaluated

We compare four distance estimation schemes:

- Omniscient approach
- GNP

- Clustering with network distance vector (Iso-bar)
  - Using Euclidean distance (Net_sim)
  - Using vector similarity (Net_vsim)
- Clustering with network proximity

The omniscient approach makes distance estimation based on complete knowledge of every pair-wise distance on the birth date. It estimates the distance between any pair of hosts as the geometric mean of the actual latency samples on the birth date. For a fair comparison, We use 15 landmarks (in 7 dimensions as configured in [83]) for GNP and 15 clusters for clustering approaches. For GNP, we ran five experiments, choosing random landmarks each time, and present the one with the best results.

For Iso-bar approaches, clustering based on Net_sim slightly outperforms Net_vsim, as shown in Figure 7.3). So we only use Net_sim, and refer it as Iso-bar hereafter.

7.4.2 Iso-bar Sensitivity to Different Number of Landmarks

In this section, we examine how dependent clustering performance is on the number of landmarks. Our approach is to compare the performance of Iso-bar ($limit\_num\_minDistSum$) when the number of randomly-chosen landmarks M varies from 6 to 106. Simulation shows they differ by less than 5% in terms of prediction accuracy (Figure 7.4). This suggests that only a small number of landmarks is sufficient. Unlike GNP, the landmarks are used for initial clustering only (see Chapter 7.2.1). We use 15 landmarks for the remaining experiments.

7.4.3 Results of Estimation Accuracy and Stability

We first present results on latency estimation when there is no congestion, then show the congestion estimation performance of clustering-based dynamic monitoring systems.

Normal latency estimation

For uncongested latency samples, Figure 7.5 shows the CDF of the relative errors for both static and stability analysis. The stability results for daily, weekly and monthly intervals are very close to the static results. Figure 7.6 shows the amount of relative errors that are below the 90th percentile, and how they change over time. In this metric, a lower value represents a higher level of accuracy. Most methods are relatively stable, except the omniscient scheme and GNP. Both of them have scalability problems for continuous update of the distance estimation. Thus unlike the clustering-based approaches, we do not update their distance estimation.

We make the following observations. First, estimation based on the omniscient approach gives the highest accuracy of all the estimation dates. We believe this represents an underlying stability in the current Internet. Whether the stability is applicable to other parts of Internet in general needs further verification. However, as a static estimation

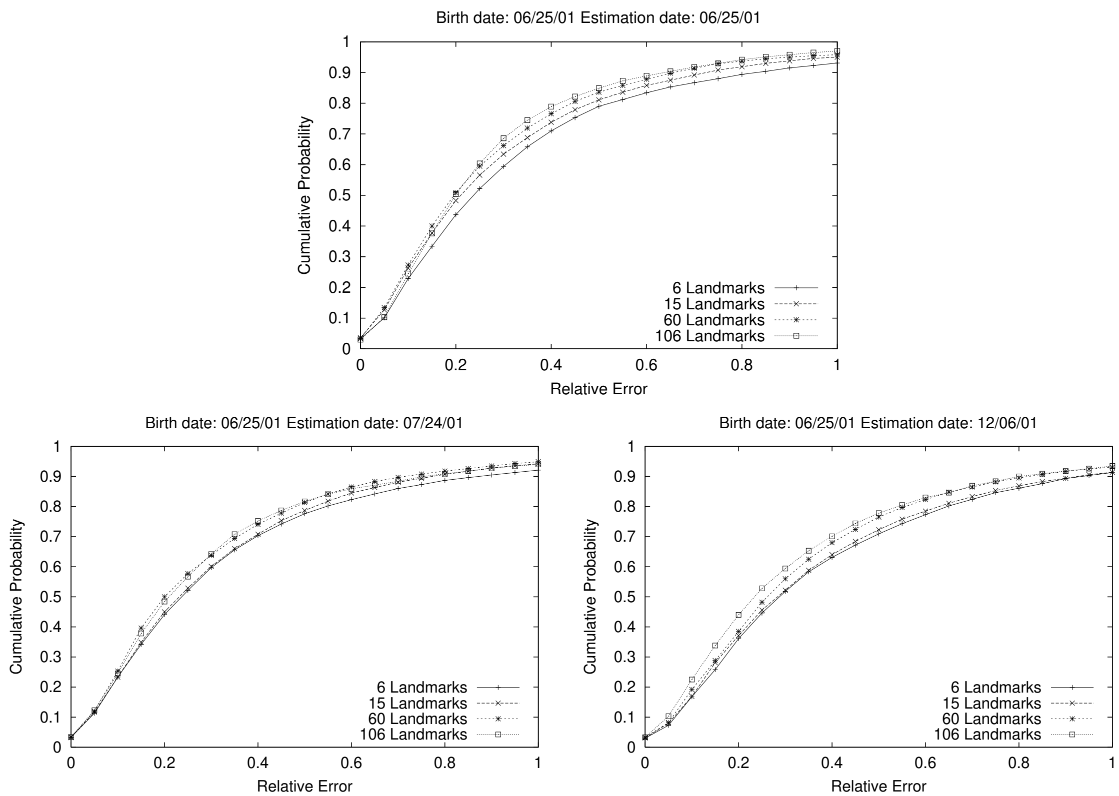


Figure 7.4: Sensitivity of Internet Iso-bar to various number of landmarks: static (top) and stability analysis (bottom).

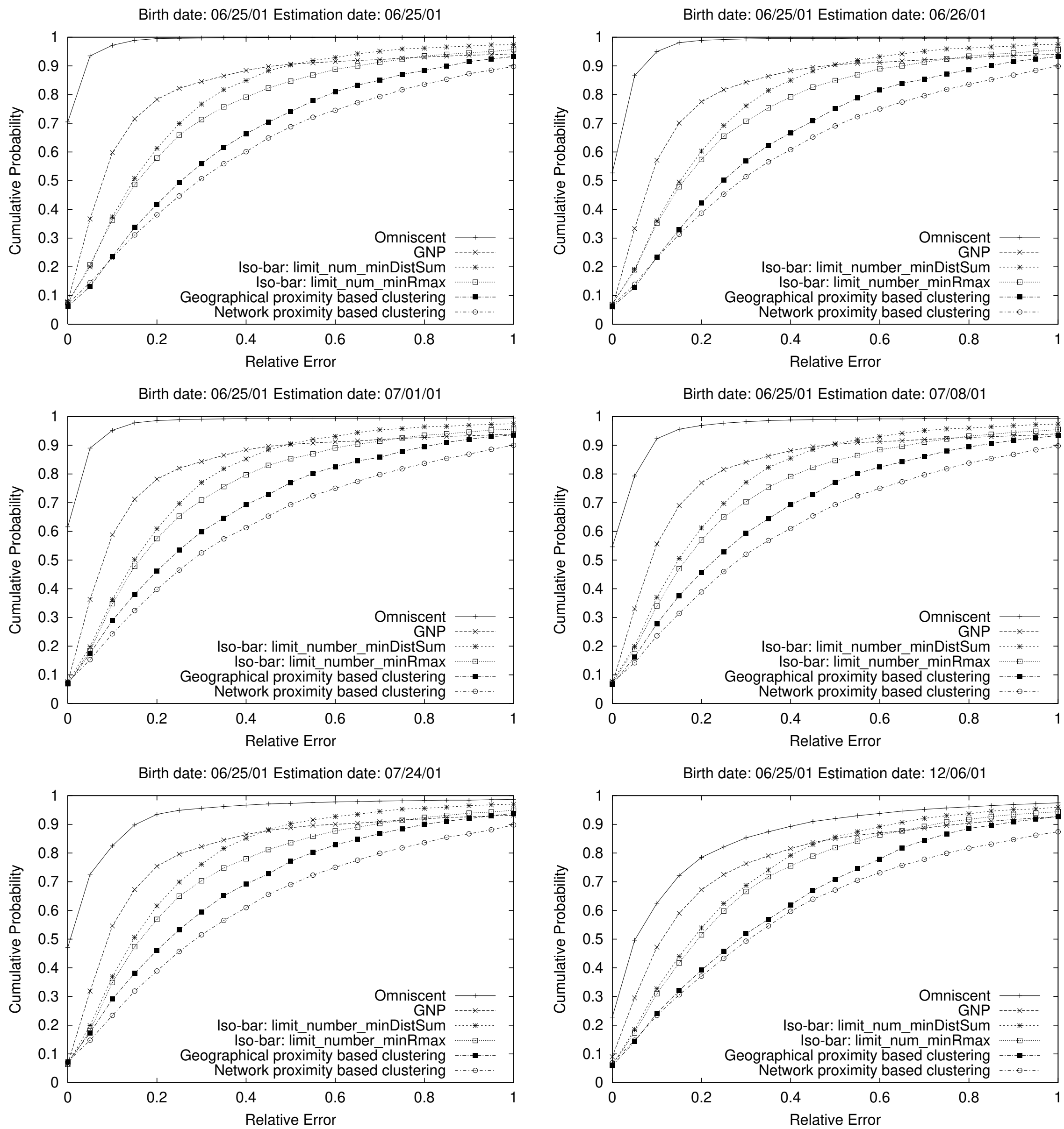


Figure 7.5: Cumulative Distribution Function (CDF) of relative prediction errors for both static analysis and stability analysis (6-monthly interval)

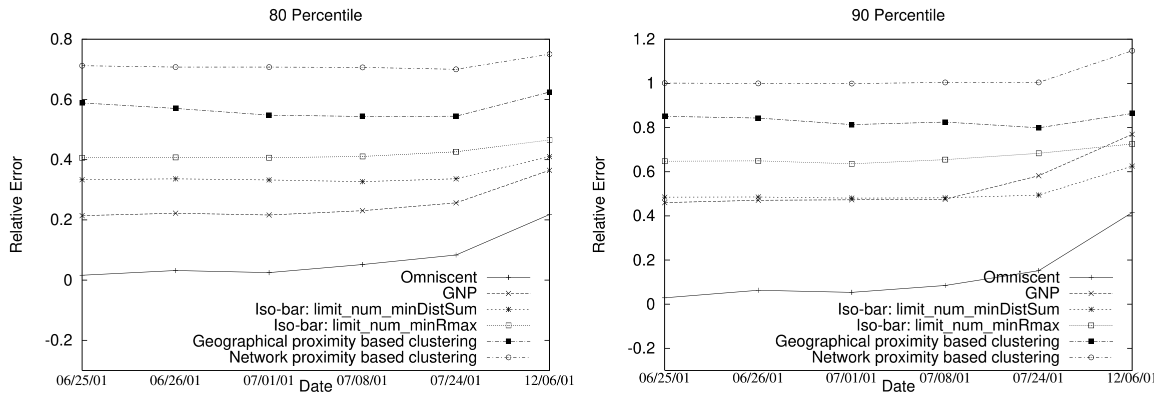


Figure 7.6: 80 percentile (left) and 90 percentile (right) of the relative errors for various estimation schemes under six different time intervals

scheme, the omniscient approach can not report any congestion information. Furthermore, it requires the full $N \times N$ network distance matrix, and is thus not scalable. Note that the omniscient approach does not achieve perfect accuracy when it is used to estimate distance on the actual birth date, because the distance estimate between any pair of hosts is the geometric mean of latency samples obtained during the whole day. The error reflects the amount of fluctuation in RTT within a day.

Second, GNP and Iso-bar (*limit_num_minDistSum*) have similar performance, only a little bit worse than the omniscient scheme. This implies that GNP provides a very accurate estimate when the underlying data set is stable. However, GNP requires all hosts to constantly measure distance to the landmarks for online monitoring, which is impractical. Thus it cannot report timely congestion information.

Third, the network distance similarity based clustering performs much better than the network proximity based clustering, which is used in IDMaps [51] and Network Distance Maps [127].

Finally, it is interesting to see that geographical distance proximity based clustering performs better than network distance proximity based clustering. This may be because most sites in our experimental data set are educational and research institutes, and they are connected by the same backbone: Internet2. The correlation between geographical distance and network distance need to be further verified. Previous work has indicated that estimation techniques based on geographic distance in general results in poor accuracy [83].

Congestion/failure estimation

During the week of 06/25/01 to 07/01/01, there are averagely 148K congested measurements per day, and Iso-bar (*limit_num_minDistSum*) captures 78% of congestion with a 32% false positive ratio. Clustering with network proximity captures 75% of congestion with a 44% false positive ratio. Note that given 15 monitors, the measurement overhead is (15×14 + 106 - 15), which is less than 3% of RON measurement cost (106×105).

7.5 Summary

In summary, in this chapter, we propose a clustering-based overlay latency monitoring service, the *Internet Iso-bar*. In contrast to traditional network or geographical proximity based clustering, Iso-bar groups hosts based on the similarity of their perceived distance to a small number of landmarks. Evaluation based on real Internet measurements show that Iso-bar has small online measurement overhead, and high latency estimation accuracy and stability, comparable to the best known work, GNP. Unlike static estimation systems like GNP, Iso-bar can also detect the majority of congestion/failures in real time with certain false positive.

The congestion/failure detection accuracy is reasonable for non-mission-critical applications, such as P2P file sharing, but probably not acceptable for applications that demand high availability, like Virtual Private Network (VPN). In the next chapter, we will introduce a tomography-based monitoring service which provides accurate congestion/failure estimation with a bit more measurement overhead than Iso-bar ($O(N \log N)$ amount of measurements instead of $O(K^2 + N)$, but still much less than $O(N^2)$).

Chapter 8

TOM: A Tomography-based Overlay Monitoring System

8.1 Introduction

In the previous chapter, we presented a scalable and accurate overlay latency monitoring system, Internet Iso-bar. In this chapter, we will focus on loss rate estimation for congestion and failure monitoring. Consider an overlay network of n end hosts; we define a path to be a routing path between a pair of end hosts, and a link to be an IP link between routers. A path is a concatenation of links. There are $O(n^2)$ paths among the n end hosts, and we wish to select a minimal subset of paths to monitor so that the loss rates and latencies of all other paths can be inferred. The loss rates are used to estimate the congestion/failures on the overlay paths.

To this end, we propose a tomography-based overlay network monitoring system in which we selectively monitor a *basis set* of k paths (typically $k \ll n^2$). Any end-to-end path can be written as a unique linear combination of paths in the basis set. Consequently, by monitoring loss rates for the paths in the basis set, we infer loss rates for all end-to-end paths. This can also be extended to other additive metrics, such as latency. The end-to-end path loss rates can be computed even when the paths contain *unidentifiable links* for which loss rates cannot be computed. We provide an intuitive picture of this characterization process in terms of *virtual links*.

Although congestion outbursts within seconds are hard to detect and bypass, the delay in Internet inter-domain path failovers averages over three minutes [69]. Our loss rate estimation will filter out measurement noise with smoothing techniques, such as exponentially-weighted moving average (EWMA), and detect these path failovers quickly to have applications circumvent them.

Network tomography has been extensively studied ([36] provides a good survey). Most existing systems assume that limited measurement information is available (often in a multicast tree-like structure), and they try to infer the characteristics of the links [1, 2, 18, 92] or shared congestion [114] in the middle of the network. In many cases, these inferences are restricted due to limited measurement and the irregularity of Internet topologies. They are facing a fundamentally under-constrained system (as the unidentifiable links in Chapter 8.2). Thus the inferences are only of statistical meanings, and no hard guarantee.

In contrast, we do not care about the characteristics of *individual* links. Furthermore, we do not have any restriction on the paths to measure. Our goal is to selectively measure a small subset of paths so that we can infer the loss rates of all other paths.

Our key observation is that k grows relatively slowly as a function of n. The dimension k is bounded by the number of links in the subgraph induced by the routing paths. In an Internet-like topology with a power-law degree distribution, there are $O(N)$ links, where N is the total number of end hosts in the network. This is because a small number of nodes have high degree and the links between them are heavily used [47]. Consequently, if $n = O(N)$, then $k < O(n)$. However, even when $n \ll N$, the moderately hierarchical structure of the network causes many routing paths to overlap [126], so that the number of links in the routing path subgraph grows much slower than $O(n^2)$. Our extensive study of both synthetic and real Internet topologies suggests that for a randomly selected subset of n end hosts, k grows like $O(n \log n)$ when n is sufficiently large (say 100).

In addition, we design efficient algorithms to adapt to topology changes (such as end host join/leave and routing changes), distributes measurement load evenly across the end hosts, and handle topology measurement errors.

