Efficient, Reliable, and Secure Content Delivery

by

Yin Lin

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Bruce Maggs, Supervisor

_____
Theo Benson

_____
Jeffrey Chase

_____
Ramesh Sitaraman

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2014

Abstract

Efficient, Reliable, and Secure Content Delivery

by

Yin Lin

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Bruce Maggs, Supervisor

_____
Theo Benson

_____
Jeffrey Chase

_____
Ramesh Sitaraman

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University
2014

Abstract

Delivering content of interest to clients is one of the most important tasks of the Internet and an everlasting research question in the field of computer networking. Content distribution networks(CDNs) emerged in response to the rising demand of content providers to deliver content to clients efficiently, reliably, and securely at relatively low cost.

This dissertation explores how CDNs can achieve major performance benefits by adopting better caching strategies without changing the network, or by collaboration with ISPs and taking advantage of their better knowledge of network status and topology. It discusses the emerging trends of hybrid CDN architectures and solutions to reliability problems introduced by them. Finally, it demonstrates how CDNs could better protect both content providers and consumers from attacks and other malicious behaviors.

To Prof. Bruce Maggs for his patience, encouragement and unwavering support over the years.

Contents

Abstract **iv**

List of Tables **xi**

List of Figures **xii**

List of Abbreviations and Symbols **xiv**

Acknowledgements **xv**

1 Introduction **1**

2 Related Work and Background **7**

2.1 Content Delivery Architectures 8

2.2 Hybrid System Reliability 11

2.3 Content-Centric Networking 13

3 NetSession: a Peer-Assisted CDN **15**

3.1 Introduction 15

3.2 The NetSession system 16

3.2.1 Design Goals 16

3.2.2 Architecture 17

3.2.3 Example: Download Manager 18

3.2.4 The NetSession Interface 19

3.2.5 Interaction with Edge Servers 20

3.2.6 The NetSession Control Plane 21

3.2.7 Peer Selection . . . 22
3.2.8 Robustness . . . 23
3.2.9 Best Practices . . . 24
3.3 Measurement Study . . . 25
3.3.1 Data Set . . . 25
3.3.2 Number and Location of the Peers . . . 26
3.3.3 Content Providers . . . 28
3.3.4 Available Content . . . 29
3.4 Benefits . . . 29
3.4.1 How Well Does Peer Assist Work? . . . 29
3.4.2 Does Peer Assist Reduce Performance? . . . 31
3.4.3 Do Peers Help Improve Global Coverage? . . . 35
3.4.4 Summary . . . 36
3.5 Conclusion . . . 37
4 CDN-ISP Collaboration **38**
4.1 Introduction . . . 38
4.2 Data Set . . . 42
4.3 Evaluation . . . 43
4.4 Conclusion . . . 46
5 Cost-Efficient Caching **47**
5.1 Introduction . . . 47
5.2 Background and Motivation . . . 49
5.2.1 ICN principles and benefits . . . 49
5.2.2 Motivation: Heavy-tailed workloads . . . 52
5.3 Design Space for Caching . . . 55

5.4 Benefits of Caching 57
5.4.1 Setup 57
5.4.2 Baseline results 59
5.4.3 Summary of main results 62
5.5 Sensitivity Analysis 65
5.5.1 Single-dimension sensitivity 65
5.5.2 Best scenario for ICN-NR 68
5.5.3 Summary of key observations 70
5.6 Conclusions 70
6 Reliable Client Accounting for Hybrid CDNs 71
6.1 Introduction 71
6.2 Attacks on hybrid systems 74
6.2.1 Threat model 74
6.2.2 Attack vectors 74
6.2.3 Inflation attack on NetSession 75
6.2.4 Impact of the attack 76
6.2.5 How serious is this attack? 76
6.3 Reliable accounting 78
6.3.1 System model 78
6.3.2 Threat analysis 78
6.3.3 Approach 79
6.3.4 Require message commitment 80
6.3.5 Check logs for consistency 80
6.3.6 Check logs for plausibility 81
6.3.7 Control client pairings 82

6.3.8 Quarantine anomalous clients 82
6.3.9 Flag/throttle suspicious user behavior 83
6.3.10 Enforce resource limits 84
6.3.11 Summary 84
6.4 Application to NetSession 85
6.4.1 Overview 85
6.4.2 Assumptions 86
6.4.3 Resource certificates 86
6.4.4 Tamper-evident log 87
6.4.5 Consistency checking 88
6.4.6 Plausibility checking 90
6.4.7 Statistical tests and quarantine 90
6.4.8 Limitations 91
6.5 Evaluation 92
6.5.1 Validation 92
6.5.2 Experimental setup 93
6.5.3 Cost: Traffic 94
6.5.4 Cost: CPU 95
6.5.5 Cost: Log storage and log upload 95
6.5.6 Cost: Log processing 96
6.5.7 Examples of statistical tests 98
6.5.8 Effectivity 98
6.6 Conclusion 99
7 CDN as Security Overlay Network **100**
7.1 Introduction 100

7.2 Background and Motivation . . . 102
7.2.1 Operation Ababil . . . 102
7.2.2 Perceived Impact on Akamai . . . 103
7.2.3 Lessons Learned . . . 108
7.3 System Design . . . 109
7.3.1 Design Goals . . . 109
7.3.2 System Overview . . . 110
7.3.3 Firewall Rules . . . 112
7.4 Measurement Study . . . 113
7.4.1 Data Set . . . 114
7.4.2 Noises . . . 115
7.4.3 Repeated Attackers . . . 117
7.4.4 Attack Pervasiveness . . . 118
7.4.5 Attack Origins . . . 119
7.4.6 Attack Size . . . 121
7.5 Discussion . . . 122
7.5.1 Authentication Data Pre-Screening . . . 122
7.5.2 Client Reputation Accounting . . . 124
7.5.3 Security Analysis . . . 125
7.6 Conclusion . . . 127
8 Conclusion **128**
A Pseudo-Code for Tamper Evident Log **130**
Bibliography **132**
Biography **141**

List of Tables

3.1 Overall statistics for our data sets. 25

3.2 Global distribution of downloads for the ten largest customers. 27

3.3 Observed changes to the setting that enables content uploads. 30

3.4 Fraction of peers that have content uploads enabled. 30

5.1 Feature-Benefit Matrix for ICN 51

5.2 Analysis of requests from three CDN cache clusters from different geographical regions. 53

5.3 Comparison of simulation results for query latency on request trace and synthetic data (with best-fit Zipf). 64

5.4 Effect of access trees arity on performance gain of ICN-NR over EDGE. 67

6.1 Types of client misbehaviors. 79

6.2 Statistical tests used with NetSession-RCA 97

7.1 Number of WAF rules triggered during a day. 115

List of Figures

2.1 Content delivery taxonomy. 8

3.1 Overview of the NetSession system. 18

3.2 Global distribution of peers. 27

3.3 Overall workload characteristics. 28

3.4 Comparison of edge-only downloads and peer-assisted downloads. . . 32

3.5 Number of file copies vs. peer efficiency 33

3.6 P2P efficiency vs. number of peers. 33

3.7 Downloads of larger files are terminated more often. 34

3.8 Peer contributions in different regions. 35

4.1 Two enables for CDN-ISP collaboration. 40

4.2 Activity of CDN in two days. 44

4.3 Potential hop reduction. 45

4.4 Traffic demand by ISP network position. 45

5.1 Request popularity distribution across different geographical locations. 53

5.2 Optimal utility of different cache levels with a simplified optimization model 54

5.3 Cache placement strategies. 55

5.4 Request routing strategies. 56

5.5 An example network topology with four PoP nodes and their corresponding access trees. 59

5.6 Trace-based simulations results with proportional cache budget. . . . 63
5.7 Trace-based simulations results with uniform cache budget. 63
5.8 Sensitivity to simulation parameters. 64
5.9 Exploring the maximum performance gap between ICN-NR and EDGE 69
5.10 Bridging the performance gap between the best scenario for ICN-NR via simple extensions to EDGE 69
6.1 Effect of our attack on the NetSession logs. 76
6.2 Network traffic overhead in NetSession-RCA 94
6.3 Average log size per client 96
7.1 Attacks on Akamai customers. 104
7.2 DNS traffic handled by Akamai. 104
7.3 Phase 2 DDoS attack 105
7.4 Phase 3 DDoS attack 106
7.5 Rate control on forward requests. 106
7.6 Phase 4 DDoS attack 107
7.7 Observed probes employed by Operation Ababil. 107
7.8 Overview of a security overlay solution. 110
7.9 Repeated SQLi attackers during two weeks. 118
7.10 Number of targeted customers per /24 subnet 119
7.11 Number of maximum hourly attack sources for each customer. 120
7.12 Attack volume during a day. 121

List of Abbreviations and Symbols

Abbreviations

AS	Autonomous system.
CCN	Content-centric networking.
CDN	Content distribution network.
CMS	WordPress and Joomla content management systems
CP	Content provider such as a software company, a media website or an e-commerce vendor.
DoS	Denial-of-service.
ICN	Information-centric networking.
ISP	Internet service provider.
IXP	Internet exchange point.
P2P	Peer-to-peer.
PoP	Point of presence.
URL	Uniform resource locater.
VM	Virtual machine.
VPS	Virtual private servers.
WAF	Web application firewall.

Acknowledgements

First of all, I would like to express my sincere thanks to my advisor Bruce Maggs, who provided full support for my five years of graduate study. Not only did he offered abundant ideas and resources for my research, but he also taught me how to do solid work and critical thinking. Most importantly, he has referred me to the most successful and intelligent researchers in my field, through the collaboration with whom I gained enormous knowledge and experience. I consider myself the luckiest person to be able to work with him.

I am very grateful to Prof. Vyas Sekar and Prof. Ramesh Sitaraman, who generously volunteered large amount of their precious time for our weekly meeting. They have provided me immense help and invaluable ideas and taught me how to become a successful researcher.

Prof. Xiaowei Yang and Prof. Romit Roy Choudhury have selflessly shared their research ideas and suggestions with me in the fields of peer-to-peer reliability, mobile computing and network security. Besides them, Prof. Jeff Chase and Prof. Theo Benson have also served as my committee members and provided enormous support during my preliminary exam and final defense.

I am indebted to my collaborators Seyed K Fayazbakhsh, Ingmar Poese, Paarijaat Aditya, Mingchen Zhao, Prof. Andreas Haeberlen, Prof. Peter Druschel, Prof. Georgios Smaragdakis and many others for their great efforts to get our work published. I also owe my special thanks to many Akamai engineers and staffs such as

James Chalfant and Steve Hoey, who offered me internship opportunities at Akamai where I gained deep insight into various components of CDN, and Bill Wishon, K C Ng, Mangesh Kasbekar, David Gillman, Jon Thompson, who provided invaluable data and feedback to our research.

Last but not least, I wish to thank Yu Chen, Bi Wu, Qiang Cao, Dongtao Liu, among with other friends and colleagues. The heated discussion and group reading with you are the most rewarding and enjoyable experience in my PhD journey.

1

Introduction

The Internet is designed to share information between different parties of interest. The information flowing inside the Internet is a variety of content, such as web objects (text, image and scripts), multimedia objects(audio, video, live stream), applications(software, e-commerce), and social networks. In an oversimplified model, the Internet is made up of producers of such content, also known as content providers (CP), like CNN, Youtube, and Facebook that deliver their product or services to the end-users (a.k.a. clients) over network infrastructures maintained by Internet service providers (ISP).

In order to satisfy the soaring demand of content (Leighton, 2009), a variety of CDNs have come into being to host content for its customers (used interchangeably in this dissertation with "content providers") and serve them to the clients (interchangeable with "end-users") with low latency and high availability. They massively deployed data centers across the Internet (Ager et al., 2011), often over multiple backbones. In this way, content providers can greatly cut their capital investment on expanding their footprint globally in order to serve their worldwide clients, but instead leasing nodes and resources from CDNs' existing infrastructures. The scale

of CDNs can be as small as a single data center such as RapidShare (Antoniades et al., 2009), or as large as Akamai, whose data centers can be found across almost 1,000 networks in more than 1,800 locations. Some large software or service vendors also operate their own CDNs. For example, Google (Tariq et al., 2009) uses tens of data centers to deliver their products and services; Microsoft Azure operates in 24 locations; Amazon AWS owns 6 data centers over more than 20 locations.

The dominance of the total Internet traffic by CDNs and the important role they play can never be overestimated. Studies (Gerber and Doverspike, 2011; Poese et al., 2010) show that more than half of the North American and a European tier-1 carrier's traffic are contributed by various CDNs. Google is claimed to be responsible for more than 10% of the total Internet inter-domain traffic (Labovitz et al., 2010). 20% of the global web traffic is contributed by Akamai (Nygren et al., 2010). As another example, during its busy hours, Netflix is reported to utilize more than 30% of North America's total traffic (Inc., 2011).

With its growing popularity and importance, CDNs have been seeking ways to meet the ever-growing demand, provide better performance, promise higher availability, enable greater deployment agility, and reduce the operating and capital costs to guarantee their places in the competitive content distribution markets (Qureshi et al., 2009). Many innovations and new architectures have appeared during the past a few decades. Hybrid, or "peer-assisted" CDNs try to break the boundary between traditional infrastructure-based and peer-to-peer (P2P) data transfers. They are trying to combine the merits of both architectures. The P2P component of a hybrid CDN can shift some workload to clients' machines by leveraging their hardware and bandwidth resources. The dedicated servers, the centrally managed elements, on the other hand, serve as a strong and reliable backup delivery mechanism and ensures high availability as traditional CDNs. The design and case study of a hybrid CDN is described in Chapter 3.

CDNs can also achieve more efficient delivery by actively collaborating with ISPs, who own and thus have a better view of the networks. There are two good incentives for CDNs to seek such collaborations. On the one hand, it takes time for CDNs to find the most cost-efficient locations to deploy their new servers (Nygren et al., 2010), and ISPs can offer physical or virtual machines to CDNs that greatly reduces their need to expand footprints physically and statically. On the other hand, in order to best serve end users, CDNs have to assign servers that are closest to them. However, CDNs do not perfectly know real-time network topology and status, and may make sub-optimal assignments or slow adjustments to link congestions. ISP, as the network operator, can provide extremely helpful recommendations for CDNs in this regard. A framework for such collaboration is proposed in Chapter 4.

Another popular idea for efficient content delivery, as an alternative to CDNs, is the information-centric networking (ICN). It was brought up more than twenty years ago and has seen significant renewed interest recently. The resurgence of this idea was triggered largely by Jacobson et al. (2009)'s efforts on the CCN project, and followed up by several workshops, conferences, and support from the industry (Li et al., 2013; CCNIndustry, 2012). It is claimed to benefit both content providers and end-users with better performance, enhanced security, and stronger mobility. Lying in the center of the ICN concept is the observation that the Internet has evolved from a means of communication to a rich pool of resources; users are more interested in the content, agnostic of locations they are trying to connect to retrieve that content from. The traditional location-based architecture such as IP does not adapt to this content-centric focus and introduces unnecessary addressing overhead. Besides, the centralized content hosting or caching approaches for content delivery are also inefficient. ICN argues that content should be cached pervasively on the routers and network nodes that are closer to the end-users. The existing ICN proposals often require significant change to the entire network infrastructure such as ability of routers

to store content and the complexity introduced by these changes are questioned by their objectors as impractical or not worthwhile. Chapter 5 discusses the possibility of realizing most of ICN's benefits without any changes to the network and building an ICN that is incrementally deployable.

Innovations are always accompanied by new challenges. Several undiscovered research problems arise with the arrival of new content delivery architectures. Maintaining service reliability is one of these challenges. Take hybrid CDN as an example. The peer-to-peer communication in the hybrid architecture are not directly observable to the CDN operators, who wish to monitor the clients' activities in order to ensure their service quality and make it accountable to their customers. A malicious or compromised client can delay or abort transfers, tamper with the content sent to other clients, or misreport download/upload activities between peers in order to manipulate the accounting for certain customers. Chapter 6 performed an attack against a specific hybrid CDN to demonstrate how easy such misbehaviors can be conducted by the clients and proposed a system that removes these vulnerabilities and ensures accounting reliability.

Providing additional security and protection for customers is another important responsibility for CDNs on top of efficiency and reliability. The security features are not only highly attractive to CDNs because they can be developed as add-on services and products that generate revenues, but also in a large part compulsory because of the nature of CDNs being a large aggregate resource pool that falls too easy a target for the attackers. Triukose et al. (2009) show that because a CDN maintain a separate connection with the origin server from that with the client, a smart attacker could abuse these decoupled TCP connections to amplify workload and bandwidth consumption on the origin server thanks to the existence of the CDN. The goal CDNs attempt to achieve is twofold. On the one hand, CDNs want to minimize the impact of attacks on their deployed servers so that they can promise resilient and consistent

performance. On the other hand, they also need to direct the least malicious or malformed traffic to the servers that hold original content and therefore make their customers "hidden" from attackers. One way to achieve this goal is to set up a virtual network, known as the "overlay network", on top of Internet (referred to as the "underlay") as an isolation. Other mechanisms include dropping packets on non-service ports, building rule-based application layer firewalls and so on. Chapter 7 makes a case study of a security overlay network design and proposes mechanisms to mitigate DoS attacks at the edges.

This dissertation covers three major aspects of content delivery: efficiency, reliability and security. It examines the existing systems and technologies that have been proposed or implemented to achieve these qualitative goals, discusses their advantages and shortcomings, and suggests frameworks to overcome the challenges faced by these systems. It researches three approaches to more efficient content delivery, namely hybrid CDNs, CDN-ISP collaboration and ICN by studying example systems, measuring industry datasets, and performing trace-driven simulations. It discusses reliability problems involved with hybrid CDNs and proposes a prototype to address them. With respect to security, it analyzes data collected from an example industry solution to better understand attacks on CDNs in the real world. It also discusses ways that CDNs can better protect their customers.

The rest of this dissertation is organized as follows: Chapter 2 lays out the backgrounds of various CDNs and briefly reviews related research work. The efficiency, reliability and security aspects of content delivery are discussed in Chapter 3-5, Chapter 6 and Chapter 7, respectively. Chapter 3 demonstrates how CDNs can achieve better performance by integrating P2P elements and gives the case study of NetSession, a concrete hybrid CDN design. Chapter 4 discusses how CDNs can perform better by collaborating with ISPs. Chapter 5 shows how the qualitative benefits of ICN, another efficient content delivery approach, can be gained without changing

existing networks. Chapter 6 examines potential reliability problem in accounting hybrid CDNs. Chapter 7 studies the security overlay networks. Chapter 8 concludes the dissertation.

2

Related Work and Background

This chapter provides a background for works discussed in this dissertation and give an overview of research efforts in the content distribution area. It is reported (Ager et al., 2012; Gerber and Doverspike, 2011; Labovitz et al., 2010; Poese et al., 2010) that delivery of content dominates today's Internet traffic, which is growing at a skyrocketing 30% rate each year (Networking and Index., 2013). Various CDN architectures, which are overlays that build upon existing network infrastructure, have emerged to deal with the rising demand for content and meet the requirement of scalability and reliability required by commercial-grade applications. These architectures can be categorized by operating parties, such as independent entities, ISPs, content providers, self-organized clients, or combinations of the above. An overview of these architectures are given in Section 2.1. Among them is the hybrid CDN that incorporates a peer-to-peer network and is discussed in great detail in this dissertation because of its outstanding popularity and wide adoption by the industry recently. The reliability and accountability of P2P or hybrid network has been explored by immense amounts of research and are briefly enumerated in Section 2.2. Finally, as a promising alternative to CDNs, ICN is a relatively novel concept that

attempts to align the Internet's more location-centric architecture with its more content-centric nature, and thus achieve better caching efficiency and less addressing overhead. Section 2.3 introduces major prior work in this area.

2.1 Content Delivery Architectures

This section categorizes CDN architectures according to their major participants and the resulting taxonomy is provided in Figure 2.1.

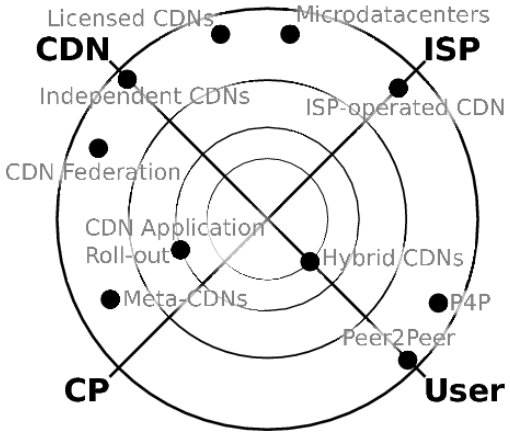


FIGURE 2.1: Content delivery taxonomy.

Independent CDNs: Independent CDNs have a solid business relationship with content providers and host data for them that are delivered to end users around the world. Depending on the scale of an independent CDN, there can be three approaches to deploy their servers.

Small CDNs, one-click hosters, and applications running in public clouds usually distribute their content from a single central location. This approach benefit CDNs with the economics of lightweight deployment (Armbrust et al., 2009), flexibility of multihoming (Goldenberg et al., 2004), and connectivity of IXPs (Ager et al., 2012), but also has the risk of single point of failure (Slashdot, 2011). It also suffers from severe network latency with distributed end-user locations.

CDNs such as EdgeCast, BitGravity and Limelight and cloud providers such as Microsoft Azure and Amazon CloudFront adopts another approach. They deploy several large data centers that are often IXP members to establish direct connectivity to a large number of networks. The approach also benefit from improved reliability by eliminating the single point of failure problem.

The third approach consists of highly distributed servers that scatter across a large number of networks. It is employed by large CDNs such as Akamai to leverage replication of content and locations for high availability and flexibility. It can balance traffic across different locations and thus react quickly to crowds and enjoy improved latency.

Peer-to-peer: P2P network was first promulgated as a collaborative resource sharing environment where each participating network node, known as "peers", acts both as a consumer and a supplier of resources, as opposed to the traditional client-server model where client nodes only consume resources from a centralized server without contributing to others. The history of P2P can be traced back to the early 1960s with the invention of APRANET (McQuillan and Walden, 1977) and has seen significant development recently by protocols such as BitTorrent (Cohen, 2003). Despite the copyright and privacy concerns that come along with it and perhaps the declining of it by certain regions of the world, P2P is still a significant contributor to the total Internet traffic and has been proven to scale extremely well during flash crowds (Yang and de Veciana, 2004).

Hybrid CDNs: Hybrid CDNs are an effort to combine the merits of traditional client-server and peer-to-peer architecture. In a hybrid CDN, content are broken into chunk that can be retrieved both from the CDN's servers and from other end-users who have installed the CDN's client software. In order to coordinate the clients, the CDN set aside a group of servers, called control plane servers, to direct the client softwares to download which chunks from which peers. Examples of commer-

cial hybrid CDNs include LiveSky (Yin et al., 2009), who delivers streaming video, Thunder (Dhungel et al., 2012), an application aggregator with high penetration in China, and NetSession (Zhao et al., 2013), a low cost solution developed by Akamai to deliver software updates and large files for its customers. Hybrid CDNs highlight its leverage of end-users' hardware and bandwidth resource for better scalability and cost reduction on CDNs' and content providers' side. A recent studies (Huang et al., 2008a) showed that as much as 80% of the traffic can be shifted from CDNs' servers to the end-users without hurting their download experience.

ISP-operated CDNs: ISPs have witnessed the success and rapid growth of CDNs who deploy servers and data centers inside their networks, and are motivated to build their own CDNs for their enormous revenue generation potential. Examples of ISPs starting to create their own CDNs include AT&T, Verizon and their CDN architectures generally follow those of independent CDNs except for a less distributed deployment. Other examples include Telefonica and Level3 (Laoutaris et al., 2009, 2011), who have large footprints around the world and are also building their own CDNs for efficient global content distribution. ISP-operated CDNs benefit from their ownership of the last mile network and its closeness to the end-users.

Licensed CDNs: Licensed CDNs are partnerships between CDNs and ISPs to combine the former one's steady customer base with the latter one's large end-user base. In this business model, the CDN provides content delivery software that runs on servers owned by the ISP. The servers collect logs for its service and reports them back to the CDN. The revenue derived from content providers are divided between the two parties. This architecture saves the ISP from the difficulty to negotiate directly with a large number of content providers and the CDN from necessity for vast hardware and network investment to expand its footprint.

Application-based CDNs: Large content providers have also demonstrated interest to build their own CDN to server their clients. Software giants such as Google

have generated such enormous traffic that they are considering rolling out their own content delivery system to amortize te cost. In order to achieve this, Google has deployed numerous data centers that sit upon high speed backbone networks and connects with major ISPs via IXPs and private peering. Netflix has also launched its own CDN called Open Connect Network (OCN, 2012) with a BGP interface through which ISPs can advertise their mapping of subnets to Open Connect Network servers.

Meta CDNs: Large content providers may contract with multiple CDNs to achieve best availability and global coverage nowadays. Meta-CDNs have come into being as a broker to help them select best CDNs by collecting performance metrics from the end-users. Examples of meta-CDNs include Cedexis and Conviva (Dobrian et al., 2011). They benefit content providers by selecting an alternative CDN when a CDN is not performing well or is not assigning the optimal server to serve an end-user.

CDN Federations: CDN federations have formed, usually among small CDNs or ISPs, to reduce investment cost on footprint expansion and to strengthen individual CDNs' negotiating power with the content providers and ISPs. Members of a CDN federation can populate its content to their partners' servers in locations where they have no footprint. The transit cost is incurred only once for replicating the content and all end-users in the new location can be served with low latency. Operator Carrier Exchange (OCX) is an example of CDN federation (Rayburn, 2011).

2.2 Hybrid System Reliability

The concept of peer-assisted CDN has been explored by both academia (Peterson and Sirer, 2009; Freedman, 2010) and industry (Vu et al., 2010; Yin et al., 2009). Lu et al. (2012) makes a good comparison of these systems. The benefits of the hybrid design were foreseen by studies (Huang et al., 2008b; Karagiannis et al., 2005) about a decade ago and have been proven by recent measurements on commercial hybrid CDNs such as LiveSky (Yin et al., 2009), PPLive (Vu et al., 2010) and Net-

Session (Zhao et al., 2013).

Built into the heart of swarming protocols like BitTorrent (Cohen, 2003) is a mechanism that rewards end-users according to the bandwidth they contribute. The assumption behind the mechanism is that clients are unwilling to upload unless they get benefits from it. A large body of research work has been unwound around developing such incentive systems that ensure fairness and encourage uploading among participating peers. Dandelion (Sirivianos et al., 2007a) proposes a fair-exchange protocol that prevents freeriding by crediting uploading clients with virtual currencies that can be used to redeem downloads. A centralized infrastructure is involved to circulate the currencies and blacklist malbehaving clients. Antfarm (Peterson and Sirer, 2009) also issues virtual currencies in the form of cryptographically signed tokens. Forgery or double-spending of the tokens can be detected by a centralized coordinator. Unlike Dandelion, tokens in Antfarm does not only serve as a means of payment for chunk downloads, but also as a bearer of transaction record that can be used for accounting purposes. The bandwidth of the infrastructure servers can then be carefully directed to maximize the aggregate bandwidth of the swarms by extracting statistics from the tokens recycled.

The reliable accounting problem in distributed systems like P2P was earlier studied by Seuken and Parkes (2011). It is shown that a Sybil-proof solution to this problem does not exist without help of a trusted infrastructure and with only cooperative or rational peers. However, when Byzantine peers are considered and when number of Sybils only constitutes a small portion of participating nodes, it is possible to detect misbehaving peers automatically using accountability systems like PeerReview (Haeberlen et al., 2007), which achieve fault-detection by keeping tamper-evident logs. RCA (Aditya et al., 2012) further develops the idea and proposes a reliable accounting system for hybrid systems that withstand Sybil attacks.

Many other efforts have been attempted to mitigate Sybil attacks. Following

(Douceur, 2002)'s suggestion of resorting to resource testing, many different type of resources have been explored by subsequent works, such as IP addresses (Freedman and Morris, 2002), physical location (Bazzi and Konjevod, 2005), money Margolin and Levine (2008), and even social relationships with trusted users Yu et al. (2006, 2008, 2009). Besides resource testing, certification authorities is another source of trust to rely on for Sybil prevention and its viability is demonstrated by Adya et al. (2002).

Anomaly detection (Chandola et al., 2009) is another important aspect of hybrid system reliability. Early researches include intrusion-detection systems (Denning, 1987) that can be traced back to decades ago. In a security context (Barreno et al., 2006), studies show that frog-boiling or similar attacks (Chan-Tin et al., 2009) can bypass anomaly detection by shifting the system's workload gradually and thus not leaving sudden bumps that triggers alarms. Aditya et al. (2012) cope with this problem by adding consistency and invariant checks as a supplement to anomaly detection, which is independent of the system's workload.

2.3 Content-Centric Networking

Among the large body of ICN related works, the CCN/NDN and DONA projects are highlighted by their envisioning of pervasive cache placement (Jacobson et al., 2010). These two project differ in the way that they handle routing. In DONA, routing is consistently done in nearest-replica fashion. In CCN/NDN, this approach is only used for LANs; it adopts shortest path to origin for WANs. They have also been designed to support other qualitative ICN features such as security and naming. For example, both human-readable names and self-certifying names are supported by NDN.

The PURSUIT (Fotiou et al., 2010) project, based on in predecessor PSIRP, follows the publish-subscribe routing paradigm. It was a combined efforts of many

European universities and was completed in February 2013. Compared with CCN, data can travel through a different path than the interest packets take in PSIRP. Besides, PSIRP attaches zFilters (Jokela et al., 2009) to packets and thus alleviate routers from exhausting its resources in order to keep track of interest packet states. This is a major contribution to the network architecture revolution.

The NetInf architecture (Ahlgren et al., 2010) is another approach for ICN. It was initially envisioned in the 4WARD project. It is also based on a name lookup resolution mechanism and uses a distributed hash table. This design supports an information abstraction model that enables multiple different representations of the same object that can be used in an end-to-end fashion. It is suggested by Fayazbakhsh et al. (2013) that the name resolution services of all these projects can be adapted to not require pervasive caching and are thus incrementally deployable.

Although not strictly an ICN, the Serval project (Nordstrom et al., 2012) is also an attempt to separate content naming from its location. It focuses on service-centric networking and proposes an end-host stack that supports such architectures. Much functionality of Serval is placed on end-hosts. Serval also provides API specifications and other implementation details for integration of ICNs or service-centric networks into the stack of state-of-art computers.

In parallel with various efforts to explore and implement novel ICN architectures, another group of research have focused on evaluating the practicality and cost-efficiency of such architectures. Perino and Varvello (2011) studies the scalability of ICN-capable routers and economic concerns when deployed worldwide. Arianfar et al. (2011) addresses privacy concerns about ICN. Ahlgren et al. (2012)) surveys legal problems with ICN such as access and copyright restrictions. Ghodsi et al. (2011) raises doubt about the actual performance boost that ICN can provide despite its potential in other aspects.

3

NetSession: a Peer-Assisted CDN

3.1 Introduction

Traditional CDNs can be divided into two classes: CDNs that rely solely upon centrally managed infrastructures, and ones that requires no infrastructure at all. A well-known example of *infrastructure-based* CDN is Akamai's main CDN (Dilley et al., 2002). It owns more than 119,000 servers in 80 countries within over 1,100 Internet networks (Akamai, 2014). Infrastructure-based CDNs typically have the benefit of professional administrators and amply provisioned resources—they can control and authenticate the content they distribute, and they can achieve reliable accounting and ensure content integrity, but they are also expensive to scale. Peer-to-peer CDNs, represented by BitTorrent (Cohen, 2003) must rely on resources contributed by their peers, which means that their properties are often the exact opposite: they are inexpensive and easy to scale but seem to be plagued by security issues and low QoS.

NetSession is a hybrid CDN that tries to reconcile these two traditional CDNs with very different trade-offs. Surprisingly, it is able to achieve the 'best of both

worlds': it can offer most of the benefits of both architectures while avoiding most of the drawbacks. This is because the strengths of the infrastructure and the peers complement each other: The infrastructure provides a central point of coordination that can quickly match up peers and can add resources when peers cannot provide adequate QoS; the peers provide resources and scalability, and they extend the 'reach' of the infrastructure to underserved areas.

Finally, we report some observations from the day-to-day operation of the NetSession system. Among other things, we discover a surprising degree of user mobility in the system, and we describe what appear to be the effects of cloning and re-imaging client installations.

The rest of this chapter is structured as follows: Section 3.2 describes NetSession's design goals and implementation. Section 3.3 conducts a measurement study providing an overview of the scale on which NetSession operates and the sorts of content it is used to deliver. Section 3.4 analyzes of whether NetSession meets its design goals and realizes the potential benefits of hybrid systems. We present our conclusions in Section 3.5.

3.2 The NetSession system

NetSession system was originally developed by RedSwoosh and has been operated by Akamai since 2007. As of October 2012, NetSession has been in service for five years and has almost 26 million users in 239 countries.

3.2.1 Design Goals

NetSession was designed with the following three high-level goals in mind:

1. A substantial fraction of the content should be delivered by the peers.

2. Peer-assisted delivery QoS should be comparable to that of infrastructure-based

delivery; in particular,

(a) downloads should be no less reliable; and

(b) downloads should not be much slower.

3. The system should offer reliable accounting for services provided.

In other words, the system was meant to combine the key benefits of peer-to-peer CDNs (scalability) and infrastructure CDNs (quality of service). The third goal was an operational requirement: Content providers, who pay for the CDN's services, expect detailed logs that show the amount and the quality of the services provided. There were also two explicit non-goals:

1. The system need not be *more* reliable than an infrastructure-based CDN; and

2. Peers need not contribute equally.

The first point sets realistic expectations about security; the second point reflects the fact that the system has a large infrastructure to fall back on, so some proportion of peers who opt out of serving content to peers would not be a concern. Serving content reliably and with good QoS is more important than minimizing load on the infrastructure.

3.2.2 Architecture

NetSession distributes content via an infrastructure of *edge servers* that are operated by Akamai, and a number of user-operated peers that have special software, the *NetSession Interface* (Section 3.2.4), installed on them. In addition to the edge servers, the infrastructure also contains a group of NetSession-specific servers called the *NetSession control plane* (Section 3.2.6), which serve as coordinators and perform accounting, but do not directly serve any content. Figure 3.1 illustrates the high-level interaction between these components.

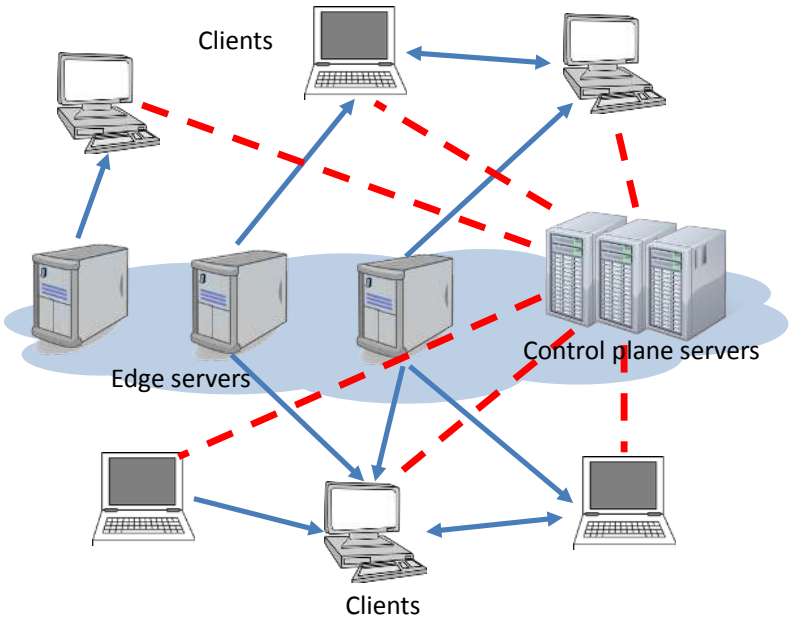


FIGURE 3.1: Overview of the NetSession system.

3.2.3 Example: Download Manager

The Download Manager (DLM) is one of several applications that use the NetSession system; a typical use case is to distribute a large object several gigabytes in size, such as software installation images.

When a user attempts to download an object that is distributed using DLM, she is first asked to install the NetSession Interface if it does not exist. Once installed, the NetSession Interface starts downloading the content from the edge servers; in parallel, it queries the control plane for a list of nearby peers that already have a copy of the object. If suitable peers are found, the local peer and the selected peers attempt to contact each other and exchange as much data as possible; however, the download from the edge servers continues in parallel. Thus, if a peer is "unlucky" and picks peers that are slow or unreliable, the infrastructure can cover the difference, so that user experience does not suffer as a result.

Users can pause and resume downloads, and they can continue downloads that

were aborted earlier, e.g., because the peer lost network connectivity or the peer's hard drive was full. Once the download completes, the NetSession Interface software remains on the peer and can be reused for future downloads, and it can upload the downloaded content to other peers.

3.2.4 The NetSession Interface

The NetSession Interface is available for Windows and Mac OS. It is implemented as a background application which runs whenever the user is logged into their system. This design choice is different from many P2P clients, which must be launched explicitly by the user. The short session times that have been observed in P2P systems (Bhagwan et al., 2003; Guo et al., 2005; Saroiu et al., 2002) suggest that users launch the client only when they intend to download something, so the time window in which objects can be uploaded to other peers tends to be very short. As a persistent background application, NetSession does not have this problem, but in return, it must take great care not to inconvenience the user. We discuss some of its best practices in Section 3.2.9.

Whenever the NetSession Interface is active and the peer is online, it maintains a TCP connection to the control plane. When a download is started, the peer uses this control connection to query the control plane for other peers. The connection is also useful for opening peer-to-peer connections through NATs, which in most cases requires coordination between the peers; the control plane can facilitate this by informing both endpoints using their control connection. Finally, peers use the connection to learn about configuration updates, and they report usage statistics, which are used for billing, performance monitoring, and to generate reports for customers. Each peer has a unique GUID, which is chosen at random during installation.

NetSession uses the standard HTTP(S) protocol to download content from edge servers; for downloads from peers, it uses a swarming protocol not unlike BitTorrent's.

As in BitTorrent, objects are broken into fixed-size pieces that can be downloaded and their content hashes verified separately, and peers exchange information about which pieces of the file they have locally available.