Both simulation and PlanetLab [102] experiments results show that we achieve high accuracy when estimating path loss rates with $O(n \log n)$ measurements. For the PlanetLab experiments, the average absolute error of loss rate estimation is only 0.0027, and the average error factor is 1.1, although about 10% of the paths have no or incomplete routing information. The average setup (monitoring path selection) time is 0.75 second, and the online update of the loss rates for all 2550 paths takes only 0.16 second. In addition, we adapt to topology changes within seconds without sacrificing accuracy. The measurement load balancing reduces the load variation and the maximum vs. mean load ratio significantly, by up to a factor of 7.3.

The rest of the chapter is organized as follows. We describe our model and basic theory in Chapter 8.2 and present the basic algorithms in Chapter 8.3. We extend the algorithms to adapt to topology changes in Chapter 8.4, and to handle overloading and topology measurement errors in Chapter 8.5. The methodology and results of simulation are described in Chapter 8.6, and those of real Internet experiments are presented in Chapter 8.7. Finally, we discuss the generalization of our framework in Chapter 8.8 and conclude in Chapter 8.9.

8.2 The Algebraic Model and Scalability Analysis

In this section, we develop the model for tomography-based overlay monitoring.

Given n end hosts to be monitored, we assume that they belong to an overlay network (such as a virtual private network), or that they cooperate to share the monitoring services. Thus, we can measure the routing topology and loss rate of any path. The end hosts are under the control of a central authority (e.g., an overlay network operation center (ONOC)) to measure the topology and loss rates of paths, though in the future we plan to investigate techniques to distribute the work of the central authority.

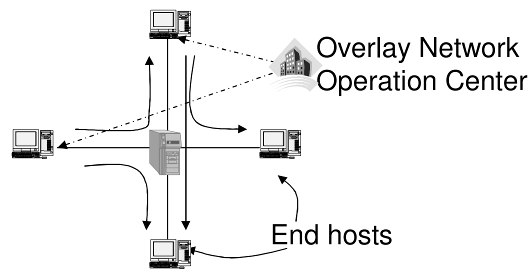


Figure 8.1: Architecture of a tomography-based overlay network monitoring system

For simplicity, we mostly assume symmetric routing and un-directional links in the chapter. But our techniques work without changes for asymmetric routing, as used in the Internet experiments. Figure 8.1 shows an example where there are four end hosts on the overlay network. There are six paths and four links. The end hosts measure the topology and report to the ONOC, which selects four paths and instruments two of the end hosts to measure the loss rates of the four paths. The end hosts periodically report the loss rates measured to the ONOC. Then the ONOC infers the loss rates of every link, and consequently the loss rates of the other two paths. Applications can query the ONOC for the loss rate of any path, or they can set up triggers to receive alerts when the loss rates of paths of interest exceed a certain threshold.

The path loss rates can be measured by either passive observation of normal traffic to estimate packet drop rate [92] or active measurement. The measurements of selected paths do not have to be taken at exactly the same time because Zhang, *et al.* report that the loss rate remains operationally stable in the time scale of an hour [136]. The network topology can be measured via traceroute or other advanced tools [56, 35]. We discuss topology changes in Chapter 8.4.

8.2.1 Theory and Notations

Suppose an overlay network spans s IP links. We represent a path by a column vector $v \in \{0,1\}^s$, where the jth entry v_j is one if link j is part of the path, and zero otherwise. Suppose link j drops packets with probability l_j; then the probability p of packet loss on the path represented by v is given by

$$1 - p = \prod_{j=1}^{s} (1 - l_j)^{v_j} \tag{8.1}$$

By taking logarithms on both sides of (8.1), we have

$$\log(1 - p) = \sum_{j=1}^{s} v_j \log(1 - l_j) \tag{8.2}$$

If we define a column vector $x \in \mathbb{R}^s$ with elements $x_j := \log(1 - l_j)$, and write v^T for the row vector which is the transpose of v, we can rewrite (8.2) in the following dot product form:

$$\log(1 - p) = \sum_{j=1}^{s} v_j x_j = v^T x \tag{8.3}$$

Symbols	Meanings
M	total number of nodes
N	number of end hosts
n	number of end hosts on the overlay
$r = O(n^2)$	number of end-to-end paths
s	# of IP links that the overlay spans on
t	number of identifiable links
$G \in \{0,1\}^{r \times s}$	original path matrix
$\bar{G} \in \{0,1\}^{k \times s}$	reduced path matrix
$k \leq s$	rank of G
l_i	loss rate on ith link
p_i	loss rate on ith measurement path
x_i	$\log(1 - l_i)$
b_i	$\log(1 - p_i)$
v	vector in $\{0,1\}^s$ (represents path)
p	loss rate along a path
$\mathcal{N}(G)$	null space of G
$\mathcal{R}(G^T)$	row(path) space of G (== range(G^T))

Table 8.1: Table of notations

Considering all $r = O(n^2)$ paths in the overlay network, there are r linear equations of the form (8.3). Putting them together, we form a rectangular matrix $G \in \{0,1\}^{r \times s}$ to represent these paths. Each row of G represents a path in the network: $G_{ij} = 1$ when path i contains link j, and $G_{ij} = 0$ otherwise. Let p_i be the probability of packet loss during transmission on the ith path, and let $b \in \mathbb{R}^r$ be a column vector with elements $b_i := \log(1 - p_i)$. Then we write the system of equations relating the link losses to path losses as

$$Gx = b \tag{8.4}$$

In general, the measurement matrix G may be rank deficient: i.e., $k = \text{rank}(G)$ and $k < s$. If G is rank deficient, we will be unable to determine the loss rate of some links from (8.4). We call these links *unidentifiable* as in [18].

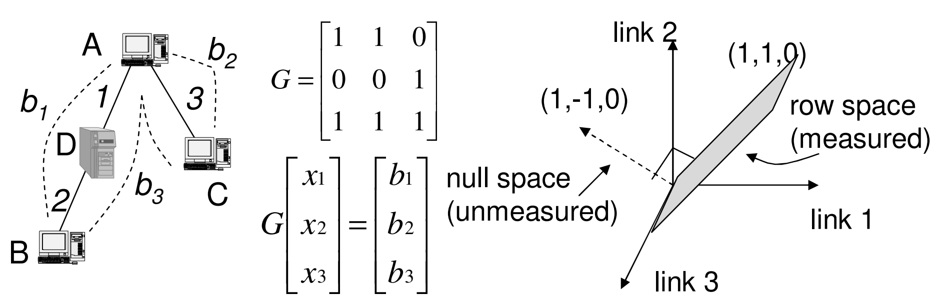


Figure 8.2: Sample overlay network.

We illustrate how rank deficiency can occur in Figure 8.2. There are three end hosts (A, B, and C) on the overlay, three links (1, 2 and 3) and three paths between the end hosts. Because links 1 and 2 always appear together, their individual loss rates cannot be computed from the measurements. For example, suppose that $x_1 + x_2 = b_1 = -0.06$ and $x_3 = b_2 = -0.01$. We know that $x_1 = -0.03 + \alpha$ and $x_2 = -0.03 - \alpha$ for some α, but the value of α cannot be determined from the end-to-end measurements. The set of vectors $\alpha \begin{bmatrix} 1 & -1 & 0 \end{bmatrix}^T$ which are not defined by (8.4) can be added to x without affecting b. This set of vectors is the *null space* of G.

To separate the identifiable and unidentifiable components of x, we write x as $x = x_G + x_N$, where $x_G \in \mathcal{R}(G^T)$ is in the *row space* of G and $x_N \in \mathcal{N}(G)$ is in the orthogonal *null space* of G (i.e. $Gx_N = 0$). The vector x_G contains all the information we can know from (8.4) and the path measurements. For instance, we can determine $x_1 + x_2$ in Figure 8.2, but not $x_1 - x_2$. Intuitively, links 1 and 2 together form a single *virtual link* with an identifiable loss rate $x_1 + x_2$. All end-to-end paths can be written in terms of such *virtual links*, as we describe in more details in Chapter 8.2.3. So x_G involves all the links, while x_N only involves unidentifiable links. The decomposition of x for the sample overlay network is shown below.

$$x_G = \frac{(x_1 + x_2)}{2} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} + x_3 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} b_1/2 \\ b_1/2 \\ b_2 \end{bmatrix} \tag{8.5}$$

$$x_N = \frac{(x_1 - x_2)}{2} \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} \tag{8.6}$$

Because x_G lies in the k-dimensional space $\mathcal{R}(G^T)$, only k independent equations of the r equations in (8.4) are needed to uniquely identify x_G. By measuring k independent paths, we can compute x_G. Since $b = Gx = Gx_G + Gx_N = Gx_G$, we can compute all elements of b from x_G, and thus obtain the loss rate of all other paths. For example, in Figure 8.2, we only need to measure b_1 and b_2 to compute x_G, from which we can calculate b_3. Detailed algorithms are described in Chapter 8.3.

8.2.2 Scalability Analysis

In this section, we will examine asymptotically how big k is in terms of n.

Theorem 1 *Given a power-law degree network topology of M nodes, the frequency f_d of nodes with outdegree d is proportional to d^c, where c is the outdegree exponent constant (i.e., $f_d \propto d^c$). With $d \geq 1$ and $c < -2$ (as found in [47]), the number of end hosts N is at least $M/2$.*

Proof: Given that the power-law distribution topology has out-degree exponent: the frequency f_d of an outdegree d is proportional to the outdegree to the power of a constant, i.e., $f_d = Nd^c$, where N is the proportion constant. Assume that end hosts have degree 1, then the number of end hosts is N.

If $c < -1$, then

$$
\begin{aligned}
M &= N \sum_{d=1}^{M-1} d^c && (8.7)\\
&\leq N\left(1 + \int_1^{M-1} x^c \, dx\right) && (8.8)\\
&\leq N\left(1 + \int_1^{\infty} x^c \, dx\right) && (8.9)\\
&= N\left(1 - \frac{1}{1+c}\right) && (8.10)\\
&= N\frac{c}{1+c} && (8.11)
\end{aligned}
$$

Therefore, the fraction $\frac{N}{M}$ is at least $\frac{1+c}{c} = 1 + \frac{1}{c}$. If $c \leq -2$ then $\frac{N}{M} \geq \frac{1}{2}$. ■

This theorem also follows the intuition that the number of end hosts should be more than the number of routers in the Internet.

Meanwhile, Faloutsos, *et al.* prove that such a topology has only $O(M)$ links (Lemma 2 in [47]). Combining the two facts, given N end hosts, there are at most $O(N)$ links in the topology. Thus, if the majority of the end hosts are on the overlay network ($n = O(N)$), the dimension of $\mathcal{R}(G^T)$ is $O(n)$.

What about if only a small portion of the end hosts are on the overlay? Tangmunarunkit, *et al.* found that the power-law degree Internet topology has moderate hierarchy due to the heavy-tailed degree distribution [126]. Because G is an r by s matrix, k is bounded by the number of links s. If it is strict hierarchy like a tree, $s = O(n)$, thus $k = O(n)$. But if there is no hierarchy at all (e.g., clique), $k = O(n^2)$ because all the $O(n^2)$ paths are linearly independent. Moderate hierarchy should fall in between. We found that for reasonably large n (e.g, 100), $k = O(n \log n)$.

We run linear regression tests of our hypothesis on both synthetic and real topologies. We experiment with three types of BRITE router-level topologies: Barabasi-Albert, Waxman and hierarchical models, as well as a real router level topology with 284,805 nodes [56]. For hierarchical topologies, BRITE first generates an autonomous system (AS) level topology with a Barabasi-Albert model or a Waxman model. Then for each AS, BRITE generates the router-level topologies with another Barabasi-Albert model or Waxman model. So there are four types of possible topologies, and they all have the same trend for regression. We just show the results of AS-level Barabasi-Albert model and router-level Waxman model as the representative. For simplicity, we only compute an upper bound on k for the real topology. We prune nodes and links that are not on the overlay paths, and collapse linear link chains with no branches into one link, then we count the number of links as the upper bound.

We randomly select end hosts, which has the least degree, to form an overlay network. We fit linear regression of k on $O(n)$, $O(n \log n)$, $O(n^{1.25})$, $O(n^{1.5})$, and $O(n^{1.75})$. As shown in Figure 8.3, results for each type of topology are averaged over three runs with different topologies for synthetic ones and with different random sets of end hosts for the real one. We find that for Barabasi-Albert, Waxman and real topologies, $O(n)$ regression has

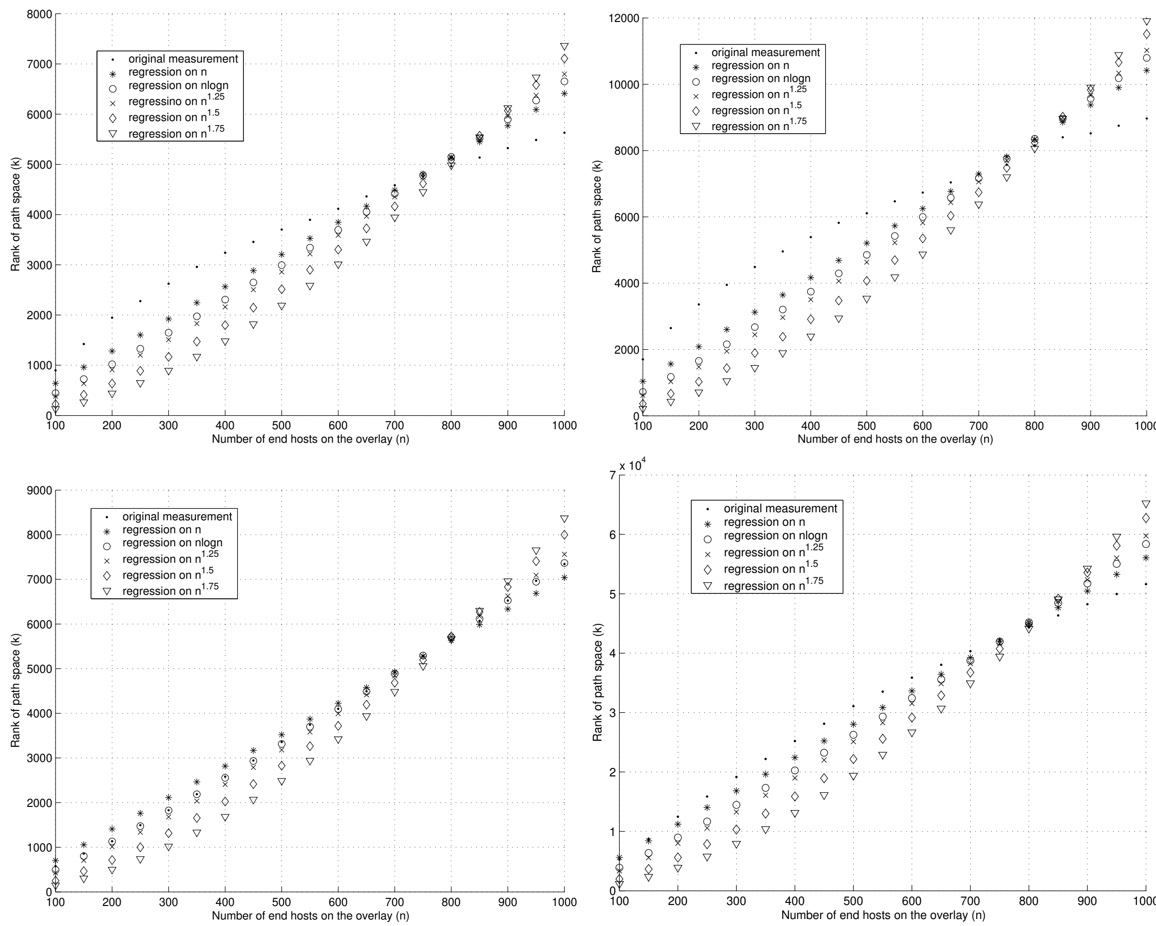


Figure 8.3: Regression of k in various functions of n under different router-level topologies. Top: Barabasi-Albert model of 20K nodes (left), and Waxman model of 10K nodes (right). Bottom: hierarchical model of 20K nodes (left) and a real topology of 284K routers (right).