A key difference to BitTorrent is the absence of an incentive mechanism: in NetSession, peers can always obtain the content from the infrastructure, so it is not as important to discourage "freeloading". In fact, users of NetSession Interface are given an option to turn off peer content uploads permanently or temporarily in the NetSession application preferences.

3.2.5 Interaction with Edge Servers

NetSession's HTTP(S) connections to the edge servers are used not only for downloading files, they also support many other critical functions.

One important function is to ensure content integrity. File pieces can be corrupted in transit or on the peers; additionally, content can change over time, so it is important that different versions are not mixed up in the same download. Edge servers generate and maintain secure IDs of content, which are unique to each version, as well as secure hashes of the pieces of each file. The IDs and the hashes are provided to the peers, so they can validate the content they have downloaded. If a peer cannot validate a file piece, it discards the piece and does not upload it to other peers.

Another key function is authorization. Before a peer can receive content from other peers, it must authenticate to an edge server over the HTTP(S) connection; this yields an encrypted token that can be used to search for peers. This is done to prevent users from downloading from peers files they are not authorized to obtain from the infrastructure.

Finally, the HTTP(S) connections are used for configuration and reporting. A customer-defined policy is used to decide whether a particular file may be downloaded

and uploaded; in addition, various configurable options apply to each download and upload. These policies and options are securely communicated to the peers through the trusted edge-server infrastructure. NetSession also uses information from the trusted edge servers to prevent accounting attacks, where compromised or faulty peers incorrectly report downloads and uploads.

3.2.6 The NetSession Control Plane

The NetSession control plane consists of a number of globally-distributed servers that are operated by Akamai. Its main function is to coordinate between the peers. Each control plane server runs some of the following components:

Connection node (CN): The CNs are the endpoints of the TCP connection that each peer opens to the control plane when it is active; they receive and collect the usage statistics that are uploaded by the peers, and they handle queries for objects the peers wish to download. These persistent TCP connections are also used to tell peers to connect to each other in order to facilitate sharing of content; such coordination is necessary for both security reasons and to overcome NATs and firewalls.

Database node (DN): The DNs maintain a database of which objects are currently available on which peers, as well as details about the connectivity of these peers. Peers appear in the database only when a) uploads are explicitly enabled on the peer, and b) the peer currently has objects to share.

STUN: Peers periodically communicate with STUN components over UDP and TCP to determine the details of their connectivity (which are then stored in the DN databases) and to enable NAT traversal. This involves a protocol with goals similar to Rosenberg et al. (2008), but NetSession uses a custom implementation.

Monitoring nodes: Peers upload information about their operation and about problems, such as application crash reports, to these nodes. Processing their logs helps to monitor the network in real-time, to identify problems, and to troubleshoot

specific user issues during support procedures.

3.2.7 Peer Selection

When a CN receives a query for an object with peer-to-peer delivery enabled, the CN asks the DNs to identify suitable peers that currently have a copy of the requested object. The CN then returns information about these peers to the querying peer. By default, up to 40 peers are returned, and if connections to some of these peers cannot be established, additional queries are issued until a sufficient number of peer connections succeed. Peers control the number and utilization of their connections based on current resource availability.

The DN chooses peers using a locality-aware strategy at two different levels. First, when peers establish their persistent TCP connection to the control plane, they are are mapped to the closest available CN by Akamai's DNS system. When a CN queries the DN for peers for a specific object, it prefers to contact only local DNs, i.e., DNs running on machines in the same network region as the CN that performs the query. Since the same process is used when a peer registers a local copy of a file, DNs tend to have information about their *local* peers. The CN/DN system is interconnected across regions, so it is possible in principle to search for peers from any region; however, through long-term experimentation it has been found that using only local DNs in searches does not negatively impact performance.

Since the system is divided into less than 20 network regions, the first, region-based selection strategy is not sufficiently fine-grained for popular content that is available on many peers. Hence, the DNs use another level of locality-based peer selection that is based on the geolocation of each peer. Each peer belongs to multiple sets, based on its public IP address and the Autonomous System (AS) it is located in. For example, a peer can simultaneously be in a universal *World* set, a subset for a large geographical region, a subset for a smaller region, and a subset for its specific

AS.

DN selection begins with peers from the most specific set that the querying peer belongs to, and proceeds to less specific sets until enough suitable peers are found. An additional mechanism adds diversity: Occasionally, peers are selected from a less specific set, with probability proportional to the specificity of the set. Also, when a peer is selected, it is placed at the end of a peer selection list for fairness. The selection process can be modified with a set of configurable policies.

In addition to locality and file availability, the DN also takes the connectivity of the peers into account: it selects only peers that are likely to be able to establish a connection with each other, e.g., based on the type of their NAT or firewall. Due to the vast diversity in NAT implementations today, NAT hole-punching is a complex issue, and the necessary code takes up a large fraction of the NetSession codebase.

3.2.8 Robustness

The design of NetSession employs the notions of soft state and fate sharing to provide robustness against failures. At first glance, it might seem that the loss of CN or DN components could be catastrophic to NetSession. Indeed, many peers rely on each CN – over 150,000 might be connected to one simultaneously. But ultimately, all of the data about the peers that matters is held by the peers themselves. If a CN goes down, the peers that are connected to that CN simply reconnect to another one. If a DN goes down, the CNs connected to that DN send a RE-ADD message to their peers, asking them to list the files that they are storing. The CN passes these lists on to the available DNs in order to re-populate their database. In practice, failures of CN and DN nodes occur routinely, e.g., during server maintenance or during software updates. In fact, when a new CN/DN software version is released, all CNs and DNs are restarted in a short timeframe, and this does not negatively affect the service. (In the event of an unexpectedly large-scale failure, reconnections are rate-limited to

ensure a smooth recovery.) Finally, if a peer is not able to connect to any CN at all, it retrieves the content directly from the edge servers; hence, even if the entire CN and DN infrastructure were to fail, the peers would simply fall to back to retrieving content from the CDN infrastructure.

The client software version is centrally controlled by the CDN infrastructure, and peers can perform automated upgrades in the background on demand. Most of the peer population can be upgraded to a new version within one hour. The ability to perform fast software upgrades without user interaction can help to respond quickly to security or performance incidents. Download and upload performance is constantly monitored, and automated alerts are in place to notify 24/7 service of network engineers in case of a large-scale problem.

3.2.9 Best Practices

Since NetSession uses resources that are provided by the peers, it must carefully consider the users' interests. NetSession obtains consent from users through its EULA, and Akamai provides users with information about what the NetSession Interface is, and what it does. The software includes both a control panel user interface and a command line utility that enable users to determine what the software is doing, which files it is currently storing, which applications are using it, etc. These tools also allow users to turn uploading on or off, and it comes with an uninstaller.

To avoid inconveniencing users, the NetSession Interface is designed to stay in the background as much as possible. For example, peers do not proactively download content; they only share objects that their user has previously requested. Uploads are rate-limited, and each object is uploaded at most a limited number of times by each peer. Finally, peers monitor the utilization of the local network connections and throttle or pause uploads when the connections are used by other applications. While it is important for users to experience good download performance, the performance

Table 3.1: Overall statistics for our data sets.

Control plane logs:	
Time period covered	10/01 – 10/31, 2012
Log entries	4,150,989,257
Number of GUIDs	25,951,122
Control plane servers	197
Distinct URLs	4,038,894
Distinct IPs	133,690,372
Downloads initiated	12,508,764
Geolocation data:	
Distinct IPs	133,690,372
Distinct locations	34,383
Distinct domains	31,383
Distinct countries	239

of uploads is intentionally limited by using custom protocols.

Users do not benefit directly from the bandwidth they donate, but NetSession offers a number of indirect benefits; for instance, the DLM enables users to resume aborted downloads and to download from multiple sources simultaneously. Also, peer-assisted downloads require fewer resources from the infrastructure and can thus be offered at a lower price; content providers can pass on these savings, e.g., by displaying fewer ads.

3.3 Measurement Study

We now turn to the question how well NetSession, as one instance of a peer-assisted CDN, is able to deliver the potential benefits of a hybrid architecture, and how well it is able to avoid the corresponding risks. To answer this question, we have performed a measurement study that is based on a set of logs from the production system.

3.3.1 Data Set

Our logs cover the month of October 2012. At a high level, they contain information about *downloads* and information about *logins*. When a peer downloads a file from NetSession, the CN records information about the download, including the GUID of

the peer, the name and size of the file, the CP code (number identifying a specific account of a customer that is offering the file), the time the download started and ended, and the number of bytes downloaded from the infrastructure and from peers. This information is used for accounting and billing purposes. Additionally, when a peer opens a connection to the control plane, the CN records the peer's current IP address, its software version, and whether or not uploads are enabled on that peer.

To localize the peers geographically and in the network, we also obtained geolocation data from a commercial IP address geolocation service[1] about each IP address that appears in the trace. This data includes a country code, the name of a city and state, a latitude/longitude pair, a timezone, and a network provider name. The granularity of the location information varies by region; in the United States, locations are typically at the city/suburb granularity. For instance, the data set contains 218 unique locations in the state of Pennsylvania.

To protect customer privacy, the data in our logs have been anonymized by hashing the file names, IP addresses, and GUIDs. Table 3.1 shows some overall statistics for our data set.

3.3.2 Number and Location of the Peers

We begin by giving an overview of the NetSession deployment as of October 2012, to illustrate the number and geographic distribution of the peers, as well as the type of content being served. The one-month trace contains about 25.95 million distinct GUIDs. (Recall that the NetSession software chooses a GUID when it is first installed, so the number of GUIDs should correspond roughly to the number of peers.) On a typical day, between 8.75 and 10.90 million of the GUIDs connect to the control plane at least once. The system has been growing steadily over time; for comparison, a trace from October 2010 contained 14.19 million distinct GUIDs, or

[1] Name removed for anonymity purposes.

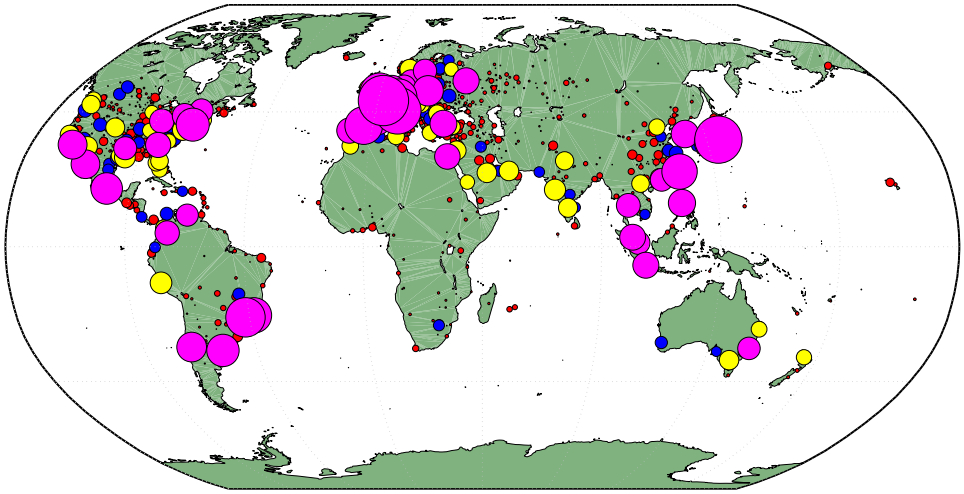


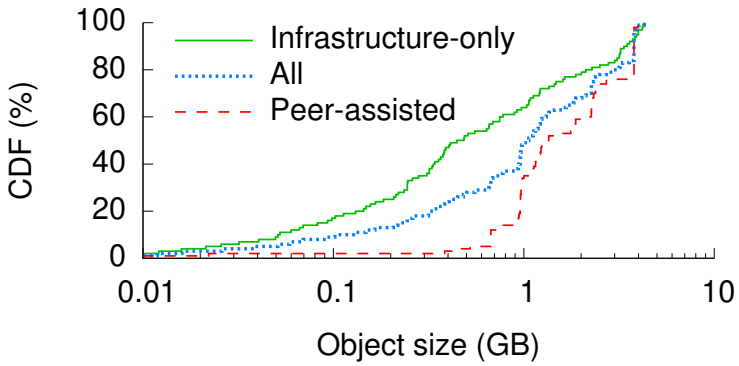
FIGURE 3.2: Global distribution of peers.

Table 3.2: Global distribution of downloads for the ten largest customers.

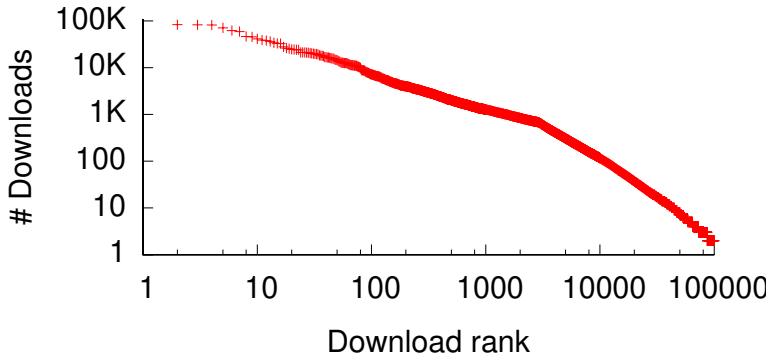
	Americas			Asia			Europe	Africa	Oceania
	US East	US West	Other	India	China	Other			
Customer A	–	–	12%	6%	6%	18%	51%	4%	3%
Customer B	2%	1%	1%	11%	–	61%	6%	17%	1%
Customer C	13%	6%	15%	1%	–	8%	55%	1%	2%
Customer D	22%	21%	6%	–	–	3%	45%	–	3%
Customer E	5%	3%	8%	2%	1%	29%	48%	2%	3%
Customer F	–	–	–	–	–	–	100%	–	–
Customer G	8%	3%	12%	2%	8%	20%	45%	2%	2%
Customer H	6%	4%	7%	4%	2%	20%	53%	2%	2%
Customer I	5%	2%	18%	–	–	15%	57%	1%	1%
Customer J	42%	24%	14%	–	–	5%	10%	1%	3%
All customers	7%	4%	11%	3%	2%	20%	46%	4%	2%

slightly more than half the number in our trace.

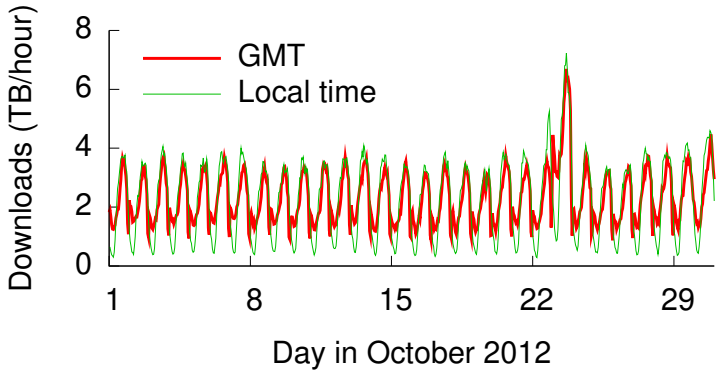
Figure 3.2 shows the global distribution of the peers as a "bubble plot": the size of each bubble corresponds to the number of peers whose first connection to the system was from that particular location. Most of the peers are located in North America (27%) and Europe (35%), but there are also sizable groups of peers in South America and Asia. Overall, we observed connections from 239 different countries across all continents, so NetSession is a truly global system.



(a) Request distrib. by object size



(b) Content popularity



(c) Bytes served over time

FIGURE 3.3: Overall workload characteristics.

3.3.3 Content Providers

Each file in NetSession is offered by a specific content provider—usually a large corporate customer. To illustrate how these customers are using the system, we selected the downloads served by the ten largest customers (here identified as Customers A

through J), we associated each download with one of ten regions, and we counted the number of downloads per customer per region.

Table 3.2 show the results. Generally, we find that roughly half of the downloads occur in Europe. However, the downloads distribution depends on customer. Customer B's content, for example, is far more popular in Asia except China and Africa. Customer J's content is mostly requested from within the United States.

3.3.4 Available Content

Overall, we observed downloads for 4,038,894 distinct objects in our trace. Figure 3.3(a) shows the distribution of requests for objects of a given size, for peer-assisted, infrastructure only, and all requests in NetSession. Peer-assisted downloads are strongly biased towards large files; 73% of peer-assisted requests are for objects larger than 500MB. Because the benefits of peer assist are most pronounced for large objects, content providers tend to enable it on such objects. The object popularity distribution and temporal request pattern in NetSession's workload are shown in Figure 3.3(b) and (c). As expected, the former shows the nearly ubiquitous power law, while the latter shows the usual diurnal patterns.

3.4 Benefits

Next, we focus on three questions that can be answered quantitatively based on our data, namely: 1) How well does peer assist work? 2) Does peer assist affect performance and reliability? And, 3) does peer assist help improve the CDN's global coverage?

3.4.1 How Well Does Peer Assist Work?

Recall that NetSession peers are not required to contribute bandwidth: users are free to disable uploads to peers by changing their preferences in the NetSession GUI.

Table 3.3: Observed changes to the setting that enables content uploads.

Uploads initially...	Nodes	Number of changes		
		0	1	$\geqslant 2$
Disabled	15,913,255	99.96%	0.03%	0.01%
Enabled	7,395,867	98.11%	1.80%	0.09%

Table 3.4: Fraction of peers that have content uploads enabled, classified by the customer from whom they have downloaded their first file.

Customer	A	B	C	D	E	F	G	H	I	J
P2P (%)	<1	20	2	94	2	45	47	<1	91	<1

This is a major difference to pure peer-to-peer systems like BitTorrent, which include incentive mechanisms like tit-for-tat to encourage uploading. In the literature, it is often assumed that users are "rational" and will avoid uploading if they don't benefit from it (Cohen, 2003; Piatek et al., 2007); it is therefore natural to ask whether NetSession peers are willing to contribute any bandwidth at all.

Do the peers contribute resources? To answer this question, we used the login records in our data set to determine the fraction of peers that have uploads enabled. Since the NetSession binary is available in two versions, one with uploads initially enabled and one with uploads initially disabled (as chosen by the content provider who bundles the binary), we also check whether users changed this setting between logins, and if so, how often.

Tables 3.3 and 3.4 show our results. About 31% of the peers have uploading enabled, but the setting is rarely changed—more than 99% of the peers keep their initial setting throughout our trace. As Table 3.4 shows, the initial setting depends on the content provider from who the user first downloaded the binary. Most users simply stick with whatever the default is. This tendency is well known in UI design (Mackay, 1991), but it also suggests that users either don't care enough about

the uploads to change the setting or are not aware of the choice, despite its mention in the NetSession user agreement.

Uploading in peer-to-peer CDNs carries the risk of legal exposure and/or degraded network performance (Sirivianos et al., 2007b). As a peer-assisted CDN, NetSession can avoid the first problem because its content is centrally controlled and vetted, and its back-off mechanism (Section 3.2.9) can avoid the second problem by consuming bandwidth only when the connection is idle. We speculate that this is part of the reason why most users don't seem to turn off peer uploads when they are enabled initially.