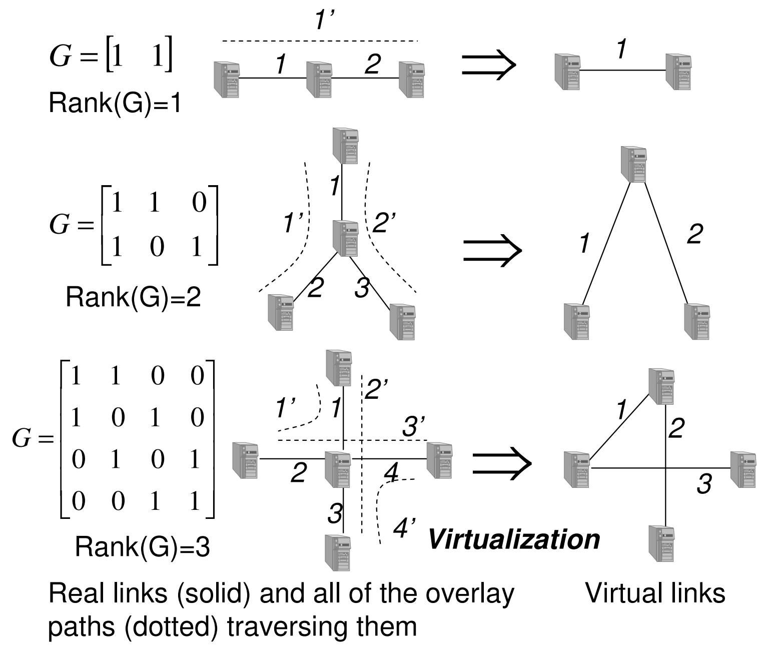


Figure 8.4: Sample parts of IP network and overlay paths.

the least residual errors, while for a hierarchical model, $O(n \log n)$ fits best. Conservatively speaking, we have $k = O(n \log n)$.

8.2.3 Intuition through Virtual Links

In Chapter 8.2.1, we explain in algebraic terms how to compute all end-to-end path loss rates from only k path measurements. Our actual computations are based completely on this algebraic picture; however, these formulas may not seem intuitive. We now describe a more intuitive picture using the notion of *virtual links*. The key idea is that although the loss rates of some individual links are uncomputable (unidentifiable links), each of them is covered by some path segment whose loss rate is computable, and the loss rates of these path segments are sufficient to compute the path loss rates in which we are interested.

We choose a minimal set of such path segments that can fully describe all end-to-end paths, and refer to them as *virtual links*. If a link is identifiable, the link itself is a virtual link.

Figure 8.4 illustrates some examples. In the top figure, the virtual link is a concatenation of two sequential physical links as we discussed before. In the middle figure, there are three links, but only two paths traverse these links. Thus, $\text{rank}(G) = 2$ and none of the links are identifiable. In the bottom figure, there are four links, and a total of four paths traversing them. But the four paths are linearly dependent, so $\text{rank}(G) = 3$, and none of the link loss rate are computable. We can use any three out of the four paths as virtual links, and the other one can be linearly represented by the virtual links. For example, path $4'$ can be described as virtual links 2+3-1.

Since the dimension of $\mathcal{R}(G^T)$ is k, the minimum number of virtual links which can fully describe $\mathcal{R}(G^T)$ is also k. x_G is a linear combination of the vectors representing the virtual links. Since virtual links are identifiable, x_G is also computable. From x_G, we can compute the loss rates of all end-to-end paths as we can do with virtual links.

8.3 Basic Algorithms

In this section, we describe basic static algorithms. We will present the dynamic algorithms for topology changes in the next section.

8.3.1 Selecting Measurement Paths

To characterize all $O(n^2)$ end-to-end paths, we monitor k linearly independent end-to-end paths and form a reduced system

$$\bar{G}x_G = \bar{b} \tag{8.12}$$

where $\bar{G} \in \{0,1\}^{k \times s}$ and $\bar{b} \in \mathbb{R}^k$ consist of k rows of G and b, respectively. Linearly independent sets of rows and columns in rank-deficient problems are usually computed using *rank-revealing decompositions* [53]. For a dense r by s matrix with rank k, common rank-revealing decompositions include Gaussian elimination with complete pivoting (as used in [119]), QR with column pivoting, and the singular value decomposition (SVD). The former two cost $O(rks)$, and the SVD costs $O(rs^2)$. Our G matrix is very sparse; that is, there are only a few non-zeros per row. Rank-revealing decompositions for many sparse problems can be computed much more quickly than in the dense case. However, the exact cost depends strongly on the structure of the problem, and efficient computation rank-revealing decompositions of sparse matrices is an open area of research [76], [100].

We select rows using Algorithm 8, which is a variant of the QR procedure [53, p.223]. The procedure incrementally builds a decomposition

$$\bar{G}^T = QR \tag{8.13}$$

where $Q \in \mathbb{R}^{s \times k}$ is a matrix with orthonormal columns and $R \in \mathbb{R}^{k \times k}$ is upper triangular. We do not store Q explicitly; instead, we write Q as $R^{-1}\bar{G}^T$. The idea is the same as the classical Gram-Schmidt algorithm: as each row is inspected, we subtract off any components in the space spanned by the previous rows, so that the remainder is orthogonal to all previous rows. If the remainder is zero, then the row was linearly dependent upon the previous rows; otherwise, we extend the factorization.

In practice, we use a variant of Algorithm 8 which uses optimized routines from the LAPACK library [6] and inspects several rows at a time. The time complexity of processing each vector is dominated by the solution of a triangular system to compute $\hat{R}_{12}$, which costs $O(k^2)$. The total cost of the algorithm is $O(rk^2)$ and the constant in the bound is modest (see the running time results in Chapter 8.6.4 and 8.7.2). The memory cost is roughly $k^2/2$ single-precision floating point numbers for storing the R factor.

When k exceeds 10000 the $O(k^2)$ memory requirement becomes too onerous. We note that dense factorization methods may still be feasible if the number of overlay end-hosts is small or if we relax our original problem statement.

procedure SelectPath(G)
for *every row v in G* **do**
  $\hat{R}_{12} = R^{-T}\bar{G}v^T = Q^T v^T$
  $\hat{R}_{22} = \|v\|^2 - \|\hat{R}_{12}\|^2$
  if $\hat{R}_{22} \neq 0$ **then**
    Mark v as a measurement path
    $\bar{G} = \begin{bmatrix} \bar{G} \\ v \end{bmatrix}$
    $R = \begin{bmatrix} R & \hat{R}_{12} \\ 0 & \hat{R}_{22} \end{bmatrix}$
  end
end

Algorithm 8: Path (Row) Selection Algorithm

8.3.2 Path Loss Calculations

The QR decomposition which we use to select measurement paths is also used to compute a solution to the underdetermined system (8.12). To choose a unique solution x_G to $\bar{G}x_G = \bar{b}$, we impose the additional constraint that $x_G = \bar{G}^T y$. We can then compute

$$\begin{aligned} y &= R^{-1}R^{-T}\bar{b} \\ x_G &= \bar{G}^T y \end{aligned}$$

This is a standard method for finding the minimum norm solution to an underdetermined system (see [53], [42]). Once we have computed x_G, we can compute $b = Gx_G$, and from there infer the loss rates of the remaining paths. The dominant cost in the computation is the solution of two triangular linear systems for y, which only costs $O(k^2)$ and is much smaller than that of the measurement path selection. Thus we can update loss rate estimates online, as verified in Chapters 8.6.4 and 8.7.2.

8.4 Dynamic Algorithms for Topology Changes

During normal operation, new links may appear or disappear, routing paths between end hosts may change, and hosts may enter or exit the overlay network. These changes may cause rows or columns to be added to or removed from G, or entries in G may change. In this section, we design efficient algorithms to incrementally adapt to these changes.

8.4.1 Path Additions and Deletions

The basic building blocks for topology updates are path additions and deletions. We have already handled path additions in Algorithm 8; adding a path v during an update is no different than adding a path v during the initial scan of G. In both cases, we decide whether to add v to $\bar{G}$ and update R.

To delete a path that is not in $\bar{G}$ is trivial; we just remove it from G. But to remove a path from $\bar{G}$ is more complicated. We need to update R; this can be done in $O(k^2)$ time by standard algorithms (see e.g. Algorithm 3.4 in [123, p.338]). In general, we may then need to replace the deleted path with another measurement path. Finding a replacement path, or deciding that no such path is needed, can be done by re-scanning the rows of G as in Algorithm 8; however, this would take time $O(rk^2)$.

procedure DeletePath(v, G, $\bar{G}$, R)
if *deleted path v is measured* **then**
    $j =$ index of v in $\bar{G}$
    $y = \bar{G}^T R^{-1} R^{-T} e_j$
    Remove v from G and $\bar{G}$
    Update R (Algorithm 3.4 in [123, p.338])
    $r = Gy$
    if $\exists$ *i such that $r_i \neq 0$* **then**
        Add the ith path from G to $\bar{G}$ (Algorithm 8, steps 2-7)
    end
end
else Remove v from G

Algorithm 9: Path deletion algorithm

We now describe Algorithm 9 to delete a path v more efficiently. Suppose v corresponds to the ith row in $\bar{G}$ and the jth row in G, we define $\bar{G}' \in \mathbb{R}^{(k-1)\times s}$ as the measurement path matrix after deleting the ith row, and $G' \in \mathbb{R}^{(r-1)\times s}$ as the path matrix after removing the jth row. By deleting v from $\bar{G}$, we reduce the dimension of $\bar{G}$ from k to $k-1$. Intuitively, our algorithm works in the following two steps.

1. Find a vector y that only describes the direction removed by deleting the ith row of $\bar{G}$.

2. Test if the path space of G' is orthogonal to that direction, *i.e.*, find whether there is any path $p \in G'$ that has a non-zero component on that direction. If not, no replacement path is needed. Otherwise, replace v with any of such path p, and update the QR decomposition.

Next, we describe how each step is implemented. To find y which is in the path space of $\bar{G}$ but not of $\bar{G}'$, we solve the linear system $\bar{G}y = e_i$, where e_i is the vector of all zeros except for a one in entry i. This system is similar to the linear system we solved to find x_G, and one solution is $y = \bar{G}^T R^{-1} R^{-T} e_i$.

Once we have computed y, we compute $r = G'y$, where G' is the updated G matrix. Because we chose y to make $\bar{G}'y = 0$, all the elements of r corresponding to selected rows are zero. Paths such that $r_j \neq 0$ are guaranteed to be independent of $\bar{G}'$, since if row j of G could be written as $w^T \bar{G}'$ for some w, then r_j would be $w^T \bar{G}'y = 0$. If all elements of r are zero, then y is a null vector for all of G'; in this case, the dimension k' of the row space of G' is $k-1$, and we do not need to replace the deleted measurement path. Otherwise, we can find any j such that $r_j \neq 0$ and add the jth path to $\bar{G}'$ to replace the deleted path.

Take the overlay network in Figure 8.2 for example, suppose $\bar{G}$ is composed of the paths AB and BC, *i.e.*, $\bar{G} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$. Then we delete path BC, $\bar{G}' = [1 \quad 1 \quad 0]^T$ and $G' = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. Applying Algorithm 9, we have $y = [0 \quad 0 \quad 1]^T$ and $r = [0 \quad 1]^T$. Thus the second path in G', AC, should be added to $\bar{G}'$. If we view such path deletion with the geometry of the linear system, the path space of G' remains as a plane in Figure 8.2, but $\bar{G}'$ only has one dimension of the path space left, so we need to add AC to $\bar{G}'$.

When deleting a path used in $\bar{G}$, the factor R can be updated in $O(k^2)$ time. To find a replacement row, we need to compute a sparse matrix-vector product involving G, which takes $O(n^2 \times \text{(average path length)})$ operations. Since most routing paths are short, the dominant cost will typically be the update of R. So the complexity of Algorithm 9 is $O(k^2)$.

8.4.2 End hosts Join/Leave the Overlay

To add an end host h, we use Algorithm 8 to scan all the new paths from h, for a cost of $O(nk^2)$. However, to delete an end host h, to simply use Algorithm 9 to delete each path involving h is not efficient. These paths are often dependent to each other. Thus deleting one path form $\bar{G}$ often causes adding another to-be-removed path into $\bar{G}$. To avoid that, we remove all these paths from G first, then use the updated G in Algorithm 9 to remove each h-related path in $\bar{G}$. Each path in $\bar{G}$ can be removed in $O(k^2)$ time; the total cost of end host deletion is then at most $O(nk^2)$.

8.4.3 Routing Changes

In the network, routing changes or link failures can affect multiple paths in G. Most Internet paths remain stable over days [97]. So we can incrementally measure the topology to detect changes. Each end host measures the paths to all other end hosts daily, and for each end host, such measurement load can be evenly distributed throughout the day.

For each link, we keep a list of the paths that traverse it. If any path is reported as changed for certain link(s), we will examine all other paths that go through those link(s) because it is highly likely that those paths can change their routes as well. We use Algorithms 8 and 9 to incrementally incorporate each path change.

8.5 Other Practical Issues

To further improve the scalability and accuracy, we need to have good load balancing, handle topology measurement errors, and to be robust to monitoring node failures, as discussed in this section.

8.5.1 Measurement Load Balancing

To avoid overloading any single node or its access link, we evenly distribute the measurements among the end hosts. We randomly reorder the paths in G before scanning them for selection in Algorithm 8. Since each path has equal probability to be selected for monitoring, the measurement load on each end host is similar. Note any basis set generated from Algorithm 8 is sufficient to describe all paths G. Thus the load balancing has no effect on the loss rate estimation accuracy.

8.5.2 Handling Topology Measurement Errors

Because our goal is to estimate the end-to-end path loss rate instead of any interior link loss rate, we can handle certain topology measurement inaccuracies, such as incomplete routing information and poor router alias resolution.

For completely untraceable paths, we add a direct link between the source and the destination. In our system, these paths will become selected paths for monitoring. For paths that only gives partial routes, we add links from where the normal route becomes unavailable (*e.g.*, self loops or displaying "* * *" in traceroute), to where the normal route resumes or to the destination if such anomalies persist to the end. For instance, if the measured route is (src, ip_1, "* * *", ip_2, $dest$), the path is composed of three links: (src ip_1), (ip_1, ip_2), and (ip_2, $dest$). By treating the untraceable path (segment) as a normal link, the resulting topology is equivalent to the one with complete routing information for calculating the path loss rates.

For topologies with router aliases which may present one physical link as several links, we do not really have to resolve these aliases. At worst, our failure to recognize the links as the same will result in a few more path measurements because the rank of G is higher. For these links, their corresponding entries in x_G will be assigned similar values because they are really a single link. Thus the path loss rate estimation accuracy is not affected, as verified by Internet experiments in Chapter 8.7.

8.5.3 Robustness and Real-time Response

There are some scenarios such that the overlay monitoring system can fail to provide real-time loss rate estimation for some paths. This can happen when a routing change is just detected, or the measurement node(s) crash, or some node(s) just join or leave the overlay network. Before we incrementally set up new measurement path(s) and collect results, for a short period, there are some paths for which we can not compute loss rates. However, we can still return bounds on the computed loss rate (see Chapter 8.8). For example, we can check whether all the links on the incomputable path are covered by $\bar{G}$, and if so, yield an upper bound (though possibly a pessimistic one) quickly. Furthermore, such bounds may be already sufficient for some applications.