How much can be offloaded to the peers? As a peer-assisted CDN, NetSession can offload some of the bandwidth needed to satisfy the download requests to the peers. Content providers can control on a per-file basis whether or not peer-to-peer downloads are allowed. In our trace, we found that peer-to-peer downloads were enabled for only 1.7% of the files, but these downloads accounted for 57.4% of the downloaded bytes overall.

The key quantity of interest is the *peer efficiency* of the system, i.e., the fraction of bytes that are downloaded from the peers. In our trace, the average peer efficiency for peer assisted downloads was 71.4%. This is a very good result, given that, even in a peer-assisted download, NetSession never *exclusively* downloads from the peers; there is always at least one connection to the infrastructure, to guarantee progress independent of the peers.

3.4.2 Does Peer Assist Reduce Performance?

Are peer assisted downloads slower? NetSession relies on peer downloads, which are limited by the peer reliability and the upstream bandwidth of broadband access networks, to which many of NetSession's peers are connected. This bandwidth is typically much smaller than the downstream bandwidth (Dischinger et al., 2007).

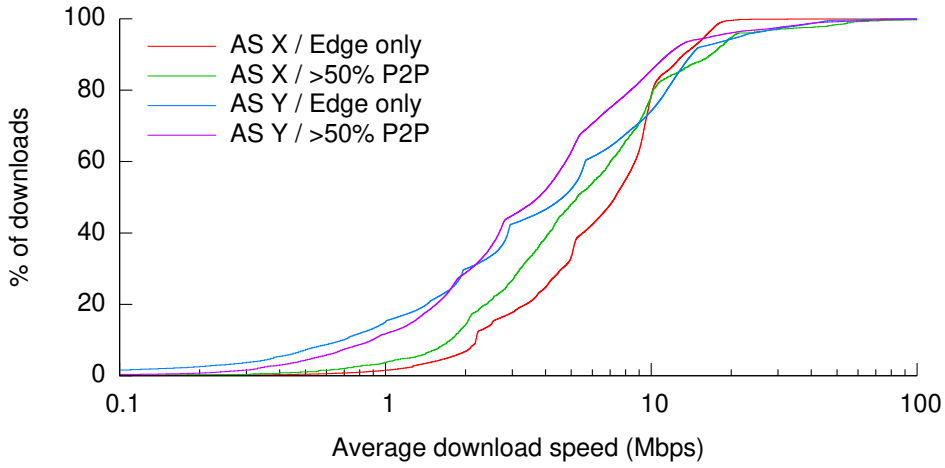


FIGURE 3.4: Comparison of edge-only downloads and peer-assisted downloads.

Hence, it is natural to ask how the performance of peer-assisted downloads compares to those that are served by the infrastructure.

Figure 3.4 makes this comparison for downloads from the two networks with the most downloads, AS X and AS Y. We identified all downloads from these networks where either a) all the bytes came from the edge servers, or b) at least 50% of the bytes came from peers. We then averaged the speed of each download across its entire length; the figure shows the results as a CDF.

We find that, although the peer-assisted downloads are somewhat slower, the speed is still quite high, with most downloads occurring at rates of multiple Mbps. When comparing the performance across networks, we find that the biggest differences between peer assisted and infrastructure download speeds occur in the networks with the highest link bandwidths. One possible explanation is the high asymmetry of most high-speed broadband links (fast downstream and slow upstream), which would limit the available bandwidth from nearby peers.

How many peers are needed for good performance? Achieving good peer efficiency in a peer-assisted download requires a sufficient number of peers who have a copy of the requested file. We now ask how many copies are needed to achieve a

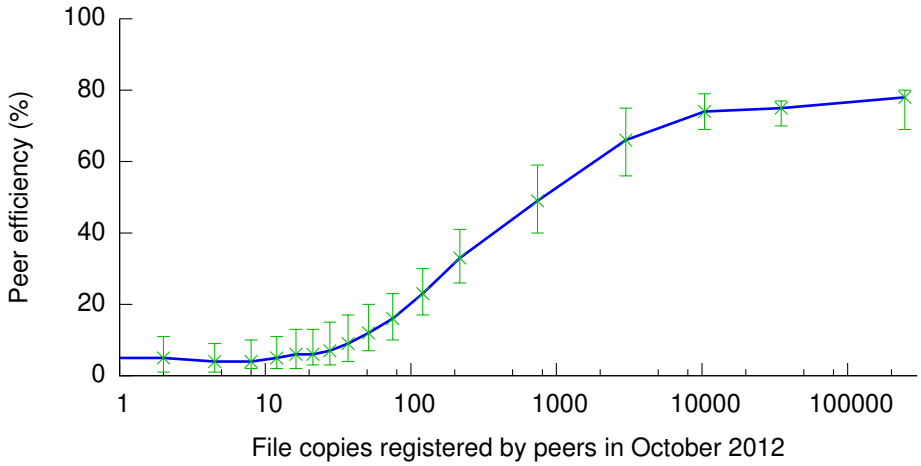


FIGURE 3.5: Number of file copies vs. peer efficiency

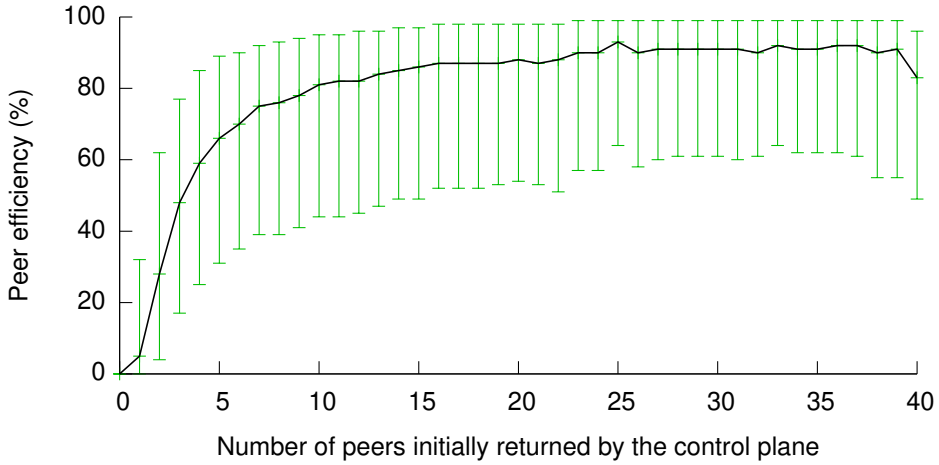


FIGURE 3.6: P2P efficiency vs. number of peers.

good peer efficiency.

NetSession does not use predictive caching—i.e., a peer only downloads a file when it is requested by the local user. Once a file has been downloaded, the peer keeps it in a local cache for a certain amount of time and informs the control plane that it is willing to upload this file to other peers (if uploading is enabled). This creates entries in the DN log, and we counted these entries for each file to estimate how many copies of that file were available. We also calculated the average peer

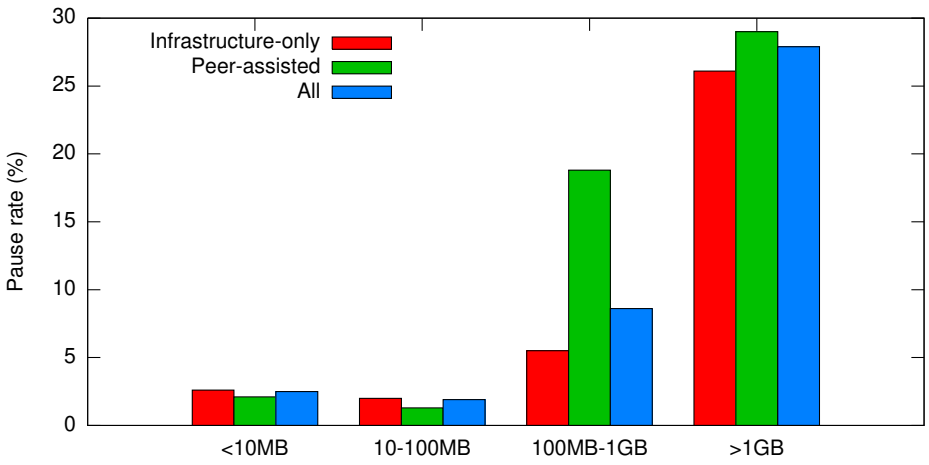


FIGURE 3.7: Downloads of larger files are terminated more often.

efficiency for each file.

Figure 3.5 shows how the average peer efficiency varied with the number of copies (the error bars show the 20th and 80th percentile). We found that, with less than 50 available copies, peer efficiency was below 10%, but rose rapidly after that, and reached 80% for approximately 10,000 copies.

Another way to look at this question is to ask how many *peers* need to assist in a given download to achieve the highest efficiency. To answer this question, we grouped the downloads by the number of peers that the DN initially suggested to the downloading peer, and we computed the average peer efficiency for each group. Figure 3.6 shows our results. We find that 80% peer efficiency is generally reached with about 25–30 peers.

Are peer-assisted downloads less reliable? To answer this question, we examined the eventual outcome of each download initiated in our trace. The logs can show three kinds of outcomes: a download can complete, it can fail, or it can be aborted/paused by the user and never resumed. When a download fails, the log also shows a specific cause; we divided these into system-related causes (e.g., too many corrupted content blocks) and other causes (e.g., the user's disk is full).

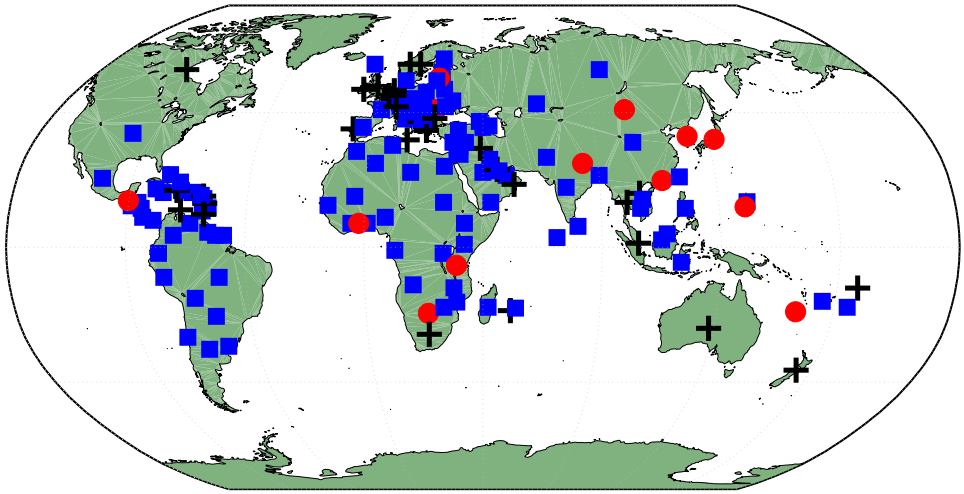


FIGURE 3.8: Peer contributions in different regions.

We find that 94% of the infrastructure-only downloads eventually complete, but only 92% of the peer-assisted downloads. At first glance, this discrepancy suggests that peer-assisted downloads are less reliable. However, the rate of system-related failures is very small (0.1% for infrastructure-only versus 0.2% for peer-assisted), but infrastructure-only downloads are paused or terminated less often than peer-assisted ones (3% versus 8%).

A closer look reveals that the termination rate increases with the length of the download. While the download *speed* for infrastructure-only and peer-assisted downloads is approximately the same, the latter are typically used for larger files (see Figure 3.3(a) earlier), which naturally take longer to download. As Figure 3.7 shows, larger downloads are terminated more often, regardless of whether or not they involve peers—hence the discrepancy.

3.4.3 Do Peers Help Improve Global Coverage?

In theory, a hybrid CDN has another advantage over its infrastructure-based counterparts: it may be easier to obtain a globally distributed population of peers than to establish a truly global infrastructure. Thus, it should be possible for a hybrid

system to provide better service for customers in under-served regions, where the closest infrastructure nodes are far away, but peers may be nearby.

To test whether this is the case for NetSession, we aggregated the completed downloads on a per-country basis, and we counted the number of bytes served by the infrastructure and by the peers. We then classified each country into one of three groups, which are shown as colored dots in Figure 3.8: countries where the infrastructure serves bytes more than the peers (circle), less than 50% of the peers (square), or between 50% and 100% of the peers (plus). Since not all content providers use peer-assisted downloads, and the popularity of the providers varies between regions, we show results for one typical, P2P-enabled provider here.

We find that, for NetSession, the picture is mixed: although the peers tend to contribute more in some regions, such as Africa and South America, the contributions do not vary much overall. We suspect that this is because NetSession relies on edge servers from Akamai's main CDN, which already has very good coverage around the globe.

Another potential benefit of a large peer population is that downloading peers might find a copy of the requested content within their local network, e.g., in a corporate LAN. In October 2012 this case appears to have been rare, but this could change, e.g., when NetSession is used to distribute large software updates.

3.4.4 Summary

NetSession demonstrates that peer-assisted CDNs can indeed deliver the key benefits of both peer-to-peer and infrastructure-based CDNs: they can offload a considerable fraction of the traffic to the peers, without a significant loss of speed or reliability. NetSession's 80% peer efficiency depends on many factors that may be different in other systems, but it provides a lower bound on what peer-assisted CDNs can achieve in a large-scale commercial deployment.

3.5 Conclusion

This chapter presented a measurement study of NetSession, a large commercial CDN to determine how well a practical system can deliver the benefits and avoid the risks of hybrid architectures.

Our results show that NetSession is able to deliver the key benefits of a hybrid architecture: it can offload a high fraction (70–80%) of the traffic to peers, but can *also* offer good performance and high reliability.

Overall, our findings suggest that a hybrid architecture is an attractive design point for a CDN. The infrastructure and the peers can deliver many of their key benefits, and they can complement one another to avoid many of their key weaknesses. NetSession's performance shows what is possible in this space, although it only represents a lower bound; other systems may be able to do even better.

4

CDN-ISP Collaboration

4.1 Introduction

Economics, especially cost reduction, is a main concern today in content delivery as Internet traffic grows at a annual rate of 30% (Networking and Index., 2013). Moreover, commercial-grade applications delivered by CDNs often have requirements in terms of end-to-end delay (Krishnan et al., 2009). Faster and more reliable content delivery results in higher revenues for e-commerce and streaming applications (Leighton, 2009; Nygren et al., 2010) as well as user engagement (Dobrian et al., 2011). Despite the significant efforts by CDNs to improve content delivery performance, end-user mis-location and the limited view of network bottlenecks are major obstacles to improve end-user performance.

Content Delivery Cost: CDNs strive to minimize the overall cost of delivering voluminous content traffic to end-users. To that end, their assignment strategy is mainly driven by economic aspects such as bandwidth or energy cost (Liu et al., 2012; Qureshi et al., 2009). While a CDNs will try to assign end-users in such a way that the server can deliver reasonable performance, this does not always result in end-

users being assigned to the server able to deliver the best performance. Moreover, the intense competition in the content delivery market has led to diminishing returns of delivering traffic to end-users. Part of the delivery cost is also the maintenance and constantly upgrade of hardware and peering capacity in many locations (Nygren et al., 2010).

End-user Mis-location: DNS requests received by the CDN name servers originate from the DNS resolver of the end-user, not from the end-user themselves. The assignment of end-users to servers is therefore based on the assumption that end-users are close to the used DNS resolvers. Recent studies have shown that in many cases this assumption does not hold (Liu et al., 2012; Qureshi et al., 2009). As a result, the end-user is mis-located and the server assignment is not optimal. As a response, DNS extensions have been proposed to include the end-user IP information (Otto et al., 2012).

Network Bottlenecks: Despite their efforts to discover the paths between the end-users and their servers to predict performances (Nygren et al., 2010; Krishnan et al., 2009), CDNs have limited information about the actual network conditions. Tracking the ever changing network conditions, i.e., through active measurements and end-user reports, incurs an extensive overhead for the CDN without a guarantee of performance improvements for the end-user. Without sufficient information about the network paths between the CDN servers and the end-user, a user assignment performed by the CDN can lead to additional load on existing network bottlenecks, or even create new ones.

Given the trends regarding server resources and increasing user demand, content delivery systems have to address two fundamental problems. The first is end-user to server assignment problem, i.e., how to assign users to the appropriate servers. The key enabler for addressing this problem is *informed user-server assignment* or in short *user-server assignment*. It allows a CDN to receive recommendations from a network

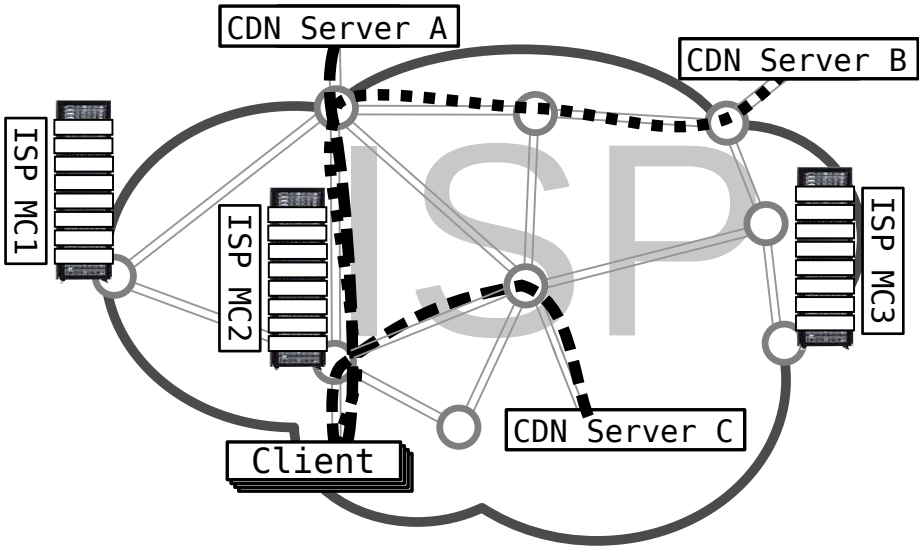


FIGURE 4.1: **Informed User-Server Assignment:** Assigning a user to an appropriate CDN server among those available (A, B, C), yields better end-user performance and traffic engineering. **In-network Server Allocation:** A joint in-network server allocation approach allows the CDN to expand its footprint using additional and more suitable locations (e.g., microdatacenters MC1, MC2, MC3) inside the network to cope with volatile demand. User-server assignment can also be used for redirecting users to the new servers.

operator, i.e., a server ranking based on performance criteria mutually agreed upon by the ISP and CDN. The CDN can utilize these recommendations when making its final decision regarding end-user to server assignments. This enabler takes full advantage of server and path diversity, which a CDN has difficulty exploring on its own. Moreover, it allows the coordination of CDNs, content providers, and ISPs at the scale of seconds or even per request. Any type of CDN can benefit from this enabler including ISP-operated CDNs. The advantage of our enabler in comparison with other CDN-ISP (DiPalantino and Johari, 2009; Jiang et al., 2009a) and ISP-P2P (Xie et al., 2008) cooperation schemes is that no routing changes are needed.