# of nodes	# of end hosts		# of paths(r)	# of links		rank (k)	MPR (k/r)	lossy paths (Bernoulli)			lossy paths (Gilbert)		
	total	OL(n)		original	AP			real	coverage	FP	real	coverage	FP
1000	506	50	1225	1997	443	275	22%	437	99.6%	1.3%	437	100.0%	0.2%
		100	4950		791	543	11%	2073	99.0%	2.0%	1688	99.9%	0.2%
5000	2489	100	4950	9997	1615	929	19%	2271	99.1%	2.0%	2277	99.7%	0.1%
		300	44850		3797	2541	6%	19952	98.6%	4.1%	20009	99.6%	0.3%
20000	10003	100	4950	39997	2613	1318	27%	2738	98.4%	3.4%	2446	99.5%	0.6%
		500	124750		11245	6755	5%	67810	97.8%	5.5%	64733	99.5%	0.4%

# of nodes	# of end hosts		# of paths(r)	# of links		rank (k)	MPR (k/r)	lossy paths (Bernoulli)			lossy paths (Gilbert)		
	total	OL(n)		original	AP			real	coverage	FP	real	coverage	FP
1000	335	50	1225	2000	787	486	40%	704	99.0%	1.1%	579	99.6%	0.4%
		100	4950		1238	909	18%	2544	98.5%	4.6%	2539	99.7%	0.5%
5000	1680	100	4950	10000	2996	1771	36%	3067	97.5%	3.9%	3024	99.5%	0.4%
		300	44850		6263	4563	10%	29135	96.8%	7.1%	28782	99.1%	1.1%
20000	6750	100	4950	40000	5438	2606	53%	3735	98.4%	2.3%	3607	99.6%	0.4%
		500	124750		20621	13769	11%	93049	96.1%	5.7%	92821	99.1%	1.5%

# of nodes	# of end hosts		# of paths(r)	# of links		rank (k)	MPR (k/r)	lossy paths (Bernoulli)			lossy paths (Gilbert)		
	total	OL(n)		original	AP			real	coverage	FP	real	coverage	FP
1000	312	50	1225	2017	441	216	18%	1034	98.8%	2.0%	960	99.6%	0.5%
		100	4950		796	481	10%	4207	98.4%	1.6%	3979	99.6%	0.3%
5000	1608	100	4950	10047	1300	526	11%	4688	99.1%	0.6%	4633	99.8%	0.2%
		300	44850		3076	1787	4%	42331	99.2%	0.8%	42281	99.8%	0.1%
20000	6624	100	4950	40077	2034	613	12%	4847	99.8%	0.2%	4830	100.0%	0.1%
		500	124750		7460	3595	3%	122108	99.5%	0.3%	121935	99.9%	0.1%

Table 8.2: Simulation results for three types of BRITE router topologies: Barabasi-Albert (top), Waxman (middle) and hierarchical model (bottom). OL gives the number of end hosts on the overlay network. AP shows the number of links after pruning, where pruning removes the nodes and links that are not on the overlay paths. MPR is the monitored path ratio between our scheme and the pair-wise scheme. FP is the false positive rate.

# of end hosts on overlay (n)	# of paths(r)	# of links after pruning	rank (k)	MPR (k/r)	lossy paths (Bernoulli)			lossy paths (Gilbert)		
					real	coverage	FP	real	coverage	FP
50	1225	2098	1017	83%	891	99.7%	0.9%	912	100.0%	0.2%
100	4950	5413	3193	65%	3570	98.7%	1.9%	3651	99.6%	0.3%
200	19900	12218	8306	42%	14152	97.9%	3.1%	14493	99.6%	0.4%

Table 8.3: Simulation results for a real router topology. MPR and FP are defined the same as in Table 8.2.

# of nodes	OL size (n)	Barabasi-Albert model								hierarchical model							
		CV				MMR				CV				MMR			
		sender		recver		sender		recver		sender		recver		sender		recver	
		LB	NLB	LB	NLB	LB	NLB	LB	NLB	LB	NLB	LB	NLB	LB	NLB	LB	NLB
1000	50	0.62	1.10	0.56	0.94	2.41	5.91	3.07	4.09	0.52	0.96	0.53	0.87	2.28	4.80	2.51	4.29
	100	0.61	1.42	0.64	1.34	3.21	11.33	3.61	10.67	0.51	1.38	0.47	1.39	2.74	10.06	2.32	10.27
5000	100	0.44	0.89	0.47	0.97	2.25	6.11	2.36	6.50	0.49	1.18	0.53	1.39	2.60	9.18	2.97	10.16
	300	0.52	1.59	0.51	1.51	2.97	18.70	2.74	17.25	0.47	1.72	0.48	1.76	3.47	23.93	4.13	25.76
20000	100	0.36	0.55	0.40	0.59	1.93	3.20	2.29	3.69	0.48	1.17	0.43	1.09	3.04	8.86	2.56	7.09
	500	0.52	1.36	0.53	1.35	2.64	19.21	3.01	16.82	0.46	1.85	0.46	1.89	5.01	25.85	5.56	27.67

Table 8.4: Measurement load (as sender or receiver) distribution for various BRITE topologies. OL Size is the number of end hosts on overlay. “LB” means with load balancing, and “NLB” means without load balancing.

8.6 Evaluation

In this section, we present our evaluation metrics, simulation methodology and simulation results.

8.6.1 Metrics

The metrics include path loss rate estimation accuracy, variation of measurement loads among the end hosts, and speed of setup, update, and topology change adaptation.

To compare the inferred loss rate $\hat{p}$ with real loss rate p, we analyze both absolute error and error factor. The absolute error is $|p - \hat{p}|$. We adopt the error factor $F_\varepsilon(p, \hat{p})$ defined in [18] as follows:

$$F_\varepsilon(p, \hat{p}) = \max\left\{\frac{p(\varepsilon)}{\hat{p}(\varepsilon)}, \frac{\hat{p}(\varepsilon)}{p(\varepsilon)}\right\} \tag{8.14}$$

where $p(\varepsilon) = \max(\varepsilon, p)$ and $\hat{p}(\varepsilon) = \max(\varepsilon, \hat{p})$. Thus, p and $\hat{p}$ are treated as no less than ε, and then the error factor is the maximum ratio, upwards or downwards, by which they differ. We use the default value $\varepsilon = 0.001$ as in [18]. If the estimation is perfect, the error factor is one.

Furthermore, we classify a path to be lossy if its loss rate exceeds 5%, which is the threshold between "tolerable loss" and "serious loss" as defined in [136]. We report the true number of lossy paths, the percentage of real lossy paths identified (coverage) and the false positive rate, all averaged over five runs of experiment for each configuration.

There are two types of measurement load: 1) sending probes, and 2) receiving probes and computing loss rates. The load reflects the CPU and uplink/downlink bandwidth consumption. For each end host h, its measurement load is linearly proportional to, and thus denoted by the number of monitored paths with h as sender/receiver. Then we compute its variation across end hosts in terms of the *coefficient of variation* (CV) and the *maximum vs. mean ratio* (MMR), for sending load and receiving load separately. The CV of a distribution x, defined as below, is a standard metric for measuring inequality of x, while the MMR checks if there is any single node whose load is significantly higher than the average load.

$$CV(x) = \frac{standard\ deviation(x)}{mean(x)} \tag{8.15}$$

The simulations only consider undirected links, so for each monitored path, we randomly select one end host as sender and the other as receiver. This is applied to all simulations with or without load balancing.

8.6.2 Simulation Methodology

We consider the following dimensions for simulation.

- Topology type: three types of synthetic topologies from BRITE (see Chapter 8.6.3) and a real router-level topology from [56]. All the hierarchical models have similar results, we just use Barabasi-Albert at the AS level and Waxman at the router level as the representative.

- Topology size: the number of nodes ranges from 1000 to 20000 [1]. Note that the node count includes both internal nodes (i.e., routers) and end hosts.
- Fraction of end hosts on the overlay network: we define end hosts to be the nodes with the least degree. Then we randomly choose from 10% to 50% of end hosts to be on the overlay network. This gives us pessimistic results because other distributions of end hosts will probably have more sharing of the routing paths among them. We prune the graphs to remove the nodes and links that are not referenced by any path on the overlay network.
- Link loss rate distribution: 90% of the links are classified as "good" and the rest as "bad". We use two different models for assigning loss rate to links as in [92]. In the first model ($LLRD_1$), the loss rate for good links is selected uniformly at random in the 0-1% range and that for bad links is chosen in the 5-10% range. In the second model ($LLRD_2$), the loss rate ranges for good and bad links are 0-1% and 1-100% respectively. Given space limitations, most results are under model $LLRD_1$ except for Chapter 8.6.4.
- Loss model: After assigning each link a loss rate, we use either a Bernoulli or a Gilbert model to simulate the loss processes at each link. For a Bernoulli model, each packet traversing a link is dropped at independently fixed probability as the loss rate of the link. For a Gilbert model, the link fluctuates between a good state (no packet dropped) and a bad state (all packets dropped). According to Paxon's observed measurement of Internet [96], the probability of remaining in bad state is set to be 35% as in [92]. Thus, the Gilbert model is more likely to generate bursty losses than the Bernoulli model. The other state transition probabilities are selected so that the average loss rates matches the loss rate assigned to the link.

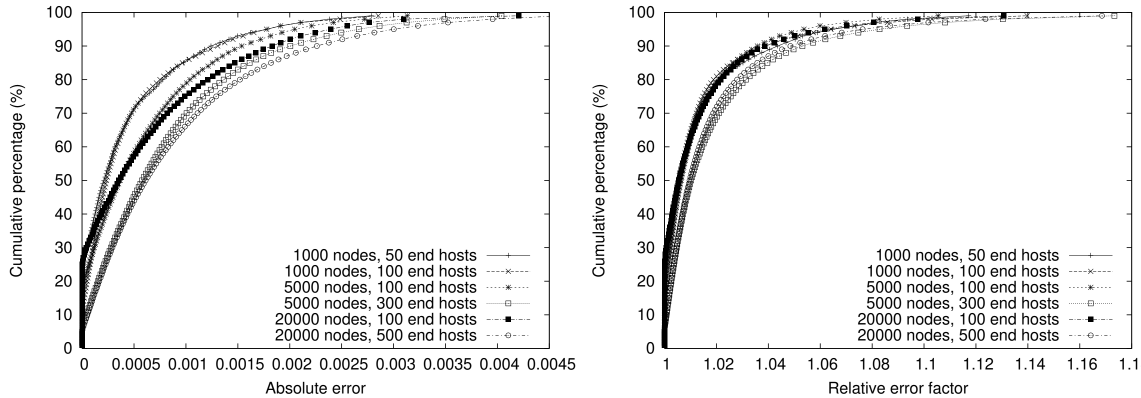
We repeat our experiments five times for each simulation configuration unless denoted otherwise, where each repetition has a new topology and new loss rate assignments. The path loss rate is simulated based on the transmission of 10000 packets. Using the loss rates of selected paths as input, we compute x_G, then the loss rates of all other paths.

8.6.3 Results for Different Topologies

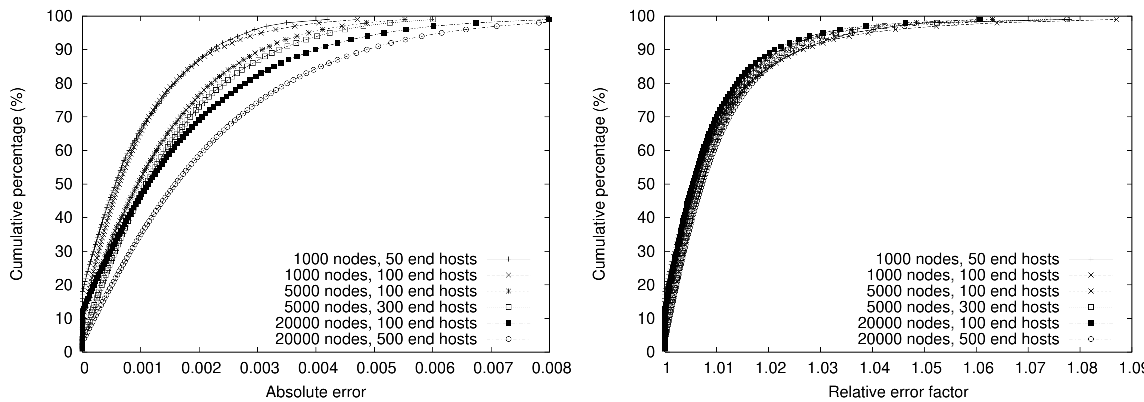
For all topologies in Chapter 8.6.2, we achieve high loss rate estimation accuracy. Results for the Bernoulli and the Gilbert models are similar. Since the Gilbert loss model is more realistic, we plot the cumulative distribution functions (CDFs) of absolute errors and error factors with the Gilbert model in Figure 8.5. For all the configurations, the absolute errors are less than 0.008 and the error factors are less than 1.18. Waxman topologies have similar results, and we omit them in the interest of space.

The lossy path inference results are shown in Table 8.2. Notice that k is much smaller than the number of IP links that the overlay network spans, which means that there are many IP links whose loss rates are unidentifiable. Although different topologies have similar asymptotic regression trend for k as $O(n \log n)$, they have different constants.

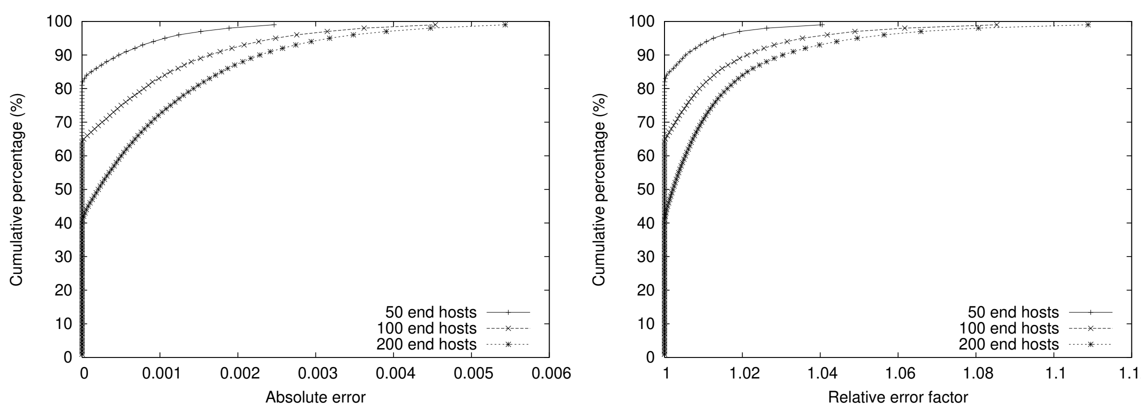
[1] 20000 is the largest topology we can simulate on a 1.5GHz Pentium 4 machine with 512M memory.



BRITE Barabasi-Albert topology



BRITE Hierarchical topology



Real topology of 284K routers

Figure 8.5: Cumulative distribution of absolute errors (left) and error factors (right) under Gilbert loss model for various topologies.

# of nodes	# of end hosts		lossy paths (Gilbert)			speed (second)	
	total	overlay	real	coverage	FP	setup	update
1000	506	50	495	99.8%	1.1%	0.13	0.08
		100	1989	99.8%	3.0%	0.91	0.17
5000	2489	100	2367	99.6%	3.5%	1.98	0.22
		300	21696	99.2%	1.4%	79.0	1.89
20000	10003	100	2686	98.8%	1.1%	3.00	0.25
		500	67817	99.0%	4.6%	1250	4.33

Table 8.5: Simulation results with model $LLRD_2$. Use the same Barabasi-Albert topologies as in Table 8.2. Refer to Table 8.2 for statistics like rank. FP is the false positive rate.

For an overlay network with given number of end hosts, the more IP links it spans on, the bigger k is. We found that Waxman topologies have the largest k among all synthetic topologies. For all configurations, the lossy path coverage is more than 96% and the false positive ratio is less than 8%. Many of the false positives and false negatives are caused by small estimation errors for paths with loss rates near the 5% threshold.

We also test our algorithms in the 284,805-node real router-level topology from [56]. There are 65,801 end host routers and 860,683 links. We get the same trend of results as illustrated in Figure 8.5 and Table 8.3. The CDFs include all the path estimates, including the monitored paths for which we know the real loss rates. Given the same number of end hosts, the ranks in the real topology are higher than those of the synthetic ones. But as we find in Chapter 8.2.2, the growth of k is still in the range of $O(n)$.

8.6.4 Results for Different Link Loss Rate Distribution and Running Time

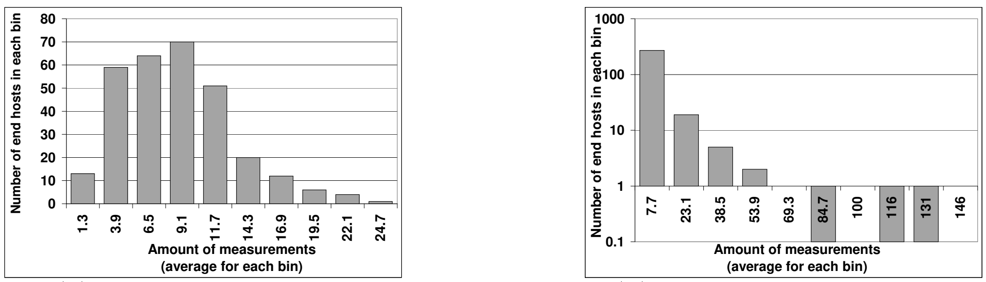
We have also run all the simulations above with model $LLRD_2$. The loss rate estimation is a bit less accurate than it is under $LLRD_1$, but we still find over 95% of the lossy paths with a false positive rate under 10%. Given space limitations, we only show the lossy path inference with the Barabasi-Albert topology model and the Gilbert loss model in Table 8.5.