The second is server allocation problem, i.e., where to place the servers and content. The key enabler is *in-network server allocation*, or in short *server allocation*, where the placement of servers within a network is coordinated between CDNs,

ISPs, and content providers. This enabler provides an additional degree of freedom to the CDN to scale-up or shrink the footprint on demand and thus allows it to deliver content from additional locations inside the network. Major improvements in content delivery are also possible due to the fact that the servers are placed in a way that better server the volatile user demand. The application of this enabler is two-fold. One, it helps the CDN in selecting the locations and sizes of server clusters in an ISP when it is shipping its own hardware. The second application is suitable for more agile allocation of servers in cloud environments. Multiple instances of virtual servers running the CDN software are installed on physical servers owned by the ISP. As before, the CDN and the ISP can jointly decide on the locations and the number of servers. A big advantage of using virtual machines is that the time scale of server allocation can be reduced to hours or even minutes depending on the requirements of the application and the availability of physical resources in the network. User-server assignment can also be used for redirecting users to the new servers. We provide the high-level intuition for both enablers in Figure 4.1.

Until now, both problems have been tackled in a one-sided fashion by CDNs. We believe that to improve content delivery, accurate and up-to-date information should be used during the server selection by the CDN. This also eliminates the need for CDNs to perform cumbersome and sometimes inaccurate measurements to infer the changing conditions within the ISP. We also believe that the final decision must still be made by the CDN. In this chapter , we argue that the above enablers (a) are necessary to enable new CDN architectures and take advantage of server virtualization technology, (b) allow fruitful coordination between all involved parties, including CDNs, CPs, and ISPs in light of the new CDN-ISP alliances, (c) enable the launch of new applications jointly by CDNs and ISPs, and (d) can significantly improve content delivery performance. Such performance improvements are crucial as reductions in user transaction time increase revenues by significant margins (Kohavi

et al., 2007).

4.2 Data Set

To evaluate the potential gains from CDN-ISP collaboration, we use traces from the largest commercial CDN and a European tier-1 ISP.

Commercial CDN Dataset: The dataset from the CDN covers a two-week period from 7th to 21st March 2011. All entries in the log we use relate to the tier-1 ISP. This means that either the server or the end-user is using an IP address that belongs to the address space of the tier-1 ISP. The CDN operates a number of server clusters located inside the ISP and uses IPs in the IP address space of the ISP. The log contains detailed records of approximately 62 million sampled (uniformly at random) valid TCP connections between the CDN's servers and end-users.

For each reported connection, it contains the time it was recorded, the server IP address, the cluster the server belongs to, the anonymized client IP address and various connection statistics such as bytes sent/received, duration, packet count and RTT. The CDN operates a number of services, utilizing the same infrastructure, such as dynamic and static web pages delivery, cloud acceleration and video streaming.

ISP Dataset: The ISP dataset contains two parts. First, detailed network information about the tier-1 ISP, including the backbone topology, with interfaces and link annotations such as routing weights, as well as nominal bandwidth and delay. It also contains the full internal routing table which includes all subnets propagated inside the ISP either from internal routers or learned from peerings. The ISP operates more than 650 routers in about 400 locations (PoPs), and 30 peering points worldwide. We analyzed more than 5 million routing entries to derive a detailed ISP network view.

The second part of the ISP dataset is an anonymized packet-level trace of residential DSL connections. Our monitor, using Endace monitoring cards (Cleary et al.,

2000), observes the traffic of more than 20,000 DSL lines to the Internet. We capture HTTP and DNS traffic using the Bro IDS (Paxson, 1999). We observe 720 million DNS messages and more than 1 billion HTTP requests involving about 1.4 million unique hostnames. Analyzing the HTTP traffic in detail reveals that a large fraction it is due to a small number of CDNs, including the considered CDN, hyper-giants and one-click-hosters (Labovitz et al., 2010; Gerber and Doverspike, 2011; Maier et al., 2009) and that more than 65% of the traffic volume is due to HTTP.

To derive the needed traffic matrices, on an origin-destination flow granularity, we compute from the DSL traces (on a 10-minute time bin granularity) the demands for the captured location in the ISP network. This demand is then scaled according to the load imposed by users of the CDN to the other locations in the ISP network. For CDNs without available connection logs, we first identify their infrastructure locations using the infrastructure aggregation approach as proposed by Poese et al. (Poese et al., 2010) and then scale the traffic demands according to the available CDN connection logs.

4.3 Evaluation

In this section, we use our observations on the traffic and deployment of the large commercial CDN inside the tier-1 ISP to quantify the potential benefits of CDN-ISP collaboration. In Figure 4.2, we plot the normalized traffic (in log scale) from CDN clusters over time. We classify the traffic into three categories: *a)* from CDN servers inside the ISP to end-users inside the ISP (annotated ISP → ISP), *b)* from servers outside the ISP to end-users inside the ISP (annotated outside → ISP), and *c)* from CDN servers inside the ISP to end-users outside the ISP (annotated ISP → outside).

We observe the typical diurnal traffic pattern and a daily stability of the traffic pattern. Over the two week measurement period, 45.6% of the traffic belongs to the ISP → ISP category. 16.8% of the traffic belongs to the outside → ISP category.

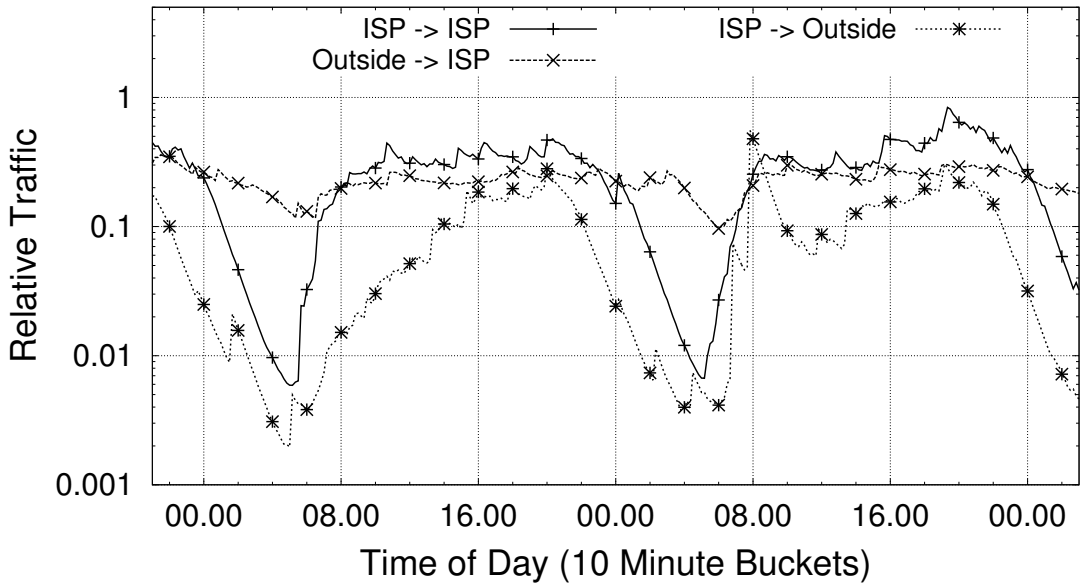


FIGURE 4.2: Activity of CDN in two days.

During peak hours, outside → ISP traffic can grow up to 40%. Finally, 37.6% of the traffic is served by inside clusters to outside end-users. Our first important observation is that a significant fraction of the CDN traffic is served from servers outside the ISP despite the presence of many servers inside the ISP that would be able to serve this traffic.

Figure 4.3 shows the re-allocation of traffic that would be possible using user-server assignment. Each full bar shows the fraction of traffic currently traversing a given number of router hops within the ISP network. In this evaluation, we only consider the end-users inside the ISP. The bar labelled "N/A" is the traffic of the outside → ISP category. The different shaded regions in each bar correspond to the different router hop distances after re-allocation of the traffic. Almost half of the traffic currently experiencing 3 hops can be served from a closer-by server. Overall, a significant fraction of the traffic can be mapped to closer servers inside the ISP. Note that the tiny amount of traffic for router hop count 0 and 1 is due to the topology

FIGURE 4.3: Potential hop reduction.

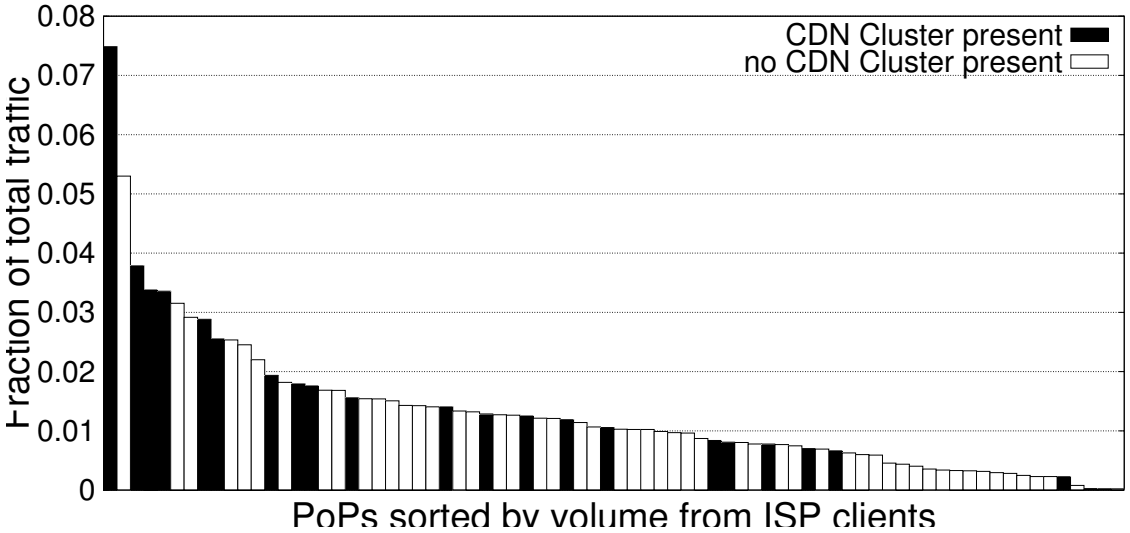


FIGURE 4.4: Traffic demand by ISP network position.

design of the ISP network: either the traffic stays within a PoP or it has to traverse at least two links to reach another PoP.

In Figure 4.4, we show the traffic demand towards the CDN generated by each PoP. We observe that some PoPs originate high demand while others have limited

demand, if any. Manual inspection reveals that some of the PoPs with high demand cannot be served by a close-by CDN server, while other low demand PoPs have a cluster near by. Variations in the demand over time exhibit even more significant mismatches between demand and CDN locations. With such a time-varying demand and the timescales at which CDN deployments take place today, such mismatches should be expected. We conclude that there are ample opportunities for CDNs to benefit from collaboration with ISPs to re-arrange or expand their footprint.

4.4 Conclusion

Motivated by recent CDN and ISP alliances we revisit the problem of CDN-ISP collaboration from a systems perspective. We identify two major enablers, namely informed user-server assignment and in-network server allocation. Today, there is no system to support CDN-ISP collaboration. To that end, we perform the first-of-its-kind evaluation of CDN-ISP collaboration based on traces from the largest commercial CDN and a large tier-1 ISP. We report on the benefits for CDNs, ISPs and end-users. Our results show that CDN-ISP collaboration leads to a win-win situation with regards to the deployment and operation of servers within the network, and significantly improves end-user performance.

5

Cost-Efficient Caching

5.1 Introduction

The idea of information- or data-centric networking is driven by the evolution of Internet traffic workloads. The insight here is that a user's intent is to fetch some *data object* rather connect to a specific host for data exchange. By decoupling the data a user wants to access from how/where the data is delivered, ICN promises several natural benefits. These include: lower response time via pervasive caching and nearest-replica routing; intrinsic content integrity without external network-level indicators (e.g., HTTPS); simplified traffic engineering capabilities; and better support for *mobility* (Ahlgren et al., 2012; Ghodsi et al., 2011).

Unfortunately, these benefits come at a non-trivial cost. Many ICN proposals envision a forklift upgrade to the entire network infrastructure requiring all end hosts and network routers to support ICN as a first-order primitive. This entails adding content stores to routers and supporting routing on content names as opposed to IP addresses. Given that these architectures mandate wholesale changes to the network infrastructure, it is only natural to ask if this complexity is worthwhile.

In order to address this question, we begin by breaking down the potential benefits of ICN into two categories. The first class of *quantitative* benefits—lower response time and simplified traffic engineering—arise from a combination of a pervasive caching infrastructure coupled with intelligent nearest-replica routing. The second class of *qualitative* benefits stem from the ability to name content and verify content integrity through the naming scheme (e.g., self-certified names or digital signatures).

Having thus dissected the potential benefits, we first focus on the quantitative benefits. Rather than commit to any specific ICN realization, we want to analyze a wide spectrum of caching architectures that differ in two key dimensions: *placement* (e.g., edge caches vs. pervasive caching) and *routing* (e.g., shortest path to origin servers vs. nearest replica routing). Using a combination of trace-driven analysis based on request logs from three CDN clusters and large-scale simulations, we find that:

1. On realistic request traces, the maximum performance gap between a simple *edge-based caching* architecture and a full-fledged ICN architecture (i.e., with pervasive caches and nearest-replica routing) is **at most 9%** with respect to response time, network congestion, and origin server load.

2. Nearest-replica routing adds marginal (**2%**) value over simple shortest path routing in ICN.

3. Using sensitivity analysis on a range of configuration parameters, we find that the *best* improvement that ICN can provide is **17%** over the simple edge-caching architecture.

4. Simple extensions to the edge cache reduces even this best-case performance gap to less than **5%**.

Note that we are not arguing that caching is not useful or that there is no redundancy in typical workloads. Our observation here is that exploiting the benefits of cacheable workloads is far easier than we imagined. In some sense, the quantitative benefits of caching largely arise from the fact that *some* cache exists; pervasive caching and nearest-replica add little value for the types of heavy-tailed workloads we expect to see.

Motivated by these findings, we analyze whether the remaining qualitative benefits can be achieved without router-level support. Somewhat surprisingly, we show that many of these benefits can be achieved using techniques that are already well known in the content distribution community.

5.2 Background and Motivation

In this section, we begin with a brief overview of the common themes underlying different ICN proposals (Jacobson et al., 2009, 2010; Aranda et al., 2010; SAIL, 2010). Then, we use real request logs to motivate the need to revisit some of the assumptions about pervasive caching and nearest-replica routing.

5.2.1 ICN principles and benefits

While ICN proposals vary in the terminology, physical implementations, and APIs to users and network operators, we identify four main themes underlying all proposals:

1. *Decouple names from locations:* Network applications and protocols are rearchitected so that the communication abstraction is based on content lookup and transfer in contrast to today's host-centric abstractions.

2. *Pervasive caching:* In the limit, every network router also acts as a content cache. This means that in addition to traditional forwarding responsibilities network routers also serve requests for content that is locally cached.

3. *Nearest replica routing:* Network routing is modified to be based on the content name rather than hosts so that requests are routed to the nearest copy of the content. (In the worst case, this is the origin server hosting the content.)

4. *Binding names to intent:* An object's name is intrinsically bound to the intent of the content publisher and the consumer. This binding helps users (and routers) to check the integrity and the provenance of the data without external indicators.

The proposals differ largely in implementation details such as specific API they expose (Ghodsi et al., 2011), which is not the focus of this dissertation; rather we want to analyze the benefits arising from these principles.

For completeness, we enumerate the perceived benefits of ICN that have been argued in prior work (Ghodsi et al., 2011; Ahlgren et al., 2012).

Lower response latency: A pervasive caching infrastructure naturally implies that the requests do not need to traverse the entire network toward the origin server. In some sense, these democratize the benefits that today's commercial CDN infrastructures provide by caching content and serving it on behalf of their customers.

Simplified traffic engineering: An additional perceived benefit of the ubiquitous caching infrastructure is that it also helps network operators by automatically eliminating content hotspots, which simplifies the traffic engineering logic necessary to balance network load.

Security: The third key benefit of content-centric solutions is that by using content as a first-class citizen, it intrinsically binds the user's intent to the eventual data being delivered without having to rely on external confirmation of the provenance or authenticity of the data.

Support for mobility: Because all network requests are routed by content rather than hosts, ICN also makes it easier to support mobile clients because traditional

Table 5.1: Feature-Benefit Matrix for ICN: the ✓shows the key features of ICN that contribute to each perceived benefit.

Benefit	Feature			
	Decoupling names from location	Pervasive Caching	Nearest-replica routing	Binding
Latency		✓	✓	
Traffic Engg		✓	✓	
Mobility	✓		✓	
Ad hoc mode	✓		✓	
Security	✓			✓

problems with handoffs, retransmissions, etc., simply go away.

Ad hoc modes: Another benefit of ICN is the ability of two nodes to communicate and share content without needing any infrastructure support. Imagine a user wanting to share a photo between a mobile phone and a laptop; today we have unwieldy workarounds via cloud-based services (TRA, 2013). Further imagine that they are in an airplane without a wireless network; in this case they cannot share the content because they do not have IP working.

Others: There are other perceived benefits such as DDoS resilience (Gasti et al., 2012) and disruption tolerance that are less well explored in the ICN community. These appear to be instantiations or combinations of the above benefits. For instance, disruption tolerance seems to be a combination of the mobility support and ad hoc mode. Similarly, DDoS resilience seems to be due to avoiding content hotspots via universal caching.

Table 5.1 summarizes the benefits and the ICN principles contributing to each perceived benefit. We can see that the *quantitative* performance benefits—low latency and traffic engineering—essentially arise as a result of the pervasive caching and nearest-replica routing infrastructure envisioned by ICN solutions. Unsurprisingly, we find that this is also the topic that has received the greatest attention in the

ICN community. The second class of *qualitative* benefits such as mobility, security, and support for ad hoc modes are rooted in the naming-related aspects of ICN (and to a lesser degree from the nearest-replica routing).

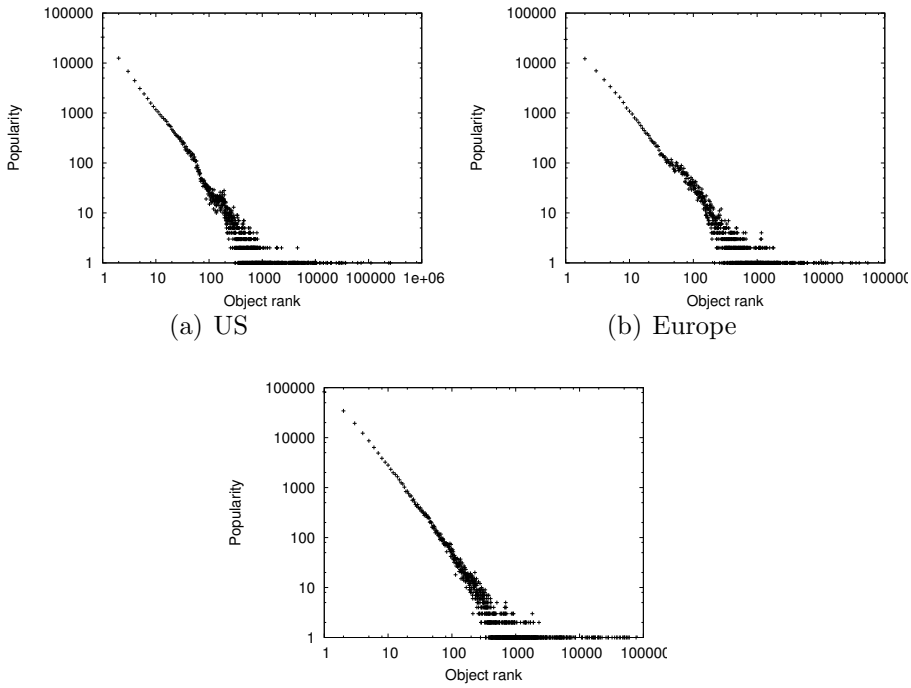
5.2.2 Motivation: Heavy-tailed workloads

Many measurement studies have observed heavy-tailed or Zipf distributions (i.e., the i^{th} popular object has request probability $\frac{1}{i^{\alpha}}$ for some $\alpha > 0$) in request popularities (e.g., Breslau et al. (1999); Gill et al. (2007)). In this section, we use request logs collected from 3 multiple vantage points of a large CDN deployment to confirm the observations of heavy-tailed behaviors with recent workloads.