The running time for $LLRD_1$ and $LLRD_2$ are similar, as in Table 8.5. All speed results in this paper are based on a 1.5 GHz Pentium 4 machine with 512M memory. Note that it takes about 20 minutes to setup (select the measurement paths) for an overlay of 500 end hosts, but only several seconds for an overlay of size 100. The update (loss rate calculation) time is small for all cases, only 4.3 seconds for 124,750 paths. Thus it is feasible to update online.

8.6.5 Results for Measurement Load Balancing

We examine the measurement load distribution for both synthetic and real topologies, and the results are shown in Table 8.4. Given the space constraints, we only show the results for Barabasi-Albert and hierarchical model. Our load balancing scheme reduces CV and MMR substantially for all cases, and especially for MMR. For instance, a 500-node overlay on a 20000-node network of Barabasi-Albert model has its MMR reduced by 7.3 times.

We further plot the histogram of measurement load distribution by putting the load values of each node into 10 equally spaced bins, and counting the number of nodes in each bin as y-axis. The x-axis denotes the center of each bin, as illustrated in Figure 8.6. With load balancing, the histogram roughly follow the normal distribution. In contrast, the histogram without load balancing is close to an exponential distribution. Note that the y-axis in this plot is logarithmic: an empty bar means that the bin contains one member, and 0.1 means the bin is empty.



(a) with load balancing (b) without load balancing

Figure 8.6: Histogram of the measurement load distribution (as sender) for an overlay of 300 end hosts on a 5000-node Barabasi-Albert topology.

8.6.6 Results for Topology Changes

We study two common scenarios in P2P and overlay networks: end hosts joining and leaving as well as routing changes. Again, the Bernoulli and the Gilbert models have similar results, thus we only show those of the Gilbert model.

End hosts join/leave

# of end hosts	# of paths	rank	lossy paths		
			real	coverage	FP
40	780	616	470	99.9%	0.2%
+5 (45)	+210 (990)	+221 (837)	+153 (623)	100.0%	0.1%
+5 (50)	+235 (1225)	+160 (997)	+172 (795)	99.8%	0.2%

Table 8.6: Simulation results for adding end hosts on a real router topology. FP is the false positive rate. Denoted as "+added_value (total_value)".

For the real router topology, we start with an overlay network of 40 random end hosts. Then we randomly add an end host to join the overlay, and repeat the process until the size of the overlay reaches 45 and 50. Averaged over three runs, the results in Table 8.6 show that there is no obvious accuracy degradation caused by accumulated numerical errors.

# of end hosts	# of paths	rank	lossy paths		
			real	coverage	FP
60	1770	1397.0	1180.3	99.9%	0.2%
-5 (55)	-285 (1485)	-245.3 (1151.7)	-210.0 (970.3)	99.8%	0.2%
-10 (50)	-260 (1225)	-156.7 (995.0)	-150.6 (819.7)	99.9%	0.1%

Table 8.7: Simulation results for deleting end hosts on a real router topology. FP is the false positive rate. Denoted as "-reduced_value (total_value)".

The average running time for adding a path is 125 msec, and for adding a node, 1.18 second. Notice that we add a block of paths together to speedup adding node (Chapter 8.3).

Similarly, for removing end hosts, we start with an overlay network of 60 random end hosts, then randomly select an end host to delete from the overlay, and repeat the process until the size of the overlay is reduced to 55 and 50. Again, the accumulated numerical error is negligible as shown in Table 8.7. As shown in Chapter 8.4, deleting a path in $\bar{G}$ is much more complicated than adding a path. With the same machine, the average time for deleting a path is 445 msec, and for deleting a node, 16.9 seconds. We note that the current implementation is not optimized: we can speed up node deletion by processing several paths simultaneously, and we can speed up path addition and deletion with iterative methods such as CGNE or GMRES [10]. Since the time to add/delete a path is $O(k^2)$, and to add/delete a node is $O(nk^2)$, we expect our updating scheme to be substantially faster than the $O(n^2k^2)$ cost of re-initialization for larger n.

Routing changes

We form an overlay network with 50 random end hosts on the real router topology. Then we simulate topology changes by randomly choosing a link that is on some path of the overlay and removing of such a link will not cause disconnection for any pair of overlay end hosts. Then we assume that the link is broken, and re-route the affected path(s). Algorithms in Chapter 8.4 incrementally incorporate each path change. Averaged over three runs, results in Table 8.8 show that we adapt quickly, and still have accurate path loss rate estimation.

# of paths affected	40.7
# of monitored paths affected	36.3
# of unique nodes affected	41.7
# of real lossy paths (before/after)	761.0/784.0
coverage (before/after)	99.8%/99.8%
false positive rate (before/after)	0.2%/0.1%
average running time	17.3 seconds

Table 8.8: Simulation results for removing a link from a real router topology.

8.7 Internet Experiments

We implemented our system on the PlanetLab [102] testbed. In this section, we present the experimental results from the implementation.

8.7.1 Methodology

We choose 51 PlanetLab hosts, each from a different organization as shown in Table 8.9. All the international PlanetLab hosts are universities.

<table>
<tr><th colspan="3">Areas and Domains</th><th># of hosts</th></tr>
<tr><td rowspan="5">US (40)</td><td colspan="2">.edu</td><td>33</td></tr>
<tr><td colspan="2">.org</td><td>3</td></tr>
<tr><td colspan="2">.net</td><td>2</td></tr>
<tr><td colspan="2">.gov</td><td>1</td></tr>
<tr><td colspan="2">.us</td><td>1</td></tr>
<tr><td rowspan="9">Inter-national (11)</td><td rowspan="5">Europe (6)</td><td>France</td><td>1</td></tr>
<tr><td>Sweden</td><td>1</td></tr>
<tr><td>Denmark</td><td>1</td></tr>
<tr><td>Germany</td><td>1</td></tr>
<tr><td>UK</td><td>2</td></tr>
<tr><td rowspan="2">Asia (2)</td><td>Taiwan</td><td>1</td></tr>
<tr><td>Hong Kong</td><td>1</td></tr>
<tr><td colspan="2">Canada</td><td>2</td></tr>
<tr><td colspan="2">Australia</td><td>1</td></tr>
</table>

Table 8.9: Distribution of PlanetLab hosts for experiments.

First, we measure the topology among these sites by simultaneously running "traceroute" to find the paths from each host to all others. Each host saves its destination IP addresses for sending measurement packets later. Then we measure the loss rates between every pair of hosts. Our measurement consists of 300 trials, each of which lasts 300 msec. During a trial, each host sends a 40-byte UDP packet [2] to every other host. Usually the hosts will finish sending before the 300 msec trial is finished. For each path, the receiver counts the number of packets received out of 300 to calculate the loss rate.

To prevent any host from receiving too many packets simultaneously, each host sends packets to other hosts in a different random order. Furthermore, any single host uses a different permutation in each trial so that each destination has equal opportunity to be sent later in each trial, because when sending packets in a batch, the packets sent later are more likely to be dropped. Such random permutations are pre-generated by each host. To ensure that all hosts in the network take measurements at the same time, we set up sender and receiver daemons, then use a well-connected server to broadcast a "START" command.

[2] 20-byte IP header + 8-byte UDP header + 12-byte data on sequence number and sending time.

Will the probing traffic itself cause losses? We did sensitivity analysis on sending frequency as shown in Figure 8.7. All experiments were executed between 1am-3am PDT June 24, 2003, when most networks are free. The traffic rate from or to each host is $(51-1) \times \textit{sending\_freq} \times 40$ bytes/sec. The number of lossy paths does not change much when the sending rate varies, except when the sending rate is over 12.8Mbps, since many servers can not sustain that sending rate. We choose a 300 msec sending interval to collect loss rate statistics quickly but with moderate bandwidth consumption.

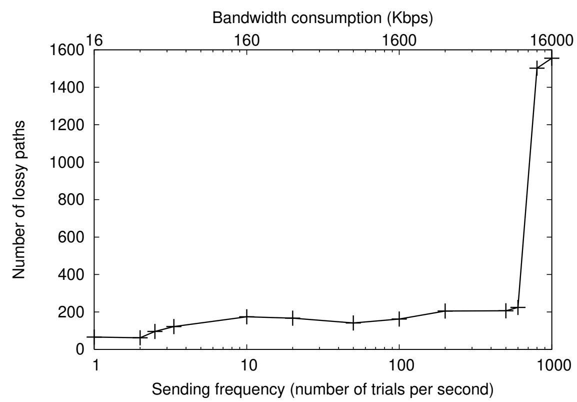


Figure 8.7: Sensitivity test of sending frequency

Note that the experiments above use $O(n^2)$ measurements so that we can compare the real loss rates with our inferred loss rates. In fact, our technique only requires $O(n \log n)$ measurements. Thus, given good load balancing, each host only needs to send to $O(\log n)$ hosts. Even for an overlay network of 400 random end hosts on the real topology of 284K nodes that we used before, $k = 18668$. If we reduce the measurement frequency to one trial per second, the traffic consumption for each host is $18668/400\times$ 40 bytes/sec $= 14.9$Kbps, which is typically less than 5% of the bandwidth of today's "broadband" Internet links. We can use adaptive measurement techniques in [40] to further reduce the overheads.

8.7.2 Results

From June 24 to June 27, 2003, we ran the experiments 100 times, mostly during peak hours 9am - 6pm PDT. Each experiment generates $51 \times 50 \times 300 = 765$K UDP packets, totaling 76.5M packets for all experiments. We run the loss rate measurements three to four times every hour, and run the pair-wise traceroute every two hours. Across the 100 runs, the average number of selected monitoring paths ($\bar{G}$) is 871.9, about one third of total number of end-to-end paths, 2550. Table 8.10 shows the loss rate distribution on all the paths of the 100 runs. About 96% of the paths are non-lossy. Among the lossy paths, most of the loss rates are less than 0.5. Though we try to choose stable nodes for experiments, about 25% of the lossy paths have 100% losses and are likely caused by node failures or other reachability problems as discussed in Chapter 8.7.2.

loss rate	[0, 0.05)	lossy path [0.05, 1.0] (4.1%)				
		[0.05, 0.1)	[0.1, 0.3)	[0.3, 0.5)	[0.5, 1.0)	1.0
%	95.9%	15.2%	31.0%	23.9%	4.3%	25.6%

Table 8.10: Loss rate distribution: lossy vs. non-lossy and the sub-percentage of lossy paths.

Accuracy and speed

When identifying the lossy paths (loss rates > 0.05), the average coverage is 95.6% and the average false positive rate is 2.75% Figure 8.8 shows the CDFs for the coverage and the false positive rate. Notice that 40 runs have 100% coverage and 90 runs have coverage over 85%. 58 runs have no false positives and 90 runs have false positive rates less than 10%.

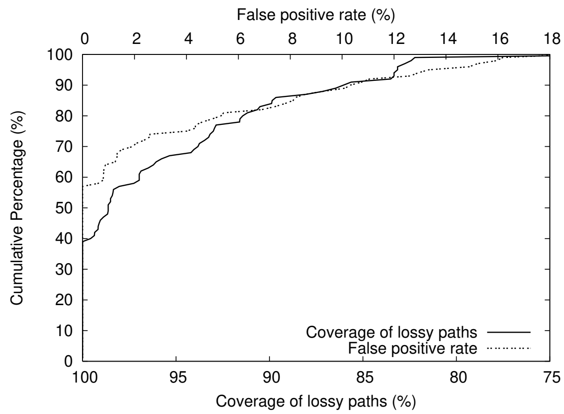


Figure 8.8: Cumulative percentage of the coverage and the false positive rates for lossy path inference in the 100 experiments.

As in the simulations, many of the false positives and false negatives absolute error across the 100 runs is only 0.0027 for all paths, and 0.0058 for lossy paths. We pick the run with the worst accuracy in coverage (69.2%), and plot the CDFs of absolute errors and error factors in Figure 8.9. Since we only use 300 packets to measure the loss rate, the loss rate precision granularity is 0.0033, so we use $\varepsilon = 0.005$ for error factor calculation. The average error factor is only 1.1 for all paths.

Even for the worst case, 95% of absolute errors in loss rate estimation are less than 0.014, and 95% of error factors are less than 2.1. To further view the overall statistics, we pick 95 percentile of absolute errors and error factors in each run, and plot the CDFs on those metrics. The results are shown in Figure 8.10. Notice that 90 runs have the 95 percentile of absolute errors less than 0.0133, and 90 runs have the 95 percentile of error factors less than 2.0.

The average running time for selecting monitoring paths based on topology measurement is 0.75 second, and for loss rate calculation of all 2550 paths is 0.16 second.

In short, we achieve high accuracy for loss rate estimation in real time with both Internet experiments and simulations.

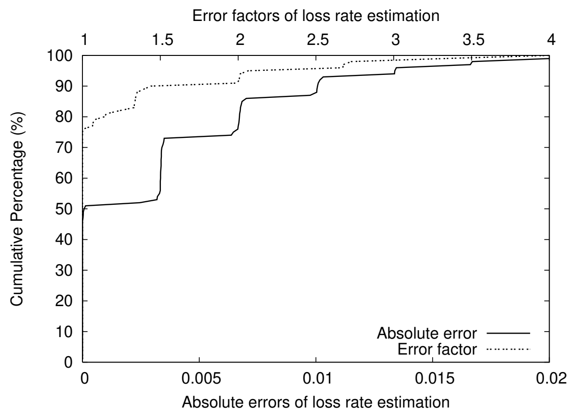


Figure 8.9: Cumulative percentage of the absolute errors and error factors for the experiment with the worst accuracy in coverage.

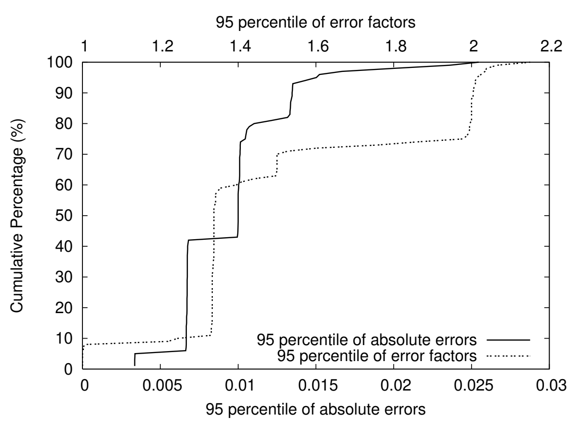


Figure 8.10: Cumulative percentage of the 95 percentile of absolute errors and error factors for the 100 experiments.

Topology error tolerance

The limitation of traceroute, which we use to measure the topology among the end hosts, led to many topology measurement inaccuracies. As found in [122], many of the routers on the paths among PlanetLab nodes have aliases. We did not use sophisticated techniques to resolve these aliases. Thus, the topology we have is far from accurate. But we are still be able to get good results as above.

Some nodes were down, or were unreachable from certain nodes. For instance, `planetlab1.ipls.internet2.planet-lab.org`, `planetlab2.sttl.internet2.planet-lab.org` and `planet2.berkeley.intel-research.net` often can not reach `uw1.accretive-dsl.nodes.planet-lab.org`, while other nodes can. Meanwhile, some routers are hidden and we only get partial routing paths. Averaging over 14 sets of traceroutes, 245 out of $51 \times 50 = 2550$ paths have no or incomplete routing information. The accurate loss rate estimation results show that the technique is robust against topology measurement errors.

8.8 Discussion

In this section, we generalize our framework to infer the path loss rate bound when we have only restricted measurements.

We note that, in addition to the equations (8.4), the unknown x_j must satisfy the inequalities $x_j \leq 0$. While we do not make use of them in our current work, these inequalities can be used in conjunction with (8.4) to bound failure probabilities, both from below and from above. For example, the loss probability l_j is bounded above by the loss probability of the least lossy path that includes link j. More generally, we have the following theorem:

Theorem 2 *Let $v \in \{0,1\}^s$ represent a network path with loss probability p, and let $w = G^T c$ for some $c \in \mathbb{R}^r$ (i.e. $w \in \mathcal{R}(G^T)$). Then*

1. *If $v \leq w$ elementwise, then $\log(1-p) \geq c^T b$*
2. *If $v \geq w$ elementwise, then $\log(1-p) \leq c^T b$*

Proof: In the first case, $v \leq w$ so that $v - w \leq 0$ elementwise. Since $x \leq 0$ elementwise, $(v-w)^T x \geq 0$, or $v^T x \geq w^T x$. We know $\log(1-p) = v^T x$ from (8.3), and $w^T x = c^T G x = c^T b$. By substitution, we have $\log(1-p) \geq c^T b$. The second case is nearly identical. ■

In principle, we can compute good upper and lower bounds on path loss rates by solving two linear programming problems:

1. Maximize $c_u^T b$ subject to $G^T c_u \geq v$,
2. Minimize $c_l^T b$ subject to $G^T c_l \leq v$.

Then $1 - \exp(c_l^T b) \leq p \leq 1 - \exp(c_u^T b)$. When $v \in \mathcal{R}(G^T)$, we have $v = G^T c_u = G^T c_l$, and the bound is tight. While this approach seems to offer bounds on path loss probabilities that are possibly optimal given the measured data, we have not yet applied the technique in practice.

8.9 Summary

In this chapter, we present a tomography-based overlay network loss rate monitoring system. For an overlay of n end hosts, the space of $O(n^2)$ paths can be characterized by a basis of $O(n \log n)$ paths. We selectively monitor these basis paths, then use the measurements to infer the loss rates of all other paths. Our approach updates the path loss rate estimation in real time, offers fast adaptation to topology changes, distributes balanced load to end hosts, and handles topology measurement errors. Both simulation and real Internet implementation show promising results.

Internet Iso-bar and TOM form the overlay network monitoring services for SCAN as in Figure 1.2. How does CDN benefit from such monitoring services? In the next chapter, we present a case study for live media streaming.

Chapter 9

Case Study: Streaming Media over A Monitoring-based Adaptive Overlay Network

Streaming delivery of media is an important technology behind applications such as Internet video conferencing and streaming playback. Traditional streaming media systems treat the underlying network as a best-effort black box, and adaptations are performed at the transmission end-points. In this chapter, we design, implement and evaluate an adaptive live streaming media system that leverages scalable monitoring services (such as Internet Iso-bar and TOM) for real-time path congestion/failure information, and an overlay network for adaptive packet relaying and buffering within the delivery infrastructure. Specifically, streaming clients in our system employs overlay routing to bypass faulty or slow links and re-establish new connection to streaming servers. We introduce a buffering technique in the overlay network to enable retransmission of lost packets during path switching, resulting in skip-free playback for live content. Using PlanetLab for Internet testbed, we show that in many cases, overlay routing can achieve lower loss rate and/or higher TCP throughput. Our PlanetLab experiments also show that our system can typically adapt to network congestion in less than five seconds, and effectively achieve skip-free streaming media playback.

For the rest of this chapter, we will first survey existing streaming media technologies in Chapter 9.1, then the diversity of overlay paths in Chapter 9.2, followed by the design and implementation of monitoring-based adaptive overlay streaming media in Chapter 9.3. Finally, we presented our evaluation methodology and results in Chapter 9.4.

9.1 Streaming Media Technologies

Streaming media applications typically require sustained network performance in terms of throughput, packet loss, and even latency for interactive applications. In contrast, the Internet provides unpredictable and time-varying service. To address the mismatch in transport requirements, many techniques have been developed and are employed by existing streaming systems. We classify the techniques into three main categories: source-coding, end-point adaptation, and infrastructure support. These techniques are complementary to the proposed adaptive overlay scheme, and can be employed in conjunction.

9.1.1 Source-coding

Video streaming is significantly more challenging than audio streaming due to the higher bit-rate and higher error-sensitivity of video data. Nevertheless, great progress has been made in video compression technology to reduce the transmission bandwidth of video. In particular, MPEG-4 enables video streaming services to wireless devices and is the basis of streaming service in third generation (3G) cellular systems [46]. More recently, H.264 [131] has been standardized and represents significant compression improvement over MPEG-4. Scalable video compression is a technology suitable for time-varying environments, and both MPEG-2 and MPEG-4 have scalable compression mode. However, due to compression performance and complexity issues, scalable compression has not been widely deployed in streaming systems today. Besides better compression efficiency, newer compression standards also handle losses better. In particular, MPEG-4 has an error-resilience feature that targets transport in both packet-erasure and bit-error channels [57].

9.1.2 End-point Adaptation

To adapt to a time-varying channel, streaming systems typically employs adaptations at the transmission end-points. Specifically, for live events, time-varying network throughput can be accommodated by adaptively adjusting the quality of the video being produced. For stored media, current streaming solutions typically employed an adaptive switching technique under which multiple copies of the same content at different bit-rates are stored. The transmission end then adaptively switches, in mid-stream, to a copy with bit-rate commensurate with prevalent network conditions. The technique has been labelled SureStream by RealNetworks® [109] and Intelligent Streaming by Microsoft® [78]. In terms of time-varying losses, many end-point adaptive techniques that employs forward error correction (FEC) and retransmissions have been proposed, including unequal error protection, where FEC is differentially applied to different parts of video data with different error-sensitivity [19]. Live streams can further benefit from client feedback information to avoid using lost frames for predictive coding of future frames. This scheme is sometimes known as Reference Picture Selection in H.263 and H.264 literature [131].

9.1.3 Infrastructure Support

Similar to data delivery, the advent of CDN technology in the Internet has improved media streaming via edge caching. The use of multiple paths for streaming in a CDN context has been investigated for the purposes of increased throughput [84] and avoiding burst losses [7]. However, none of the work address how to find multiple paths without sharing bottleneck or correlated failures.

9.2 Path Improvement with Overlay Routing

End-to-end route selection schemes have been shown effective to improve today's IP routing. Detour project found that for 30-80% of paths, there is an alternate overlay path with significantly superior quality in terms of round-trip time, loss rate and bandwidth [115]. However, for most of their datasets, they did not measure all paths simultaneously, nor do they measure all hops on a single synthetic paths simultaneously. Instead, they rely on long-term time averages of each metric of path quality. In [40], Anderson, *et al.* studied the loss rate, latency and TCP throughput improvement on an overlay network of only 12 - 16 nodes, and found that about 5% of samples have significant improvement (i.e., loss rate reduction by 0.05 and double throughput). But they uses bi-directional measurements and take the half as uni-directional performance which is not accurate for asymmetric routing and asymmetric link performance [34].

In this section, we revisit the overlay routing enhancement with improved measurements on a globally distributed overlay network of 51 nodes. We use the same PlanetLab measurement data as in Chapter 8.7. Note that for each set of measurements, the uni-directional loss rates and round-trip time for all pair of paths are measured simultaneously, and all overlay paths are measured.

We study the performance improvement achieved through single-node relay on the overlay network. For each path ($src \rightarrow dest$) considered, we find the overlay path ($src \rightarrow relay \rightarrow dest$) that gives most performance improvement. The performance of overlay path is computed with the formulas below. lr stands for loss rate and tp stands for throughput.

$$lr_{overlay} = 1 - (1 - lr(src \rightarrow relay))(1 - lr(relay \rightarrow dest)) \tag{9.1}$$

$$tp_{overlay} = min(tp(src \rightarrow relay), tp(relay \rightarrow dest)) \tag{9.2}$$

9.2.1 Loss Rate Improvement

In a total of $2550 \times 100 = 255{,}000$ path measurements, 10,980 (or 4.1%) are lossy [1]. Among them, 5,705 paths (52.0%) have loss rate reduced by 0.05 or more and 3,084 paths (28.1%) change from lossy to non-lossy.

9.2.2 TCP Throughput Improvement

For simplicity, we use the formula from [40] as below to estimate the TCP throughput.

$$throughput = \frac{\sqrt{1.5}}{rtt \times \sqrt{loss\_rate}} \tag{9.3}$$

where rtt is the round-trip time and $loss\_rate$ is the uni-directional loss rate of the path. It provides a good estimation of the path throughput for a wide range of loss rate [91].

Among the 255,000 path measurements, 60,320 paths (24%) have non-zero loss rate, and thus computable throughput. When the loss rate is 1.0, the path is disconnected,

[1] See the definition of lossy path in Chapter 8.7.

so we set the throughput to be zero. Note that most paths with small bandwidth has non-zero loss rates. So this subset covers most of the paths of which the throughput can be improved.

Since we only consider a subset of the possible overlay paths (24%), the performance improvement below is a conservative estimation. Still, the results are very encouraging. Among the 60,320 path measurements, 32,939 (54.6%) paths have improved throughput, and 13,734 (22.8%) paths have throughput which is doubled or more.

9.3 Monitoring-based Adaptive Overlay Streaming Media

Inspired by the effectiveness of overlay routing performance improvement, we propose an adaptive overlay network for streaming media that incorporates two important features. First, adaptive overlay routing is performed to bypass, in mid-session, link congestion and failure. Rather than conforming to degrading throughput or loss as many existing systems do, our adaptive overlay routing scheme can effectively and actively improve the quality of the delivery path. Second, to assist smooth or skip-free playback at the client, we employ an efficient buffering scheme in the overlay nodes so that packets transmitted during the overlay path switching time can be transmitted at a later time. Since streaming clients typically have 10 to 15 seconds of buffer time before playback starts, skip-free playback is possible if path switching takes less than 10 seconds, as we shall demonstrate in Chapter 9.4. Note that our techniques are orthogonal to the previous work in Chapter 9.1.

9.3.1 Architecture

In addition to streaming media clients and server, the system is composed of a set of overlay nodes, e.g., the CDN. For the rest of this chapter, we refer "overlay nodes" as "nodes". These nodes and the server are monitored by Internet Iso-bar and/or TOM. Thus there is an Overlay Network Operation Center (ONOC) which instruments some of the nodes to continuously measure some paths for a complete map of network conditions among the nodes.

We call this *proactive* agility, which maintains the complete map *before* receiving backup path requests. Alternatively, we can *reactively* measure and look for backup paths *after* receiving such requests. We adopt the former approach because we view the overlay monitoring service as shared by many service providers. The proactive approach offers quick response while the monitoring cost is amortized by large amount of users.

Normally a client directly connects to a server for streaming media content. It also registers the path and sets up a trigger for path performance warnings at ONOC. When the path incurs congestion/failure, ONOC detects that either through passive monitoring by the client or through some active probing instrumented by ONOC. Then ONOC searches for an alternative overlay path to bypass the faulty path [2], and sends that to the client if such path exists. The client tears down the current connection, sets up a new connection via overlay node(s), and attempts to concatenate the new streams with the old one for skip-free effect. The event driven diagram is shown in Figure 9.1.

[2] The path selection can be based on multiple metrics as in [33].

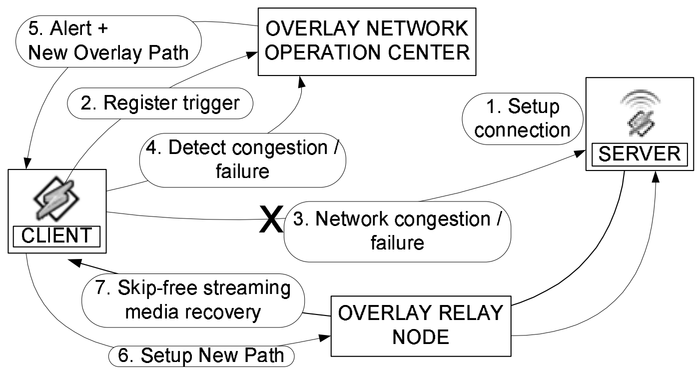


Figure 9.1: Event-driven diagram of monitoring-based adaptive overlay media streaming

Given millions of clients, we cannot monitor the network distance for each of them. We group the client by the autonomous system (AS) (or more sophisticated technique like [67]) and assume that there is an overlay node (referred as proxy) in the same AS of the client and experiences similar congestion/losses as the client does. If congestion is in the last mile of the client (*i.e.*, the access network through which the client is connected to the Internet), any scheme based on path diversity will not work. For simplicity, we assume that the client and its overlay proxy are the same node.

9.3.2 Skip-free (Lossless) Streaming Media Recovery

To bypass network congestion or failures, the client needs to tear down the current connection, switch to the overlay path and reestablish the connection to the server. For live streaming media or when the server is broadcasting the media to multiple clients, the reconnected client may lose part of the data. In this section, we discuss the protocol and implementations for continuous (skip-free) media playback.

We add a buffering layer at the server and an overlay layer at the client to work with legacy client and server softwares. The architecture is shown in Figure 9.2. Our implementation is built on Winamp [88] client and SHOUTcast [87] media server software. Media transport for SHOUTcast is carried using TCP, nevertheless, our adaptive overlay routing and buffering techniques are applicable to other transport mechanisms such as RTP/UDP.

When a client sends a connection request to the streaming media server, the media server responds with a server header indicating the server type, build, *etc.*. The buffering layer at the server keeps track of the current number of bytes broadcasted and inserts that information into the server header. The overlay layer at the client also records the current number of bytes broadcasted. This is to ensure that both the server and the client have a common reference point as the absolute byte offset from the beginning of the media, then the client starts counting the number of bytes it receives.

During normal video streaming playback, both the server and the client know the number of bytes broadcasted. The data flows directly from the SHOUTcast server to the client. Meanwhile, the most recent data streamed are cached in the buffering layer of the

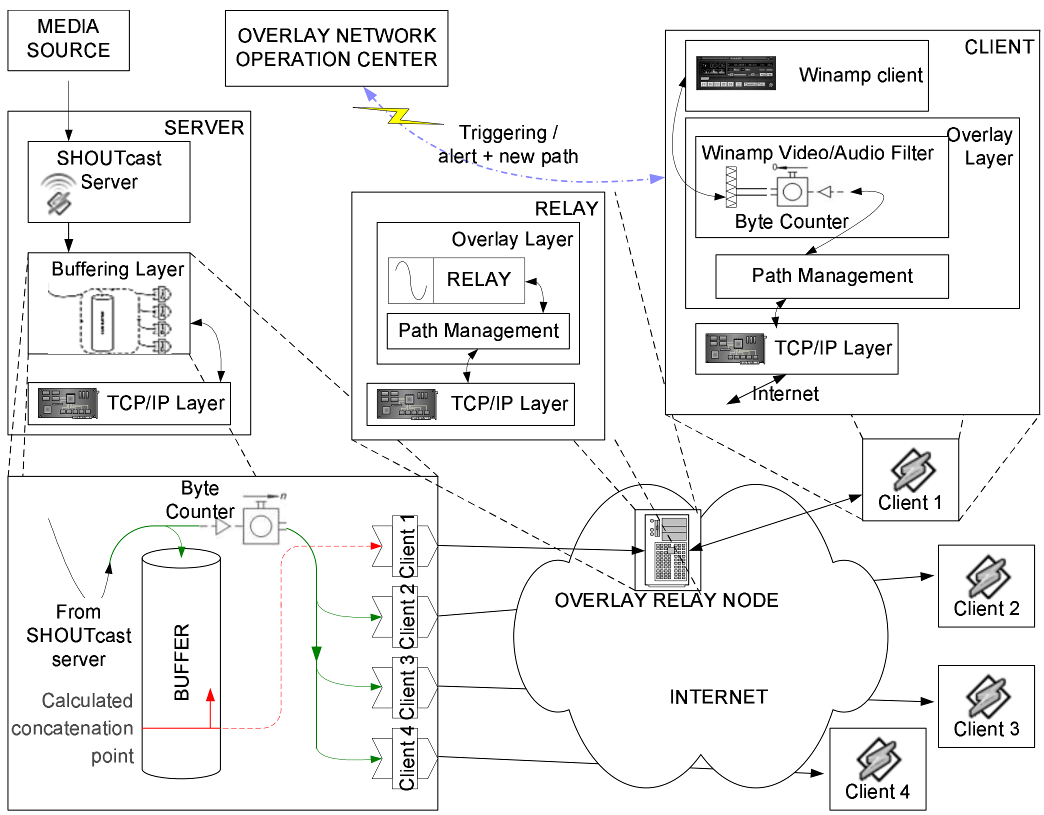


Figure 9.2: Architecture of monitoring-based adaptive overlay media streaming

server.