Dataset: The CDN serves a diverse workload spanning diverse content types: regular text, images, multimedia, software binaries, and other miscellaneous content. We focus on the logs from one day's worth of requests from three geographically diverse locations; Table 5.2 summarizes the key parameters across the different locations. Each entry of the logs consists of four relevant fields: an anonymized client IP, anonymized request URL, the size of the object, and whether the object was served locally or forwarded to a remote location.

Figure 5.1 visually confirms that request popularity is heavy tailed and close to a Zipfian distribution; the curve is close to linear on a log-log plot. The specific exponents and intercept of the linear fit do vary slightly across locations and content types but the main takeaway is that object requests are still reasonably approximated by heavy-tailed Zipfian distributions. Table 5.2 also summarizes the parameters of the Zipf-fit that we observe across the different curves; we use these parameters to guide our simulation settings in this study.

Why does Zipf matter?: Anecdotal evidence suggests that given Zipf workloads, having several layers of caching or intelligent cooperative caching provides limited improvements (Beaver et al., 2010; Wolman et al., 1999). To understand this better,



(a) US (b) Europe

(c) Asia

FIGURE 5.1: Request popularity distribution across different geographical locations. While the specific exponent parameters in the models vary slightly across the different locations we can visually see that the popularity distribution is Zipfian.

Table 5.2: Analysis of requests from three CDN cache clusters from different geographical regions.

Location	Requests	Zipf parameter
Asia	480K	1.06
Europe	503K	0.98
USA	965K	0.91

we begin with a simple analysis on a tree topology. A tree is small enough to be amenable to such analysis and at the same time it is instructive because from the view of a content origin server, the distribution topology is effectively a tree. We use an analytical optimization model to reason about the *optimal* cache management scheme—the best static placement of objects and routing of requests across the tree given a Zipfian workload. (We do not show details due to space constraints.

The high-level idea here is formulate and solve the problem of deciding where to cache specific objects and how to assign requests to different caches to minimize the expected latency as an integer linear program.)

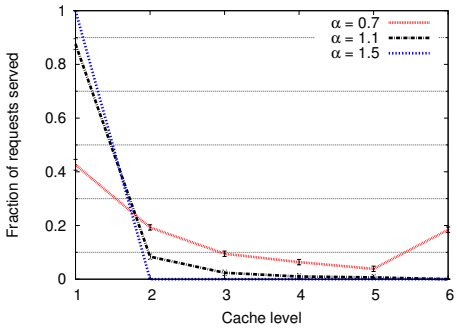


FIGURE 5.2: Looking at the optimal utility of different cache levels with a simplified optimization model on a binary tree with 5 levels. Level 6 here is the origin server to which requests are sent on cache misses.

Figure 5.2 shows the total number of requests served at different levels in the tree for request streams drawn from different Zipfian distributions. Here, level 6 denotes the origin server. We see that the intermediary levels of the tree (i.e., levels 2–5) add little value beyond caching at the edge or satisfying the request at the origin. Consider the setting with $\alpha = 0.7$. In this case, the expected number of hops that a request traverses is $0.4 \times 1 + \ldots + 0.18 \times 6 \approx 3$. Now, let us look at an extreme scenario where we have no caches at the intermediate levels; i.e., all of the requests currently assigned to levels 2–5 will now be served at the origin. In this case, the expected number of hops will be $0.4 \times 1 + 0.6 \times 6 = 4$. In other words, the improvement attributed to universal caching on a tree is only 25%. Note that this is unfair to the edge caching approach since it has $2\times$ smaller total cache capacity.

Together, these observations suggest that even if we were to enable cooperation between different caches or try to intelligently reroute requests to a replica where these other objects were available, the maximum benefit we would obtain is around 25%. The above reconfirmation that request workloads in the wild are Zipf and our

simple tree-based intuition motivates us evaluate to what extent pervasive caching and nearest-replica lookup are really necessary to achieve the quantiative benefits of ICN.

5.3 Design Space for Caching

The measurements and simplified analysis from the previous section raise the question of whether the pervasive caching and nearest-replica routing are strictly necessary. Given the diversity of ICN proposals, we want to avoid tightly coupling our analysis to any specific architecture. To this end, we consider a broad design space of caching infrastructures characterized by two high-level dimensions:

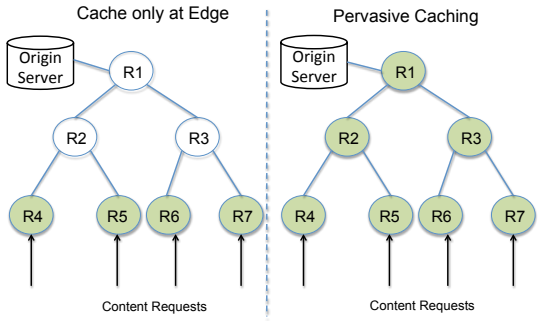


FIGURE 5.3: Example of two cache placement strategies: caches placed at select network locations such as edge of the network or pervasively throughout the network. The shaded nodes are routers augmented with content caches while the others are normal routers.

1. Cache placement: The first dimension of interest is where caches are located in the network. From the perspective of the origin server that is interested in serving content to users, the network looks like a tree of routers/caches. Figure 5.3 depicts two possible strategies in this distribution tree from the origin server. At one extreme, every network router is also a content cache. Alternatively, we can envision caches deployed close to the network edge. We can also consider intermediate placement solutions; e.g., due to economic constraints operators

may only install caches at locations that serve a sufficiently large population (Li et al., 2013). A related question here is the issue of cache provisioning; i.e., the compute and storage capacity of the various caches. For instance, we can consider a network where all caches have the same capacity or make the caches proportionally higher for nodes serving larger populations.

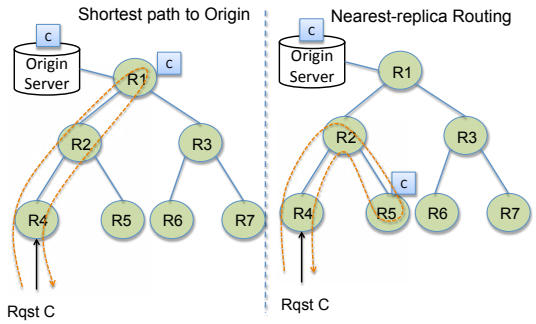


FIGURE 5.4: Example of two request routing strategies: requests are routed along the shortest path to the origin server and served from some available content cache along that path or the requests are routed to the nearest cached copy (e.g., ICN).

2. Request routing: An orthogonal dimension to cache placement is how content requests are routed through the network. As representative samples, we consider two points in Figure 5.4. In this example, a request for the object C arrives at node $R4$. The origin server and possibly some other nodes have copies of C. In the first case, a request is routed along the tree toward the origin server until it finds a node with the desired content. In the second case, we assume that the network routes the request based on the name toward the closest replica. We can also consider intermediate strategies. For instance, we can consider cooperative caching within a small search scope to lookup nearby nodes and reverting to shortest-path routing toward the origin if these lookups fail.

In this work, we are less concerned with the specific route discovery protocols to populate content routing tables (Jacobson et al., 2009) or the viability of name-

based lookup in high-speed routers (Perino and Varvello, 2011). Since our goal is to evaluate the *potential benefit* of nearest-replica routing and pervasive caching, we conservatively assume that routing and name-based lookup have zero cost.

5.4 Benefits of Caching

In this section, we use trace-driven simulations to analyze the relative performance of different caching architectures with respect to three key metrics: (1) *response latency*; (2) *network congestion*; and (3) *server load*.

5.4.1 Setup

We use realistic PoP-level network topologies from educational backbones and Rocketfuel (Spring et al., 2004). From each PoP-level topology (*core network*), we create its corresponding router-level topology by considering each PoP as the root of a complete k-ary tree. We refer to this as the *access tree*. Figure 5.5 shows an example network topology with four PoPs. We annotate each PoP with the population of its associated metro region and assume that the requests at each PoP are proportional to its population.

Requests arrive at the leaves of each access tree. We assume that requests follow a (discrete) Zipf distribution with some value of α. Within each PoP, the requests arrive uniformly at random at one of the leaf nodes in that access tree. Each PoP additionally serves as an *origin server* for a subset of the set of entire objects; the number of objects it hosts is also proportional to the population.[1] We assume that each cache has sufficient budget (i.e., storage capacity) to host a certain number of objects. We use different *budget configurations* in our simulations. Note that a PoP node serves two roles: (1) as a (root) node in an access tree and (2) as the origin

[1] We also experimented with other models such as uniform origin assignment and found consistent results.

server for a set of objects. As a regular cache, we assume the PoP node has a fixed budget, but as an origin server, we assume it has a very large cache to host all the objects it "owns".

Representative designs: We choose four representative designs from the broader design space from Section 5.3.

1. ICN-SP: This assumes pervasive cache placement and shortest path routing toward the origin server. That is, any cache along the shortest path may respond to the request if it has the object.

2. ICN-NR: This extends ICN-SP to also add intelligent nearest-replica-based routing. Our goal here is not to design new routing strategies or evaluate the overhead of these content-based routing protocols. We conservatively assume that we can route to the nearest replica with zero overhead.

3. EDGE: This is the simplest strategy where we only place caches at the "edge" of the network. The notion of edge depends on other economic and management related factors on whether it is viable to operate caches deep inside the network. We simply assume it is the access router of our given topology since our goal is to do a relative comparison between the different schemes.

4. EDGE-Coop: This uses the same placement as EDGE, but with a simple neighbor-based cooperative lookup strategy. That is, each router can do a scoped lookup to check if its sibling in the access tree has the object and reroute the request to that sibling.

All representative designs use LRU for cache management. Each cache on the path from where the object is fetched (i.e., the origin server or a cache) to the leaf at which the request has arrived stores the object. For reasons of scalability,

we use a request-level simulator and thus we do not model packet-level, TCP, or router queueing effects. Since our goal is to understand the relative performance of the different caching architectures at a request granularity, we believe this is a reasonable assumption. We optimistically assume that the ICN solutions incur no lookup or discovery overhead when modeling the response latencies and network congestion.

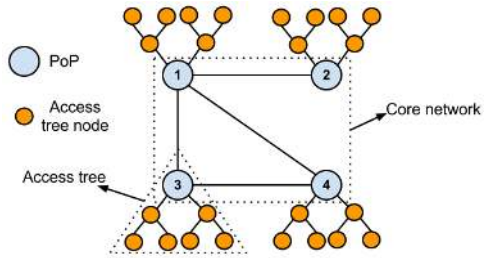


FIGURE 5.5: An example network topology with four PoP nodes and their corresponding access trees.

5.4.2 Baseline results

In this section, we use trace-driven simulations using the CDN request logs and corresponding synthetic request logs, which have similar numbers of requests, objects, and the best-fit Zipf popularity distribution.

For the following results, we report a normalized metric w.r.t to a system without any caching infrastructure. Thus, we focus on the *improvement* in response latency, reduction in network congestion, and reduction in server load. In each case, a higher value of the metric is better; i.e., the specific caching architecture is more beneficial.

We consider two cache budgeting policies for setting the cache size B_r for each router r. If there are a total of O objects being requested across the network of R routers, we assume that the total cache budget of the network is $F \times R \times O$, for some value of $F \in [0, 1]$. As a baseline, we pick $F = 5\%$ based roughly on the CDN provisioning we observe relative to the universe of objects each CDN server sees in

a day. We vary the parameter in the next section.

Given this total budget, we consider two possible splits:

1. Uniform: Each router r gets a fixed cache capacity to store 5% of the universe of all objects.

2. Population-proportional: We divide the total budget such that each PoP gets a total budget proportional to its population and then divide this budget equally within that access tree. In this case, the routers in expectation can store 5% of the objects.

We have also tried other cache budgeting policies and observe results that are qualitatively consistent. Due to space constraints, we do not report the results from those settings.

Note that this method of dividing the budget can be viewed as unfair to the EDGE and EDGE-Coop settings as they have a total budget that is half the capacity of the ICN-SP and ICN-NR cases. Thus, we also consider a new setting EDGE-Norm where we ensure that the total budgets are the same. That is, we take the EDGE configuration and multiply the budget of the edge caches by an appropriate constant (for example, 2 in case of binary trees) to make sure that the total cache capacity is the same.

Response latency: We report user-perceived latency in terms of the *number of hops* between the request and the location from which it was served. Figure 5.6(a) shows the percentage improvement in query latency for the four caching architectures (plus EDGE-Norm) in comparison to a network with no caching (i.e., all requests are routed to the origin PoP). We make three main observations. First, the gap between the different caching architectures is quite small (at most 9%) and this is consistent across the different topologies. Second, EDGE-Coop consistently achieves

comparable latency improvement compared with ICN-NR with a maximum gap of 3%. Third, nearest replica routing (ICN-NR) does not offer significant benefits over ICN-SP.

Figure 5.7(a) shows the latency improvement results for the case of uniform budget assignment across the access tree. We see no significant change in the relative performances of the different policies.

Network congestion: Other parallel work has focused on the interaction between ISP traffic engineering and "content engineering" and showed that there are natural synergies to be exploited here (Jiang et al., 2009b; Frank et al., 2012). Here, we focus on a simpler question of network congestion under different caching architectures. The congestion on a link is measured simply as the number of simultaneous object transfers traversing that link.

Figure 5.6(b) shows the effectiveness of caching in reducing the congestion level across the network. We focus on the maximum congested link in the network. Analogous to the query delay analysis, the percentage shown in each case indicates the improvement over the base case with zero budget. Once again, we see that EDGE-Coop delivers close to the best performance (with a maximum gap of 4%) and that the gap between the solutions is fairly small.[2] The success of edge-based approaches in this context is particularly promising. Unlike nearest-replica routing, caching at the edge strictly reduces traffic in the core of the network and thus eliminates any concerns that ISP traffic engineering and content engineering could be in conflict (Jiang et al., 2009b).

Figure 5.7(b) shows similar results for the case of uniform budget assignment across the access tree.

[2] The absolute improvement values for latency are typically lower than the numbers for the congestion improvement. The reason is that we are looking at the *average* in the latency metric and the *maximum* in the case of congestion.

Server Load: Next, we consider the load on the origin servers (i.e., the PoP nodes hosting the objects) in Figure 5.6(c). The metric we use here is the percentage reduction in the load on the origin server load with the maximum load in the network. Once again, we see that the various cache architectures show similar performance levels: maximum performance gap of 9% between EDGE-Coop and ICN-SP and a 2% gap between ICN-NR and ICN-SP.

Figure 5.7(c) shows the same measures for the case of uniform budget assignment across the access tree. No significant change in relative performances of different policies is observed.

Validating a synthetic request model: Ideally, we would like to vary the request popularity distribution. However, one concern is whether the performance gaps we estimate from synthetic requests log are comparable to real traces. That is, in addition to visually and statistically confirming the distribution fit in previous section, we want to ensure that this confirmation translates into system-level performance metrics.

To address this issue, for each request log, we also generate a synthetic request log with the best-fit Zipf distribution. The maximum difference between the two simulations in terms of improvement percentage over baseline is 2.74%. In Table 5.3 we show the maximum performance gap between the trace-driven simulations and the synthetic request logs. The predicted gap of ICN-NR over EDGE in different topologies (see Table 5.3) has a maximum value of 1.67%. The gap w.r.t congestion and origin server load improvements are similar and not shown for brevity. These results suggest that using Zipf-based synthetic logs is a reasonable approximation for a real trace.

5.4.3 Summary of main results

There are three main observations:

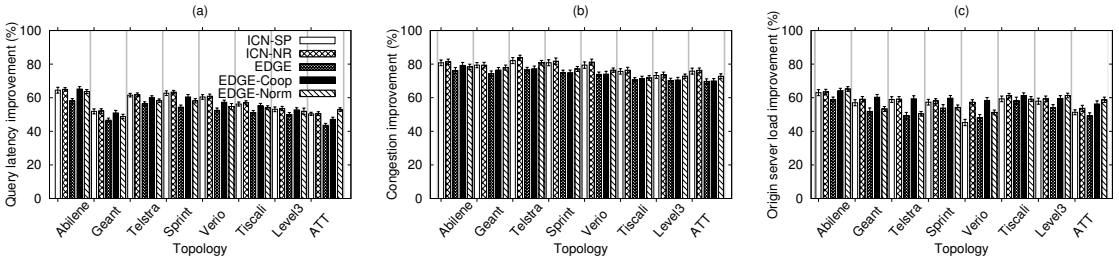


FIGURE 5.6: Trace-based simulations results. Cache budget and origin server allocation are set to be proportional to population. Parts (a), (b), and (c) show improvements in query latency, congestion, and maximum origin server load, respectively.

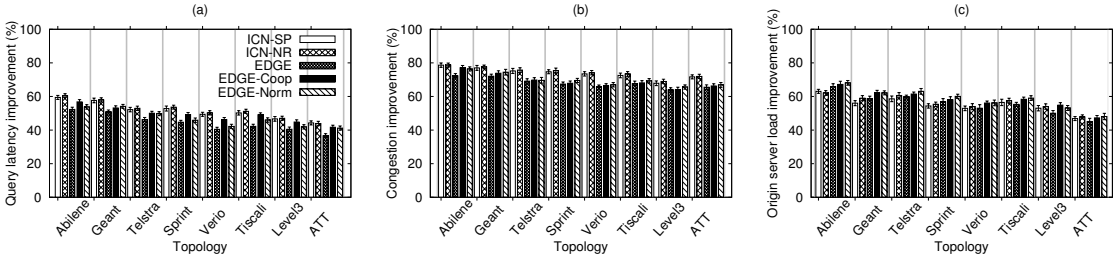


FIGURE 5.7: Trace-based simulations results. Cache budget and origin server allocation are set to be uniform across the network. Parts (a), (b), and (c) show improvements in query latency, congestion, and maximum origin server load, respectively.

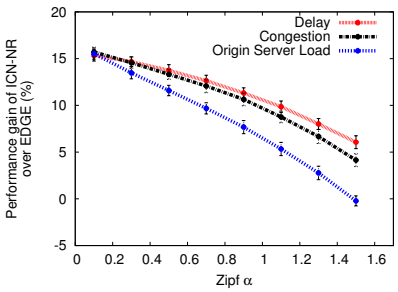
1. The performance gap between different caching policies on all three metrics (i.e., query latency, congestion, and server load) is small (at most 9%).

2. The performance gap between ICN-SP and ICN-NR is negligible (at most 2%); i.e., nearest-replica routing adds marginal value over pervasive caching.