When a path change occurs, the client sets up a new path via overlay node(s) to connect to the streaming media server. When reconnected, the filtering layer of client issues a "reset" signal along with the clients last known number of bytes the server broadcasted (denoted as *client_count*). The server then uses the current number of bytes broadcasted (*server_count*) to determine the amount of broadcast data that the client is missing ($server\_count - client\_count$). If the amount of missing data is no larger than the buffer size, the server resumes data transmission from exactly the point where the connection dropped before, and the client will have perfect concatenation with the old media streams.

During the path change, the Winamp client is disconnected from the server, but it will continue the playback with data in its own buffer. If it receives new streams from the reestablished connection before running out of the buffer, the client will perceive continuous skip-free media playback. The path change and stream concatenation are transparent to the user.

The minimum size of buffers on the server and client for skip-free playback depends on the streaming bitrate and the adaptation time defined as the period from the moment that congestion/failure occurs to successful stream concatenation. The maximum streaming bitrate for DSL/cabel modem is 450 Kbps [110]. As shown in Chapter 9.4.2, the adaptation time is less than 10 seconds. Thus a buffer of 1MB is sufficient. Even if the missing data is bigger than the server buffer size, the server can still start from the beginning of the buffer to alleviate the losses.

Implementation Details

Each normal client has a data queue. There is a main thread reading data from the SHOUTcast server and enqueuing the data to each queue. Every client also has a dequeuing thread which dequeues the data and sends it to the client. When disconnected, the client removes the queue and exit the dequeuing thread.

For each reconnected client, a separate rewind thread (denoted as the dotted curve in the server buffering layer of Figure 9.2) will be spawned to calculate the byte offset of the streaming data where the client left off (the concatenation point in Figure 9.2), and transmit the data from the buffer. The sending rate of the rewind thread for the reconnected client is higher than that of those normal clients so that the reconnected client can refill its buffer quickly. From our experiments, the catch up time is negligible to the path switch time.

When the reconnected client catches up with normal clients, *i.e.*, the rewind thread reaches the same index of the streaming data as the main thread, the main thread will take over the client, and the rewind thread can exit. To avoid the synchronization overhead between these two threads, we have the main thread enqueue the reconnected client as well (denote the starting index as reconnection point). Since the catch up time is negligible (in the order of seconds), we assume that the queue of reconnected client can hold the transmitted data of the main thread in that period. Then the rewind thread only needs to transmit the data from the concatenation point to the reconnection point. After that, it becomes a normal dequeuing thread, transmitting data from the queue to the client.

We can also apply this technique to application-level media multicast. So the edge overlay nodes use skip-free technique for their clients. Presumably, this technique is also applicable to wireless streaming media when the handoff occurs.

9.4 Evaluation

9.4.1 Methodology

We implement the client overlay layer in 2200 lines of C# code and the server buffering layer and overlay relay layer in Java, with 1100 lines and 900 lines of code, respectively.

We deploy our system on PlanetLab [102], a global network testbed. For most of our experiments, we place the SHOUTcast server on various locations as in Chapter 9.4.2. The ONOC is at Stanford University, the overlay relay node is at HP Lab of Palo Alto, California, and the Winamp client is at U. C. Berkeley. The client is an Intel PIII/500MHz Windows XP machine with 256MB RAM on 100Mbps switched Ethernet. All other hosts are PlanetLab nodes, 1.0GHz-1.8GHz Linux machines with 512MB-883MB RAM.

The bottleneck is introduced by using a Packeteer® PacketShaper [133]. The streaming bitrate is about 600 Kbps and less than the normal available bandwidth between client and server. During streaming, we set the bandwidth limit from SHOUTcast server to Winamp client as 76 Kbps.

For congestion detection, we have the client passively monitor the throughput every 200-300 msec and use exponential-weighted moving average (EWMA) for better stability. If the smoothed throughput drops below certain threshold (e.g., 50% of streaming bitrate), we assume that congestion occurs.

The baseline for comparison is the client-server streaming media without monitoring-based adaptation. Thus when congestion occurs, the Winamp client will gradually run out of buffer, and eventually stall the media playback while our monitoring-based approach will adapt to the congestion. There are two metrics: 1) the adaptation time defined as the period from the moment that congestion/failure occurs to successful stream concatenation, and 2) Effectiveness of skip-free continuous playback. In the next section, we will report our experiment results on both metrics.

9.4.2 Experiment Results

The adaptation time breakdown are as follows.

1. Detection of congestions: the time from introducing the bottleneck link via PacketShaper to when the client detects the congestion is 0.5 second on average.

2. Client reports the congestion to the ONOC. It depends on the latency between client and the ONOC, and is even less than 0.1 second in our experiment.

3. ONOC searches for non-lossy overlay path from the client to the server, and sends to the client. We run the loss rate calculation and path finding on TOM for a 51-node overlay network on PlanetLab. On average it takes 0.66 second.

4. Client tears down old connection, sets up new connection via overlay node and gets the new media data concatenated. This depends on the distance between client to the relay node and relay node to the server. For instance, we vary the locations of servers as in Table 9.1, the average time for this step is 0.73 second (exclusive of DNS lookup time since ONOC sends the client the IP address of overlay relay node directly.). One can imagine that different locations of relay node will have similar effect.

<table>
<tr><th colspan="3">Areas and Domains</th><th>Time (second)</th></tr>
<tr><td rowspan="6">US (6)</td><td colspan="2">ucsd.edu, San Diego, CA</td><td>0.60</td></tr>
<tr><td colspan="2">nbgisp.com, Seattle, WA</td><td>0.43</td></tr>
<tr><td colspan="2">nec-labs.com, Princeton, NJ</td><td>0.59</td></tr>
<tr><td colspan="2">lbl.gov, Berkeley, CA</td><td>0.64</td></tr>
<tr><td colspan="2">berkeley.intel-research.net, Berkeley, CA</td><td>0.54</td></tr>
<tr><td colspan="2">atl.ga.us, Atlanta, GA</td><td>0.75</td></tr>
<tr><td rowspan="9">Inter-national (9)</td><td rowspan="5">Europe (5)</td><td>France</td><td>1.22</td></tr>
<tr><td>Sweden</td><td>0.93</td></tr>
<tr><td>Denmark</td><td>0.75</td></tr>
<tr><td>Germany</td><td>0.87</td></tr>
<tr><td>UK</td><td>0.70</td></tr>
<tr><td rowspan="2">Asia (2)</td><td>Taiwan</td><td>0.90</td></tr>
<tr><td>Hong Kong</td><td>0.71</td></tr>
<tr><td colspan="2">Canada</td><td>0.60</td></tr>
<tr><td colspan="2">Australia</td><td>0.78</td></tr>
</table>

Table 9.1: Distribution of SHOUTcast servers on PlanetLab and corresponding latencies.

The total adaptation time is less than three seconds. It may get larger with different location of overlay relay node, but at most just for a few seconds. Conservatively speaking, the adaptation time is less than five to ten seconds.

In addition, all the experiments with various server locations as above show perfect concatenation of streaming media and skip-free playback. Given that the overlay routing can significantly improve the loss rate and TCP throughput (Chapter 9.2), we believe that monitoring-based adaptive overlay streaming media system can effectively bypass the faulty links and achieve skip-free media playback.

In summary, in Part II, we propose a scalable network monitoring services with two components: Internet Iso-bar for latency estimation, and TOM for loss rate estimation. We further demonstrate their effectiveness with a monitoring-based adaptive overlay streaming media system.

Chapter 10

Conclusions

We endeavored in this thesis to build a content distribution network, SCAN, with good scalability, efficiency, agility and security. The Internet being an enormous, highly-dynamic, heterogeneous, and untrusted environment makes this undertaking immensely challenging.

Central to my dissertation is the combination of theory and real-world measurement-based simulation and implementation. I draw from diverse fields of applied mathematics, such as combinatorial algorithms and linear algebra as needed to better understand the design space structure. Meanwhile, real-world trace analysis, simulation and implementation conducted were used to expose the real behavior of the Internet, reveal intriguing points in the design space, and to validate the design decisions. Valuable insights gained through measurement-based analysis have ultimately led to several changes of the original design choices. To get access to real Internet measurement (often proprietary), we have actively collaborated with industrial researchers from various places, such as AT&T Labs - Research, HP Labs, Keynote Inc., Microsoft Research, and National Laboratory for Applied Network Research (NLANR).

This chapter concludes the dissertation by summarizing the major contributions of the thesis and suggesting some key directions for future work.

10.1 Thesis Summary

We made the following contributions in our thesis:

- We designed the first simulation-based network DoS resilience benchmark, and applied it to evaluate three type of object location services: the centralized, the replicated, and the emerging distributed object location service [24].

- We proposed a novel CDN, SCAN, on top of a DHT, Tapestry. SCAN dynamically places close-to-minimal number of replicas to meet client QoS (*e.g.*, latency) and server resource constraints, with overlay network topology only. Furthermore, these replicas self-organize into an application-level multicast tree. In contrast to previous work, each node in the tree (including the root) only needs to maintain states for its parent and direct children, thus truly scalable. Simulation results on both flash-crowd-like synthetic workloads and real Web server traces show that our design goals are met [26, 27].

- To reduce the replica management overhead, we investigated several clustering schemes based on *aggregated* users' access patterns to find contents that are likely to be accessed by groups of clients who are topologically close. Previous work focus on *individual* user's access patterns for prefetching - in fact we found such clustering works poorly for replication in CDN-like *shared* access environment. Evaluations based on various topologies and Web server traces show that our clustering-based replication reduces the management overhead by a factor of 50 - 100 without sacrificing end users' retrieval performance [30, 31].

- We proposed the first online Web object popularity prediction algorithm based only on hyperlink structures, and applied it for online incremental clustering and replication to adapt to changes in users' access patterns. Through this scheme, we push new content to the appropriate existing cluster replicas even before accessed. In addition to reducing the management overhead as before, it cuts down the retrieval cost by 4.6 times compared with random replication, and by 8 times compared with no replication. In comparison with pull-based replication, it can effectively improve document availability during flash crowds [30, 31].

- Traditional latency estimation system cluster end hosts based on network/geographical proximity, or rely on a few landmark sites for all distance measurements and updates. We built a novel latency estimation system, Internet Iso-bar, which clusters end hosts based on the *similarity* of their perceived network distance to a small number of landmark sites, and chooses the centroid of each cluster as monitoring site. Evaluation using real Internet measurements shows that our scheme offers much better accuracy and stability than previous clustering-based approaches, and such performance is insensitive to the number of landmarks. Our accuracy is comparable to GNP, but our distributed measurement further enables online monitoring. For a real overlay network of 106 hosts, it detects 78% of congestion/failures with 32% of false positive [28, 29].

- For accurate loss rate monitoring of an overlay network with n end hosts, previous work requires $O(n^2)$ measurement, thus unscalable. We designed and implemented an overlay monitoring system, TOM, which finds a minimal basis set of $O(n \log n)$ linearly independent paths that can fully describe all the $O(n^2)$ paths. It selectively monitors and measures the loss rates of these paths, and then applies them to estimate the loss rates of all other paths. In addition, TOM is adaptive to topology changes, has good load balancing and handles topology measurement inaccuracies. Both extensive simulation and PlanetLab experiments show that we achieve high path loss rate estimation accuracy. We can also continuously update the loss rate estimates online. For example, in the Internet experiments, the average update time is 0.16 second for all 2550 paths, the average absolute error of loss rate estimation is 0.0027 and the average error factor is 1.1. So it can precisely find lossy and non-lossy paths for clients [25].

- To demonstrate how Internet Iso-bar and TOM can benefit applications, we designed and implemented a monitoring-based adaptive overlay streaming media system. Traditional streaming media schemes treat the underlying network as a best-effort black box and perform adaptations only at the transmission end-points. Instead, our system leverages scalable monitoring services for real-time path congestion/failure information, and an overlay network for adaptive packet relaying and buffering within the delivery infrastructure. Specifically, streaming clients employs overlay routing to bypass faulty or slow links and re-establish new connection to streaming servers. Experiments on the Internet show that our system typically adapts to network congestions within five seconds and achieves skip-free streaming media playback.

10.2 Future Work

There are three general areas of future work suggested by our research.

- **General DoS resilience benchmark** While our network DoS resilience study in Chapter 3 is very specific, we feel that some of our methodology can be applied in a more general setting. In particular, our approach of simulating a *complete*, well-behaved system and then injecting malicious faults and measuring the consequences should be generally applicable. Of course, we have only simulated static clients, servers, and attackers; one future task will be to incorporate more dynamic behavior. We also hope to extend the scope of our simulations to more applications. Note that the specifics, from system setup to the threat model, vary greatly from system to system. We hope to explore techniques for combining results across multiple dimensions, possibly extending the automated approach for weight generation suggested by Bayuk [11]. As more attempts are made to quantify the DoS resilience of different systems, we hope to better understand both the nature of DoS attacks and how to measure their impact.

- **Semantic search on peer-to-peer CDN** The search in current CDNs is based on an globally uniquely identifiable object name or number. But end users are often more desirable for "semantic search", searching for all relevant documents instead of simple match with keywords or document name. The world produces between 1 and 2 exabytes (or 10^{18} bytes) of unique information per year, which is roughly 250 megabytes for every man, woman, and child on earth. Printed documents of all kinds comprise only 0.03% of the total [73]. Distributed CDN provides a scalable platform for hosting such huge scale of data. But it remains an open question as how to semantically index and search the documents which are continuously updated and added to the data warehouse. Again, we can possibly leverage the DHT techniques as in SCAN.

- **Scalable network diagnostics system** When building the tomography-based overlay network monitoring system, the traditional linear decomposition technique may have overly onerous memory requirement when the rank of path matrix exceeds 10,000. For more efficient monitored path selection, we plan to investigate the use of iterative methods [10], [76] such as CGNE or GMRES both to select paths and to compute loss rate vectors. In our preliminary experiments, the path matrix G has been well-conditioned, which suggests that iterative methods may converge quickly. We are also applying the inequality bounds in Chapter 8.8 for diagnostics, to detect which links or path segments fail when end-to-end congestion occurs.

Bibliography

[1] A. Adams, T. Bu, R. Caceres, N. Duffield, T.Friedman, J. Horowitz, F. Lo Presti, S.B. Moon, V. Paxson, and D. Towsley. The use of end-to-end multicast measurements for characterizing internal network behavior. In *IEEE Communications*, May, 2000.

[2] M. Adler, T. Bu, R. Sitaraman, and D. Towsley. Tree layout for internal network characterizations in multicast networks. In *3rd International Workshop on Networked Group Communication (NGC)*, 2001.

[3] A. Adya, P. Bahl, and L. Qiu. Analyzing browse patterns of mobile clients. In *Proceedings of SIGCOMM Internet Measurement Workshop*, 2001.

[4] Akamai Technologies Inc. `http://www.akamai.com/en/html/about/company_info.html`.

[5] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveria. Characterizing reference locality in the WWW. In *Proceeding of the IEEE Conf. on Parallel and Distributed Information Systems*, 1996.

[6] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[7] J. Apostolopoulos, T. Wong, W. Tan, and S. Wee. Multiple description streaming media content delivery networks. In *IEEE INFOCOM*, July 2002.

[8] M. Arlitt and T. Jin. Workload characterization of the 1998 World Cup Web site. HP Tech Report HPL-1999-35(R.1).

[9] A. Barbir, B. Cain, F. Douglis, M. Green, M. Hofmann, R. Nair, D. Potter, and O. Spatscheck. Known CN request-routing mechanisms. http://www.ietf.org/internet-drafts/draft-ietf-cdi-known-request-routing-00.txt.

[10] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[11] J. Bayuk. Measuring security. In *First workshop on information-security-system rating and ranking*, May 2001. http://www.acsac.org/measurement/position-papers/Bayuk.pdf.

[12] BBNPlanet. telnet://ner-routes.bbnplanet.net.

[13] A. Bestavros. Demand-based document dissemination to reduce traffic and balance load in distributed information systems. In *Proc. of the IEEE Symposium on Parallel and Distributed Processing*, 1995.

[14] A. Bestavros and C. Cunha. Server-initiated document dissemination for the WWW. In *IEEE Data Engineering Bulletin*, Sep. 1996.

[15] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proc. of IEEE INFOCOMM*, 1999.

[16] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[17] Aaron Brown and David Patterson. Towards availability benchmarks: A case study of software RAID systems. In *Proceedings of the 2000 USENIX Annual Technical Conference*, San Diego, CA, June 2000.