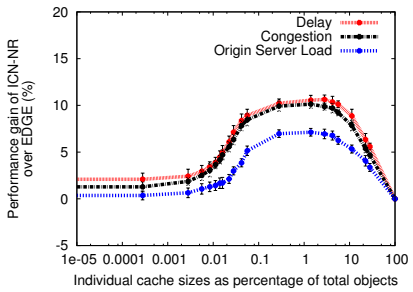
3. Different cache provisioning regimes (i.e., population-based and uniform) do not affect the relative performance of the methods.

Table 5.3: Comparison of simulation results for query latency on request trace and synthetic data (with best-fit Zipf).

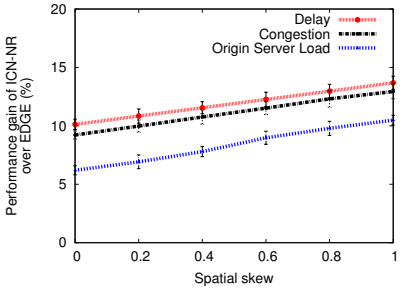
Topology	**Gap between ICN-NR and EDGE on trace (%)**	**Gap between ICN-NR and EDGE on synthetic data (%)**
Abilene	6.89	7.81
Geant	5.92	6.96
Telstra	7.44	8.63
Sprint	7.09	8.76
Verio	7.40	8.94
Tiscali	7.11	8.05
Level3	6.18	7.32
ATT	7.25	8.04



(a) Zipf α



(b) Cache budget



(c) Spatial skew

FIGURE 5.8: Effect of varying different simulation parameters on the performance gap between ICN-NR and EDGE. Here, we consider a fixed total cache budget across the nodes.

5.5 Sensitivity Analysis

The results of the last section are based on a *fixed* configuration with a specific popularity distribution, cache size, arity, and access tree depth etc. Future workloads and architectures may have more diverse characteristics and cache provisioning schemes. Therefore, in this section, we perform an extensive sensitivity analysis across different configuration parameters using synthetically generated request traces. For clarity, we only show results from the largest topology (AT&T) as the results are similar across topologies.

Rather than look at all cache architectures, here we focus on the two extreme points in this section, namely, ICN-NR and EDGE. In the following results, we report a *normalized improvement* metric:

$$RelImprov_{ICN-NR} - RelImprov_{EDGE}$$

where *RelImprov* is the improvement over the no-caching scenario that we mentioned in the previous section. By construction, a positive value of this measure implies ICN-NR performs better than EDGE and a negative value implies that EDGE performs better.

For clarity of presentation, we take the following two-stage approach. First, we begin by analyzing the sensitivity one dimension at a time, while retaining the baseline setup from the previous section for the remaining parameters in Section 5.5.1. Then, we focus on the combination of parameter(s) that provides the best performance improvement for ICN-NR in Section 5.5.2.

5.5.1 Single-dimension sensitivity

Zipf parameter α: Figure 5.8(a) shows that with increasing α, the gap between EDGE and ICN-NR becomes less positive. This is intuitively expected–as α increases, popular objects get a larger share. This reduces the value of pervasive

caching and nearest-replica routing because most of the requests are served from the edge caches.

Cache budget: Next, we consider the effect of increasing the cache size in Figure 5.8(b). As before, we represent the per-router cache size as a fraction of the total number of objects being requested. We see that the maximum improvement that ICN-NR can provide is around 10% when each cache can store $\approx 2\%$ of the objects. We also observe an interesting non-monotonic effect in the performance gap as a function of cache size. The reason is that with very small caches, none of the caching architectures are effective. With a sufficiently large cache ($> 10\%$), however, the edge caches account for a significant fraction of the requests and thus the marginal utility of interior caches is very low.

Spatial skew: In the previous section, we considered a homogeneous request stream where requests at different network locations are drawn from the same object popularity distribution. There are likely to be regional and local differences between the request streams observed at different locations. Thus, we explore the effect of *spatial skew* across locations in Figure 5.8(c). A spatial skew of 0 means that the requests at all locations follow the same global popularity distribution (i.e., objects have a unique global ranking). A spatial skew of 1, at the other extreme, implies that the most popular object at one location may become the least popular object at some other location.[3] Figure 5.8(c) shows that as the spatial skew increases, ICN-NR outperforms EDGE. Intuitively, with a large spatial skew, a less popular object at one location may become popular at a nearby location. Thus, caching objects that differ in their popularity across edge locations inside the network magnifies the benefit of ICN-NR.

[3] While the specific spatial skew metric we use is not crucial, we define it for completeness: suppose there are O distinct objects and P PoPs, and r_{op} denotes the rank of object o at PoP p. Let $S_o = stdev(r_{op})$ be the standard deviation of ranks of object o across all PoPs. Then, *spatial skew* $= \frac{avg(S_o)}{O}$, where $avg(\cdot)$ represents average.

Table 5.4: Effect of access trees arity on performance gain of ICN-NR over EDGE.

arity	**Latency gain (%)**	**Congestion gain (%)**	**Origin load (%)**
2	10.29	9.14	6.27
4	9.12	8.28	5.35
8	7.95	7.01	4.66

Access tree arity: Our baseline uses a fixed binary tree. Here, we evaluate how the structure of the tree impacts the performance difference by changing the arity, while keeping the total number of leaves fixed. Table 5.4 shows that as the access tree arity increases, the performance gap between ICN-NR and EDGE decreases. This is not surprising. With our cache budgeting mechanism, the ratio of total cache budget between ICN-NR and EDGE in a tree of arity a is $\frac{a-1}{a}$; with higher a this gaps comes closer to 1. In some sense, increasing arity in this case has a similar effect to normalizing the cache budgets in EDGE-Norm.

Other sensitivity parameters: We have also explored a range of other parameters, but we do not present the results due to space constraints as their effects are small compared to the above parameters. For completeness, we mention two other parameters that might be relevant in practice.

First, rather than assuming unit latency cost per hop, we vary the *latency model* in two ways: (1) arithmetic progression (with a positive difference) of latency toward the core and (2) the latency of each hop at the core network is k times higher. Under both models, the maximum performance gap between ICN-NR and EDGE is less than 2%. This can be explained in part by the intuition from Section 5.2.2; the intermediate levels see much fewer requests.

Second, we vary the serving capacity of caches and consider a heterogeneous workload with requests having different sizes. In this case, each cache has a limited serving capacity (i.e., the number of queries it can serve in a certain period of time is limited). If a request arrives at a cache that is overloaded, this request is redirected

to the next cache on the query path (or the origin). Again, we see that the maximum performance improvement of ICN-NR over EDGE in this case is less than 2%.

5.5.2 Best scenario for ICN-NR

We want to understand under what scenario ICN-NR has the best performance over EDGE and by how much. To this end, we begin by ordering the configuration parameters in decreasing order of the magnitude of the relative improvement. Then, we progressively change one dimension at a time to maximize the gap between ICN-NR and EDGE in Figure 5.9. In the figure, Baseline is the first configuration that is used in the previous section simulations. In each of the four subsequent configurations, one of the configuration parameters is changed (while all other parameters maintain their current values) as follows: in Alpha*, $\alpha = 0.1$; in Skew*, spatial skew= 1; in Budget-Dist*, uniform budgeting is applied (which is slightly more favorable to ICN-NR compared with population-based budgeting); and finally, in Node-Budget*, $F = 2\%$. (For completeness, we also tried a brute force exhaustive enumeration of parameters and found that the best case is identical to combining the best single-dimensional results.) We see that with the best combination of parameters, ICN-NR can improve the performance at most 17% relative to EDGE.

The next question we ask is whether this performance gap is fundamental or we can bridge it with simple extensions to the EDGE architecture. As we already saw in the previous section, simple extensions such as cooperation (as in EDGE-Coop) or doubling the budget (as in EDGE-Norm) can reduce the gap in the baseline simulations. Figure 5.10 shows how several natural extensions to EDGE can bridge the performance gap. In this figure, Baseline refers to EDGE without any changes; 2-Levels is EDGE augmented with one more layer of caching at the level above the edge; Coop refers to EDGE-Coop; 2-Levels-Coops combines the features of 2-Levels and Coop; Norm refers to EDGE-Norm; Norm-Coop is a combination EDGE-Norm

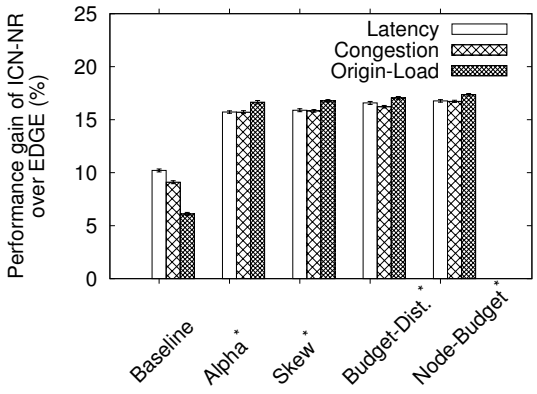


FIGURE 5.9: Exploring the best scenario for ICN-NR by progressively setting different configuration parameters to yield the maximum performance gap w.r.t. EDGE. The X axis represents cache structure configurations.

and Coop; Double-Budget-Coop is the same as Norm-Coop with the budget doubled. There are also two points of reference in the figure: Section-4 is the set of performance measures from Section 4 and Inf-Budget is a scenario in which both EDGE and ICN-NR have infinite caches (i.e., each cache has enough space to store O objects).

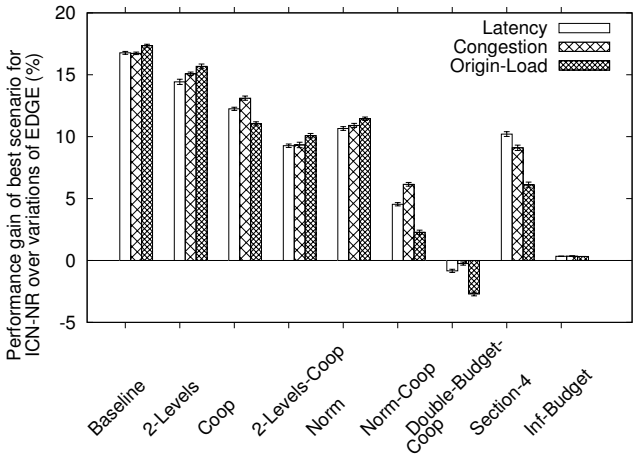


FIGURE 5.10: Bridging the performance gap between the best scenario for ICN-NR via simple extensions to EDGE. As points of reference, we also show the baseline performance from Section 5.4 and a hypothetical infinite cache setting. The X axis represents variations of EDGE.

5.5.3 Summary of key observations

The main observations from our sensitivity analysis are:

1. The key parameters that effect the relative performance of ICN-NR over EDGE are Zipf α and spatial skew.

2. The best possible performance benefit of ICN-NR over EDGE (across all metrics) by setting the above parameters to be favorable to ICN-NR is only 17%.

3. Simple extensions to EDGE such as putting all the cache at the edge and enabling local scoped cooperation can reduce even this best case performance gap to 6%.

4. Doubling the edge cache sizes can in fact make EDGE better than ICN-NR.

5.6 Conclusions

Our work can be viewed as an application of the end-to-end argument—we should impose significant changes to the network only if doing so will offer substantial performance improvements Saltzer et al. (1984). If anything, we have reasons to be optimistic about the benefits of ICN. We find that the components of ICN that might need drastic changes to the network as envisioned by some ICN proposals (pervasive caches and nearest-replica routing) do not appear to be fundamentally necessary.

6

Reliable Client Accounting for Hybrid CDNs

6.1 Introduction

In Chapter 3, we have demonstrated that hybrid CDNs can greatly save bandwidth for the service provider and reduce the costs of all parties involved in the content distribution process. However, they also face an inherent challenge: By definition, P2P communication occurs between untrusted clients, and therefore cannot be observed directly by the trusted infrastructure. As a result, faulty or compromised clients can mishandle peer communication in ways that are not observable by the infrastructure, and they can under- or overreport peer interactions. In principle, a compromised client may be able to censor or modify content, and inject unauthorized content; it may refuse, delay, or abort transfers to deny or degrade service to other clients; and it may misreport peer transfers in an attempt to manipulate the accounting for commercial content or services.

In practice, hybrid systems can take measures to mitigate this risk. For instance, clients can obtain signed content hashes from the trusted infrastructure to verify content received from peers. The infrastructure can control which clients may peer,

in order to make collusion of faulty clients more difficult. Client logs can be checked and suspicious or inconsistent records excluded at some cost in logging accuracy. Clients that have repeatedly been involved in disputed or aborted transactions can be blacklisted at some risk of blocking legitimate clients. Nevertheless, the potential remains for compromised clients to disrupt service quality and affect logging accuracy.

There is little evidence of widespread attacks of this type against hybrid systems today. However, as these systems become more popular, it is important to understand the risks. Therefore, we have (with permission) performed a 'red team' evaluation of Akamai's NetSession system described in Chapter 3. We have identified an attack vector that enables a single malicious client to report more than 30 GB of fictitious download activity per hour, an amount that can be further inflated through a Sybil attack (Douceur, 2002).

While this specific vulnerability has been removed, the underlying challenge remains: a hybrid system's accounting is based on information from untrusted peers that is difficult to verify, and a determined attacker could find other ways to exploit this vulnerability. The challenge also applies to other commercial hybrid content delivery systems, not just to NetSession, and may apply to other types of hybrid systems as well. For example, CDNs have developed methods for owner-operated network appliances to serve customer content and report on download activity (Lewin et al., 2006). Unlike edge servers, which are part of the CDN's infrastructure, these appliances are not under the administration of the CDN. Similarly, certain network games rely on direct communication and interaction between game consoles, with outcomes ultimately reported to a centralized infrastructure (Agarwal and Lorch, 2009). Hence, we are interested in a principled approach to reliable accounting of client interactions.

To address this challenge, we present a method for providing reliable client accounting in hybrid distributed systems. Our method uses the infrastructure nodes to

establish reliable facts about the clients, such as an upper bound on their available resources (which is essential to limit the effect of Sybil attacks). Moreover, all clients record a tamper-evident log (Haeberlen et al., 2007) of their actions and must periodically upload their log to an infrastructure node; this severely restricts the ability of malicious clients to lie without getting caught.

A key feature of our approach is the ability to *quarantine* suspicious clients. Quarantined clients cannot interact with other clients, and their requests are served directly by the infrastructure; thus, such clients are unable to misreport their actions or disrupt other clients. A key insight is that *in hybrid systems, quarantining is safe*: if an honest client is accidentally quarantined, its quality of service does not change, it merely causes a small amount of extra load on the infrastructure. Thus, the infrastructure can afford to err on the side of caution e.g., by using anomaly detection techniques with high false-positive rates, which are difficult for an adversary to escape.

To demonstrate that our approach is effective, we present RCA, a system that applies our approach to NetSession. We report results from a trace-driven evaluation, based on traces from Akamai's production deployment. Our results indicate that RCA increases the protocol overhead from 0.06% without RCA to 0.47% with RCA, relative to the amount of content served, and it requires clients to maintain approximately 550 bytes of extra information per MB of downloaded content that must be uploaded to, and checked by, the infrastructure. We also show that RCA is effective against a variety of attacks and misbehaviors by malicious clients.

The rest of this chapter is organized as follows: Section 6.2 demonstrates an accounting attack on NetSession. A method for providing reliable client accounting in hybrid distributed systems is then given in Section 6.3. Section 6.4 presents an application of our approach to NetSession. Section 6.5 evaluates our system based on traces from NetSession. Section 6.6 concludes this chapter.

6.2 Attacks on hybrid systems

In this section, we describe attacks on hybrid CDNs, including a novel class of *inflation attacks* on the reporting system. To demonstrate that existing systems are vulnerable, we report results from a successful inflation attack on the NetSession system.

6.2.1 Threat model

We assume that the nodes in the infrastructure (such as Akamai's edge servers) are correct and fully trusted by the operator. The clients, on the other hand, are untrusted and can be compromised or fail in various ways, so we conservatively assume that some subset of them is controlled by a malicious adversary. Clients can communicate with infrastructure nodes and with their peers, but the infrastructure cannot observe direct peer-to-peer communication.

We also assume that the system does not use strong identities and that membership is open, i.e., any client is allowed to join the system. In particular, this means that the adversary can potentially mount a Sybil attack (Douceur, 2002) on the system, e.g., by running multiple instances of the client software on the same machine.

6.2.2 Attack vectors

There are two aspects of this model that make reliable accounting fundamentally difficult. First, the clients are not fully trusted and could tell lies, suppress information, or mishandle the communication with their peers to deny them service. Second, a (potentially large) fraction of the events that are to be accounted for occur directly between the clients, where the infrastructure cannot observe them. Separately, each of these challenges would be easy to handle: if the clients were trusted, the infrastructure could rely on their correct handling of requests and the accuracy of their

reports; if the infrastructure was directly involved in all communication (as in Akamai's infrastructure-based CDN), it could intercede when clients misbehave towards their peers, and perform accounting based on its own records.

In the following, we focus on a novel type of accounting attack that exploits these challenges. We will refer to this attack as an *inflation attack*. In an inflation attack, the attacker causes the system to overreport the amount of service that it has provided. Using the same technical approach, one could also mount *deflation attacks*, in which the attacker would cause the amount of service to be underreported instead.

6.2.3 Inflation attack on NetSession

To demonstrate the potential impact of inflation attacks, we carried out such an attack on the NetSession system. The attack could be carried out easily by modifying the NetSession client software. We decided against this approach, partly because a modified client might have accidentally disrupted other clients or the NetSession infrastructure, partly because we were able to carry out the attack even with an unmodified NetSession client.

To accomplish this, we wrote a script that uses the client's API to repeatedly download a certain file, as well as a small proxy that interposed between our local NetSession client and its peers. Whenever our client attempts to contact a peer, our proxy returns a spoofed response that indicates that the peer has all of the requested blocks, and whenever our client requests any blocks, the proxy returns the blocks at LAN speed. This man-in-the-middle attack was possible because, unlike the messages exchanged between clients and the infrastructure, communication between peers was not signed. The effect of this attack was that our client would trigger a large number of downloads, and these downloads would complete at LAN speed. The resulting large number of downloads were reported to the infrastructure, even

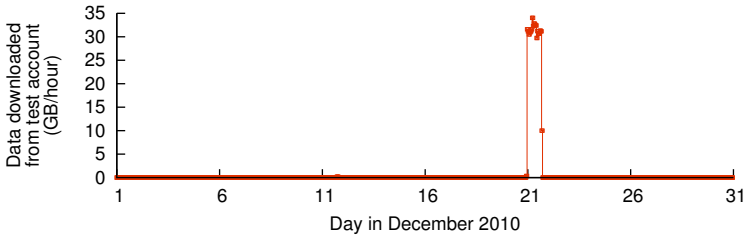


FIGURE 6.1: Effect of our attack on the NetSession logs.

though no content was actually transferred among the peers.

6.2.4 Impact of the attack

We obtained permission from Akamai to test our attack on the deployed NetSession system. To avoid the risk of interfering with the production system, we targeted a set of special files that is normally used for testing; thus, there was no risk that legitimate clients would attempt to download files from our proxied client, or that our attack would affect the logs of legitimate content providers. We ran our attack for one full day; during that day, we requested files as quickly as possible. To assess the impact of the attack, Akamai gave us access to the control plane logs for our client's specific GUID.