[18] T. Bu, N. Duffield, F. Presti, and D. Towsley. Network tomography on general topologies. In *ACM SIGMETRICS*, 2002.

[19] A. R. Calderbank and N. Seshadri. Multilevel codes for unequal error protection. *IEEE Transaction on Information Theory*, 39(4):1234–1248, 1993.

[20] CERT Coordination Center. Denial of service attacks. http://www.cert.org/tech_tips/denial_of_service.html, 1999.

[21] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *Proc. of ACM STOC*, 1997.

[22] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proceedings of FOCS*, 1999.

[23] Y. Chawathe, S. McCanne, and E. Brewer. RMX: Reliable multicast for heterogeneous networks. In *Proceedings of IEEE INFOCOM*, 2000.

[24] Y. Chen, A. Bargteil, D. Bindel, R. Katz, and J. Kubiatowicz. Quantifying network denial of service: A location service case study. In *Proc. of the Third International Conference on Information and Communications Security (ICICS)*, 2001.

[25] Y. Chen, D. Bindel, and R. H. Katz. Tomography-based overlay network monitoring. In *Proc. of ACM SIGCOMM Internet Measurement Conference (IMC)*, 2003. extended abstract.

[26] Y. Chen, R. H. Katz, and J. D. Kubiatowicz. Dynamic replica placement for scalable content delivery. In *Proc. of the First International Workshop on Peer-to-Peer Systems (IPTPS)*, Mar. 2002.

[27] Y. Chen, R. H. Katz, and J. D. Kubiatowicz. SCAN: a dynamic scalable and efficient content distribution network. In *Proc. of the First International Conference on Pervasive Computing*, Aug. 2002.

[28] Y. Chen, K. Lim, C. Overton, and R. H. Katz. On the stability of network distance estimation. In *ACM SIGMETRICS Performance Evaluation Review (PER)*, Sep. 2002.

[29] Y. Chen, C. Overton, and R. H. Katz. Internet Iso-bar: A scalable overlay distance monitoring system. *Journal of Computer Resource Management, Computer Measurement Group*, Spring Edition, 2002.

[30] Y. Chen, L. Qiu, W. Chen, L. Nguyen, and R. H. Katz. Clustering Web content for efficient replication. In *Proc. of the 10th IEEE International Conference on Network Protocols (ICNP)*, 2002.

[31] Y. Chen, L. Qiu, W. Chen, L. Nguyen, and R. H. Katz. Efficient and adaptive Web replication using content clustering. *IEEE Journal on Selected Areas in Communications (J-SAC), Special Issue on Internet and WWW Measurement, Mapping, and Modeling*, 21(6):979–994, 2003.

[32] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS*, June 2000.

[33] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the Internet using an overlay multicast architecture. In *ACM SIGCOMM*, 2001.

[34] K. Claffy, H.-W. Braun, and G. Polyzos. Measurement considerations for assessing unidirectional latencies. *Journal of Internetworking*, 1993.

[35] M. Coates, R. Castro, and R. Nowak. Maximum likelihood identification of network topology from edge-based unicast measurements. In *ACM SIGMETRICS*, 2002.

[36] Mark Coates, Alfred Hero, Robert Nowak, and Bin Yu. Internet Tomography. *IEEE Signal Processing Magazine*, 19(3):47–65, 2002.

[37] E. Cohen, B. Krishnamurthy, and J. Rexford. Improving end-to-end performance of the web using server volumes and proxy filters. In *Proceedings of ACM SIGCOMM*, Sep 1998.

[38] CERT/CC advisory ca-2000-01 Computer Emergency Response Team. Denial-of-service developments. `http://www.cert.org/advisories/CA-2000-01.html`, 2000.

[39] S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, and R. Katz. An architecture for a secure service discovery service. In *Proc. of ACM/IEEE MobiCom Conf.*, 1999.

[40] H. Balakrishnan D. G. Andersen, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proc. of ACM SOSP*, 2001.

[41] P. Danzig. Former V. P. Techonology of Akamai, Personal communication.

[42] J.W. Demmel. *Applied Numerical Linear Algebra.* SIAM, 1997.

[43] S. Dietrich, N. Long, and D. Dittrich. Anaylzing distributed denial of service tools: the Shaft case. In *Proceedings of the 14th Systems Administration Conference (LISA)*, Aug 2000.

[44] Digital Island Inc. http://www.digitalisland.com.

[45] J. Edachery, A. Sen, and F. J. Brandenburg. Graph clustering using distance-k cliques. In *Proc. of Graph Drawing*, Sep 1999.

[46] I. Elsen, F. Hartung, U. Horn, M. Kampmann, and L. Peters. Streaming technology in 3G mobile communication systems. *IEEE Computer*, 34(9):46–53, 2001.

[47] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationship of the Internet topology. In *ACM SIGCOMM*, 1999.

[48] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary cache: A scalable wide-area Web cache sharing protocol. In *Proc. of ACM SIGCOMM Conf.*, 1998.

[49] A Feldmann, R. Caceres, F. Douglis, G. Glass, and M. Rabinovich. Performance of Web proxy caching in heterogeneous bandwidth environments. In *Proceedings of IEEE Infocom*, 1999.

[50] P. Francis. Yoid: Your own Internet distribution. Technical report, ACIRI, http://www.aciri.org/yoid, April, 2000.

[51] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: A global Internet host distance estimation service. *IEEE/ACM Trans. on Networking*, Oct. 2001.

[52] D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and Jr J. W. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proc. of USENIX Symp. on OSDI*, 2000.

[53] G.H. Golub and C.F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, 1989.

[54] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[55] Google Inc. `http://www.google.com/`, 2003.

[56] R. Govindan and H. Tangmunarunkit. Heuristics for Internet map discovery. In *IEEE INFOCOM*, 2000.

[57] S. Gringeri, R. Egorov, K. Shuaib, A. Lewis, and B. Basch. Robust compression and transmission of MPEG-4 video. In *ACM MultiMedia*, 1999.

[58] Erik Guttman, Charles Perkins, John Veizades, and Michael Day. Service Location Protocol, Version 2. IETF Internet Draft, November 1998. RFC 2165.

[59] J. Gwertzman and M. Seltzer. An analysis of geographical push-caching. In *Proceedings of International Conference on Distributed Computing Systems*, 1997.

[60] J. Howard. *An Analysis of Security Incidents on the Internet.* PhD thesis, Carnegie Mellon University, Aug. 1998.

[61] Timothy A. Howes. The Lightweight Directory Access Protocol: X.500 Lite. Technical Report 95-8, Center for Information Technology Integration, U. Mich., July 1995.

[62] Computer Security Institute and Federal Bureau of Investigation. 1999 CSI/FBI computer crime and security survey. In *Computer Security Institute publication*, March 2000.

[63] IPMA project. http://www.merit.edu/ipma.

[64] K. Jain and V. Varirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. In *Proc. of IEEE FOCS*, 1999.

[65] S. Jamin, C. Jin, A. Kurc, D. Raz, and Y. Shavitt. Constrained mirror placement on the Internet. In *Proceedings of IEEE Infocom*, 2001.

[66] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley & Sons, 1990.

[67] B. Krishnamurthy and J. Wang. On network-aware clustering of Web clients. In *Proc. of SIGCOMM*, 2000.

[68] B. Krishnamurthy, C. Wills, and Y. Zhang. On the use and performance of content distribution networks. In *Proceedings of SIGCOMM Internet Measurement Workshop*, 2001.

[69] Craig Labovitz, Abha Ahuja, Abhijit Abose, and Farnam Jahanian. An experimental study of delayed Internet routing convergence. In *Proceedings of ACM SIGCOMM*, 2000.

[70] T. Leighton. The challenges of delivering content and applications on the Internet. Talk at UC Berkeley, Oct. 2002.

[71] B. Li, M. J. Golin, G. F. Italiano, X. Deng, and K. Sohraby. On the optimal placement of Web proxies in the Internet. In *Proceedings of IEEE INFOCOM*, 1999.

[72] A. Luotonen and K. Altis. World-Wide Web proxies. In *Proc. of the First International Conference on the WWW*, 1994.

[73] P. Lyman and H. R. Varian. How much information, 2002. Retrieved from http://www.sims.berkeley.edu/how-much-info.

[74] Z. M. Mao, C. Cranor, F. Douglis, M. Rabinovich, O. Spatscheck, and J. Wang. A precise and efficient evaluation of the proximity between Web clients and their local DNS servers. In *Proc. of USENIX Technical Conf.*, 2002.

[75] MediaMetrix. http://www.mediametrix.com.

[76] C. Meyer and D. Pierce. Steps toward an iterative rank-revealing method. Technical Report ISSTECH-95-013, Boeing Information and Support Services, 1995.

[77] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson. Adaptive Web caching: Towards a new caching architecture. In *Proceedings of 3rd International WWW Caching Workshop*, June, 1998.

[78] Microsoft Windows Media. http://www.microsoft.com/windows/windowsmedia.

[79] Mirror Image Internet Inc. http://www.mirror-image.com.

[80] MSNBC. http://www.msnbc.com.

[81] NASA kennedy space center server traces. http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html.

[82] R. Ng and J. Han. Efficient and effective clustering methods for data mining. In *Proc. of Intl. Conf. on VLDB*, 1994.

[83] T. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *Proc. of IEEE INFOCOM Conf.*, 2002.

[84] T. Nguyen and A. Zakhor. Path diversity with forward error correction (PDF) system for packet switched networks. In *IEEE INFOCOM*, 2003.

[85] NLANR. http://amp.nlanr.net/.

[86] Nua Analysis. http://www.usabilitynews.com/news/article637.asp, Sep. 2002.

[87] Nullsoft. Shoutcast. http://www.shoutcast.com/.

[88] Nullsoft. Winamp. http://www.winamp.com/.

[89] NY Times. http://www.cs.columbia.edu/~hgs/internet/traffic.html, Sep. 2002.

[90] H. C. Ozmutlu, N. Gautam, and R. Barton. Managing end-to-end network performance via optimized monitoring strategies. *Journal of Network and System Management, Special Issue on Management of Converged Networks*, 10(1), 2002.

[91] J. Padhye, V. Firoiu, D. Towsley, and J. Krusoe. Modeling TCP throughput: A simple model and its empirical validation. In *ACM SIGCOMM*, 1998.

[92] V. Padmanabhan, L. Qiu, and H. Wang. Server-based inference of Internet performance. In *IEEE INFOCOM*, 2003.

[93] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve World Wide Web latency. In *ACM SIGCOMM Computer Communication Review*, July 1996.

[94] V. N. Padmanabhan and L. Qiu. Content and access dynamics of a busy web site: Findings and implications. In *Proc. of ACM SIGCOMM Conf.*, 2000.

[95] M. Pastore. The Web: More than 2 billion pages strong, July 2000. `http://cyberatlas.internet.com/big_picture/traffic_patterns/article/`.

[96] V. Paxon. End-to-end Internet packet dynamics. In *ACM SIGCOMM*, 1997.

[97] V. Paxon. End-to-end routing behavior in the Internet. *IEEE/ACM Transactions on Networking*, 5(5), 1997.

[98] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *Proceedings of 3rd USENIX Symposium on Internet Technologies*, 2001.

[99] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for scalable distributed location. In *Proc. of 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, Mar. 2003.

[100] D. Pierce and J. Lewis. Sparse multifrontal rank revealing QR factorization. *SIAM Journal on Matrix Analysis and Applications*, 18(1), January 1997.

[101] A. Piszcz, N. Orlans, Z. Eyler-Walker, and D. Moore. Engineering issues for an adaptive defense network. Technical report, MITRE Technical Report MTR 01W0000103, 2001. `http://www.mitre.org/work/tech_papers/tech_papers_01/piszcz_engineering%/piszcz_engineering.pdf`.

[102] PlanetLab. http://www.planet-lab.org/.

[103] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. of the SCP SPAA*, 1997.

[104] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the placement of Web server replica. In *Proceedings of IEEE INFOCOM*, 2001.

[105] M. Rabinovich and A. Aggarwal. RaDaR: A scalable architecture for a global Web hosting service. In *Proceedings of WWW*, 1999.

[106] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal. A dynamic object replication and migration protocol for an Internet hosting service. In *Proceedings of IEEE Int. Conf. on Distributed Computing Systems*, May 1999.

[107] P. Radoslavov, R. Govindan, and D. Estrin. Topology-informed Internet replica placement. In *Proceedings of the International Workshop on Web Caching and Content Distribution*, 2001.

[108] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, 2001.

[109] Real Networks Inc. `http://www.real.com/`.

[110] RealOne Player. Pre-processing and encoding: Making the most of your bandwidth. `http://service.real.com/learnnav/ppth1.html`.

[111] P. Rodriguez and S. Sibal. SPREAD: Scalable platform for reliable and efficient automated distribution. In *Proceedings of WWW*, 2000.

[112] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of ACM Middleware*, 2001.

[113] A. Rowstron, A-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *Proceedings of International Workshop on Networked Group Communication (NGC)*, 2001.

[114] D. Rubenstein, J. F. Kurose, and D. F. Towsley. Detecting shared congestion of flows via end-to-end measurement. *IEEE/ACM Transactions on Networking*, 10(3), 2002.

[115] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The end-to-end effects of internet path selection. In *ACM SIGCOMM*, 1999.

[116] C. Schuba, I. Krsul, M. Kuhn, and et. al. Analysis of a DoS attack on TCP. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, May 1997.

[117] Semiconductor Industry Association, European Semiconductor Industry Association, Japan Electronics and Information Technology Industries Association, Korea Semiconductor Industry Association, and Taiwan Semiconductor Industry Association. *International Technology Roadmap for Semiconductors 2002 Update*. `http://public.itrs.net/`, 2003.

[118] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. In *ACM SIGCOMM IMW*, 2003.

[119] Y. Shavitt, X. Sun, A. Wool, and B. Yener. Computing the unmeasured: An algebraic approach to Internet mapping. In *IEEE INFOCOM*, 2001.

[120] O. Spatscheck and L. Peterson. Defending against DoS attacks in Scout. In *Proceedings of the 3rd Symposium on Operating System Design and Implementation*, 1999.

[121] Speedera Inc. http://www.speedera.com.

[122] N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A facility for distributed Internet measurement. In *USITS*, 2003.

[123] G. W. Stewart. *Matrix Algorithms: Basic Decompositions*. Society for Industrial and Applied Mathematics, 1998.

[124] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM*, 2001.

[125] Z. Su, Q. Yang, H. Zhang, X. Xu, and Y. Hu. Correlation-based document clustering using Web. In *Proceedings of the 34th HAWAII International conference on System Sciences*, January 2001.

[126] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network topology generators: Degree-based vs structural. In *ACM SIGCOMM*, 2002.

[127] W. Theilmann and K. Rothermel. Dynamic distance maps of Internet. In *Proceedings of IEEE Infocom*, 2000.

[128] A. Venkataramani, P. Yalagandula, R. Kokku, S. Sharif, and M. Dahlin. The potential costs and benefits of long term prefetching for content distribution. In *Proc. of Web Content Caching and Distribution Workshop 2001*, 2001.

[129] E. M. Voorhees. Implementing agglomerative hierarchical clustering algorithms for use in document retrieval. *Information Processing & Management*, 22(6):465–476, 1986.

[130] WebReaper. http://www.webreaper.net.

[131] T. Wiegand et al. Overview of the H.264/AVC video coding standard. *IEEE Trans. Circuits and Sys. for Video Technology*, 13(7), 2003.

[132] Alec Wolman, Geoffrey M. Voelker, Nitin Sharma, Neal Cardwell, Molly Brown, Tashana Landray, Denise Pinnel, Anna R. Karlin, and Henry M. Levy. Organization-based analysis of web-object sharing and caching. In *USENIX Symposium on Internet Technologies and Systems*, 1999.

[133] Workgroup Solutions. PacketShaper: Bandwidth Control and IP Management. `http://www.bandwidth-management.org/`.

[134] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an Internetwork. In *Proceedings of IEEE INFOCOM*, 1996.

[135] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proc. of ACM SIGMOD Conf.*, 1996.

[136] Y. Zhang, N Duffield, V. Paxson, and S. Shenker. On the constancy of Internet path properties. In *Proc. of SIGCOMM Internet Measurement Workshop*, 2001.

[137] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 2003.

[138] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of ACM NOSSDAV*, 2001.