Figure 6.1 shows the reported downloads in this log. Our single modified client was able to generate about 30 GB/hour of fictitious downloads, which show up as a sharp spike in the figure. We note that this is a proof-of-concept attack, and that its throughput was limited by the throughput of our proxy. Had we chosen to directly modify the client software, we could have reported downloads at arbitrary rates.

6.2.5 How serious is this attack?

Our specific proof-of-concept attack is not difficult to prevent; indeed, the attack as described no longer works. However, our specific attack is just one example from an entire class of attacks; for instance, we could have reverse-engineered the NetSession client software and directly modified the reports it sends to the NetSession

infrastructure. So the root of the problem runs deeper.

Moreover, the vulnerability seems to exist in hybrid systems generally; it is inherent in the fact that hybrid systems must account for interactions between untrusted clients, who cannot be relied upon to report them accurately. To prevent similar attacks on NetSession and other hybrid systems, a more comprehensive solution is needed. Also, to show that a simple fix is not sufficient, we briefly discuss a few strawman solutions.

Sign all messages: Cryptography can be used to defend against man-in-the-middle attacks, but recall that this was just a trick we used to make our proof-of-concept attack easier to implement. Instead, an attacker could simply modify the client software to generate whatever messages he needs it to send.

Detect software modifications: There are techniques that aim to detect whether a client has modified software. Some of these techniques are purely software-based; for example, certain multiplayer games scan the clients' memory for known cheats (Punkbuster, 2000). However, an adversary can circumvent these techniques by disabling the scan or by reporting incorrect results. Other techniques require hardware support; for example, trusted platform modules can be used to certify that a client has loaded a certain software image (Garfinkel et al., 2003). However, the requisite hardware usually is not available on all clients, and even where it is available, it may be able to certify only that a certain binary was loaded, not that it is still running.

Limit clients to one download per file: This would thwart our specific attack; however, an adversary could still download many different files or create many Sybil identities that each download the file only once.

Anomaly detection: A massive load spike like the one in Figure 6.1 would probably raise the CDN operator's suspicion. However, there could be a legitimate reason for the spike, and there is no way to establish its provenance after the fact. Thus, the operator is caught in a dilemma: if the spike is genuine, it is probably important for

the content provider to know about it, so it must be left in the log; if the spike is fake, its presence distorts the accounting, so it should be removed.

6.3 Reliable accounting

In this section, we describe our method for providing reliable accounting in hybrid systems. Although we developed this method with NetSession in mind, it should be applicable to other types of hybrid systems (such as P2P streaming or storage systems). We begin with a description of the general approach and then show (in Section 6.4) how it can be applied to NetSession.

6.3.1 System model

We consider a system that consists of a number of trusted infrastructure nodes and a (potentially much larger) number of untrusted clients. The system offers a service to the clients that can be provided either by the infrastructure nodes or by other clients. The infrastructure cannot directly observe interactions between the clients.

Our goal is to provide an *accounting mechanism* that reliably captures client activity. Specifically, we are interested in activity by faulty or compromised clients that could degrade the performance of the system or distort the record of services actually rendered.

6.3.2 Threat analysis

Next, we characterize the kinds of threats that a faulty or malicious client poses to the rest of the system. To do this in a protocol-independent way, we model the client as an abstract state machine that accepts requests from the local user (e.g., names of files to download) and eventually produces responses (e.g., the contents of the file). The state machine can send and receive messages, and it must periodically upload a log of its actions to the infrastructure. Each state machine is expected to follow a

Table 6.1: Types of client misbehaviors. The last column shows the subsection that describes the countermeasure.

F1	Fail to log exact set of messages sent or acknowledged	6.3.4
F2	Fail to log consistent sequence of messages	6.3.5
F3	Execute illegal, or fail to execute required, protocol action	6.3.6
F4	Faulty peers collude to report fictitious exchanges	6.3.7, 6.3.8
F5	Render poor service to peers	6.3.8
F6	Nefarious user requests	6.3.9
F7	Sybil attack	6.3.10

specific protocol, and this protocol is not necessarily fully deterministic (e.g., clients might be allowed to choose the peers with the highest throughput).

Table 6.1 summarizes the types of threats we consider here. Faulty or malicious clients can *fail to log* the exact set of messages they have sent or acknowledged (F1), or fail to log them in a sequence that is causally consistent (F2). They can *violate the protocol*, either by making a bad state transition or by failing to make a required state transition (F3). They can collude with faulty peers in order to report fictitious transactions amongst each other (F4). They can *deliver poor performance*, e.g., by being slow to respond to messages from peers, by aborting or delaying peer transfers, or by sending corrupted content (F5). A user can issue *nefarious content requests* in order to create artificial demand for a provider's content or to degrade the service quality enjoyed by other clients (F6). Finally, a user can mount a *Sybil attack* by joining the system under more than one identity, in order to amplify other attacks (F7).

6.3.3 Approach

In a fully decentralized system, it is difficult or even impossible (Seuken and Parkes, 2011) to provide reliable accounting. In a hybrid system, however, we can do better by leveraging some of the unique characteristics of these systems, namely:

Trusted infrastructure: The nodes in the infrastructure are directly controlled by the operator; **Central control:** The operator can prescribe a single protocol that all the nodes must follow; **Global view:** The operator is able to observe the status of all clients eventually; and **Dedicated resources:** The infrastructure has the capacity to take over for under-performing or suspicious clients.

In the following, we describe a sequence of techniques for constructing a reliable accounting mechanism that takes advantage of these characteristics. Each technique adds some constraints on the kinds of behaviors a faulty node can manifest without getting caught.

6.3.4 Require message commitment

First, we require clients to cryptographically sign all messages and acknowledgments they send. Moreover, clients record the signatures of received messages and acknowledgments in a log, which they periodically forward to the infrastructure. As a result, a faulty client can no longer deny having sent or received a message it has previously sent or acknowledged. Likewise, a faulty client cannot falsely claim that a correct client has sent or acknowledged a message, because the faulty client would not be able to produce the corresponding signature.

6.3.5 Check logs for consistency

Even with message commitment in place, a faulty client could give a false account of the order in which certain events happened. For instance, a client could receive an object from the infrastructure, acknowledge it, and then later decline a peer's request for the same object. In its logged record, it could claim that the peer's request had arrived before it received the object from the infrastructure, in order to justify its failure to serve the object.

This form of misrepresentation can be avoided by requiring each client to maintain

a *tamper-evident log* (Haeberlen et al., 2007) of its actions. For this purpose, the log entries form a hash chain, and the hash of the most recent log entry is included in the signature of any message that the node signs. Thus, the node commits to its entire event history each time it sends a message or acknowledgment. Because the logs and all of a node's signatures are eventually sent to and checked by the infrastructure, a client would be caught if it ever omitted, fabricated, or manipulated events in its logs, or gave inconsistent accounts of the sequence of events.

Each client is forced to log a single linear account of its actions that includes all acknowledged messages sent to, or received from, correct peers. If messages are forged, omitted, reordered, or tampered with, the client effectively makes a signed admission of guilt.

6.3.6 Check logs for plausibility

Even when a client's log is consistent with the logs of all other clients it has communicated with, the log can still be implausible; for example, a client A might download a file from a peer B and then, when a third peer C requests that file from A, serve a modified version of the file or claim that it no longer stores the file. To prevent this, our second step is to verify that a log is *plausible*, i.e., it is consistent with a valid execution of the software the client is expected to run.

We can decide whether a log is plausible by checking that it satisfies a set of *invariants*, which must hold in any correct execution of the client software. Typical example invariants state that a client must only serve content it has previously received, may only contact or accept requests from peers suggested by the infrastructure, etc.

6.3.7 Control client pairings

If clients are free to choose which clients to request services from, malicious clients can collude to request services from each other (and thus make consistent and plausible logs without actually doing any work), or they can 'gang up' on some correct clients to deny them services. To make collusion more difficult, the infrastructure can impose *restrictions* on the clients, e.g., by limiting each client i to only request services from peers in some set S_i.

Restrictions should be neither too tight nor too loose. In the above example, very small sets S_i will force the clients to contact the infrastructure frequently (e.g., to ask for additional clients if the ones in S_i fail), which increases overhead and decreases performance, whereas very large sets S_i increase the chances that a malicious client i will find an accomplice in its set S_i.

The infrastructure's choice of S_i may depend on which peers have the requested content, which ISP a peer is connected to, and whether it has a compatible type of NAT. Some of these factors can be influenced by a peer; for instance, a peer could download rarely requested content in order to increase the chance that it will be paired with a colluding peer that requests that same content. Therefore, controlling client pairing can only mitigate but not eliminate client collusion.

6.3.8 Quarantine anomalous clients

A faulty client can degrade the service received by its peers, e.g., by providing the requested services inconsistently or too slowly. Moreover, as discussed above, colluding peers could inflate their upload activity. Our next step is to apply statistical *anomaly detection* to identify potentially problematic clients, and to quarantine those clients.

Ordinarily, anomaly detection systems face a difficult tradeoff between effectivity and the number of false positives. Our key insight is that, in the specific case of hybrid systems, *this tradeoff can be avoided by redirecting any suspicious clients to*

the infrastructure. In other words, when a client c manifests anomalous behavior, the infrastructure can restrict c to contacting only trusted infrastructure nodes, and it can tell other clients not to contact c.

If c is malicious, quarantining c will ensure accurate logging because a) c's interactions with the infrastructure will be logged by the trusted nodes, and b) c cannot plausibly log any interactions with other nodes. On the other hand, if c is correct and its detection was a false positive, c's requests will still be handled by the infrastructure, so c's user will still receive good service. Since quarantining clients is 'safe' from a QoS perspective, we can perform anomaly detection aggressively and accept a nontrivial false-positive rate, as long as the infrastructure has sufficient resources to handle the extra load.

6.3.9 Flag/throttle suspicious user behavior

In addition to the types of client misbehaviors covered in the previous section, there is a class of attacks that is caused solely by user activity, while the client software behaves as expected. For instance, a user could nefariously download content from a specific content provider, in order to drive up demand for that provider's content. (This type of attack would typically be combined with a Sybil attack and possibly a botnet. Also, note that this attack is not specific to hybrid CDNs.)

Based on the tamper-evident logs collected by the system, we can perform statistical anomaly detection to identify clients whose download activity stands out in terms of volume and content selection. A flagged client can be immediately subjected to a download rate-limit by the infrastructure, pending resolution by a human operator. At the same time, a human operator is notified of the anomaly for further inspection and resolution. For instance, an operator can contact a content provider to check if a sudden increase in demand (possibly from a specific set of IP addresses) is expected.

6.3.10 Enforce resource limits

Some of the attacks described in the previous sections can be amplified by a *Sybil attack*, where an attacker registers more than one instance of the client software for each physical node he controls. For instance, the impact of a colluding-peers attack or a nefarious download attack increases with the number of client instances. Without strong user identification, we cannot effectively prevent Sybil attacks, but we can at least constrain the aggregate amount of service that Sybils can log. For example, we can check that, in aggregate, the activities recorded in these logs cannot exceed the physical capacity of the adversary's nodes.

Since a hybrid system contains trusted infrastructure nodes, we can achieve this goal through resource testing. For example, a client could be required to demonstrate its upstream or downstream bandwidth in a short data exchange with the infrastructure. The infrastructure could refuse to accept multiple clients with the same IP address, and/or ask a group of clients with a common IP prefix to demonstrate that they run on separate machines by asking each of them to simultaneously solve a different crypto puzzle. The results could then be used to flag implausible client activity, such as clients who claim to have exchanged data with peers in different networks at a rate that exceeds their measured access link capacity, or clients who claim to have exchanged data with a number of clients on the same network that exceeds the number of separate machines they have demonstrated.

6.3.11 Summary

The cumulative effect of the above steps is to constrain what the adversary's nodes can do without being detected. The first four steps ensure that the adversary's nodes tell a 'straight story', i.e., send logs that are plausible, consistent with those of its peers, and feasible given their physical resources. If the adversary fails to comply, the infrastructure can detect this, discard the invalid logs, and blacklist the misbehaving

nodes. The last three steps force the adversary's clients to behave statistically like the overall set of nodes; if they do not, they are flagged and measures are taken to limit any damage they might cause.

6.4 Application to NetSession

In this section, we describe the RCA system, which applies the method from Section 6.3 to NetSession.

6.4.1 Overview

Our design instantiates each of the building blocks we have presented in Section 6.3. We use resource certificates (Section 6.4.3) to limit the aggregate bandwidth of Sybils, a novel implementation of tamper-evident logs that has been optimized for hybrid systems (Sections 6.4.4 and 6.4.5) to check for consistency, a set of NetSession-specific invariants (Section 6.4.6) to check for plausibility, and a set of statistical tests (Section 6.4.7) for quarantining anomalous clients. The rest of this section describes each of these building blocks in more detail.

The basic workflow in RCA is as follows. When a client i first joins RCA, it contacts one of the control plane servers and uploads a short file to demonstrate its link capacity; the control plane then issues the client a private key σ_i (for signing messages), a public key π_i, and a certificate Γ_i that encodes the measured capacity. The client can then download or upload content, just as in the original NetSession system, but it additionally maintains a tamper-evident log, which it periodically uploads to the control plane. The control plane forwards the logs to a set of backend servers, which process them and produce the accounting information. The control plane also applies statistical tests to detect and quarantine anomalous clients.

6.4.2 Assumptions

The design of RCA relies on the following assumptions:

1. Infrastructure nodes are trusted by the operator and can only fail by crashing.

2. All nodes have access to a cryptographic hash function H.

3. Faulty clients cannot forge the signature of correct peers or of the infrastructure.

Assumption 1 seems reasonable in a centrally managed CDN like NetSession; assumptions 2 and 3 are commonly assumed to hold for hash functions like SHA-256 and algorithms such as RSA, provided that the keys are sufficiently strong.

6.4.3 Resource certificates

To prevent malicious clients from reporting more activity than they physically have the capacity to perform, RCA uses a few simple resource tests, as discussed in Section 6.3.10.

When a client i first joins the system, it contacts one of the control plane servers and requests a key pair, which will constitute the client's identity for the purposes of reporting. The control plane then exchanges some amount of data with the client, and it measures the maximum throughput C_i that i achieves during the exchange.[1] Finally, the control plane then generates a fresh key pair σ_i/π_i and returns it to the client, along with a certificate $\sigma_P(\pi_i, G_i, C_i, A_i, T_i)$ that is signed with the control plane server's private key σ_P and binds the client's public key π_i to the client's GUID G_i, its measured capacity C_i, its IP address A_i, and an expiration time T_i (on the order of a few hours). When a client's current certificate expires or its IP address changes, the client repeats this process to obtain a fresh certificate.

[1] Note that this requires two-way communication and thus prevents malicious clients from obtaining certificates for spoofed addresses.

To prevent Sybil attackers from obtaining multiple certificates for the same IP address, the control plane internally maintains a table with all unexpired certificates. Suppose a client c requests a new certificate from an address A_k while there are still unexpired certificates for A_k. Then the control plane revokes[2] any certificates for A_k whose clients are not currently logged in, and it asks the remaining clients to upload *at the same time* as c. It then measures the aggregate bandwidth C of all the uploads and issues c a certificate for the difference between C and the sum of the capacities from the existing certificates. To defend this mechanism against malicious clients that attempt to overload the control plane with requests for new certificates, clients can be required to solve a puzzle (Parno et al., 2007a) before submitting a request; the difficulty of the puzzle can be a function of the current load on the control plane.

In summary, the infrastructure ensures that there can be only one valid certificate per IP address at a time, and that an adversary with an aggregate capacity C cannot obtain certificates whose aggregate capacity exceeds C. Additional resource tests could be implemented and the results included in the resource certificate.

6.4.4 Tamper-evident log

RCA requires clients to maintain a tamper-evident log of all the messages they send and receive. Unlike previous implementations designed for decentralized systems (Haeberlen et al., 2007), a hybrid system requires different tradeoffs. On the one hand, logs are audited exclusively by the infrastructure, which simplifies the implementation. On the other hand, we need to aggressively minimize the overhead for the infrastructure—particularly the number of cryptographic signatures it has to verify—since we expect the number of clients to be orders of magnitude higher than the number of infrastructure nodes. The pseudocode of our implementation is

[2] In our setting, revocation is comparatively easy because each client has to show its current certificate to the control plane when logging in.

provided in Appendix A.

Each client maintains a log of entries $e_k := (h_k, s_k, t_k, c_k)$, where h_k is a hash value, s_k a sequence number, t_k an entry type (SEND or RECV), and c_k some type-specific content. The hash values form a hash chain of the form $h_k := H(h_{k-1}||s_k||t_k||c_k)$. Whenever a node i sends a message, it must attach an *authenticator* $(s_k, h_k, \sigma_i(s_k \,||\, h_k))$, which is signed with i's private key σ_i and represents a commitment to the current state of i's hash chain. Each message must be acknowledged, and clients may have at most $\mathrm{n_{max}}$ unacknowledged messages in flight at any given point in time. Finally, each message or acknowledgment contains enough information to verify that its transmission has been recorded in the log. If i forges, omits, or tampers with log entries after the fact, the infrastructure can detect this by comparing the log i has uploaded to the authenticators i has sent to its peers.

RCA's tamper-evident log maintains sub-chains for each pair of communicating peers. As a result, RCA's authenticators are cumulative, i.e., the authenticator in a message or acknowledgment from i can be used to verify all previous messages or acknowledgments from i, respectively. Hence, each client need only keep one pair of authenticators for each other peer it has communicated with–rather than one for each message it has sent or received–which dramatically reduces the number of authenticators that must be uploaded to, and verified by, the infrastructure.

In summary, the tamper-evident log ensures that inconsistencies between logs can be attributed to a specific misbehaving client.

6.4.5 Consistency checking

When a client i is ready to upload its log λ_i, it signs λ_i with its private key σ_i, and it attaches its certificate Γ_i as well as the set A_i of authenticators it has collected from other nodes. The infrastructure must then check the log for consistency and plausibility.

At first glance, the consistency check seems to require cross-checking logs from different clients. However, RCA's tamper-evident log is structured such that, for each message transmission, *both* endpoints have sufficient evidence (specifically, an authenticator from the message or its acknowledgment) to show that the entry in the local endpoint's log is consistent with the entry in the remote endpoint's log. Thus, logs can be checked individually, which makes the log checking both efficient and trivially scalable.

Although i uploads the above information to a specific infrastructure node, other infrastructure nodes may also require information about i, namely authenticators or a certificate revocation. Before checking can begin, all information about i must be collected at a single node $H(i)$, which can be chosen, e.g., via consistent hashing. This requires a 'shuffle' step (similar to the one in MapReduce) in which each infrastructure node sends copies of its received authenticators to the nodes that are 'responsible' for them.

Next, the infrastructure inspects λ_i and checks whether a) the log is well-formed and signed with σ_i; b) the certificate Γ_i is valid and matches σ_i; c) Γ_i was not expired or revoked when the log was signed, d) at no point in the log were there more than n_{max} unacknowledged messages, e) each of the sub-hashchains is intact, f) the end of each sub-hashchain corresponds to one of the authenticators uploaded by i, and g) λ_i is consistent with all authenticators signed with σ_i. The last check can be done incrementally if more authenticators are uploaded. If any of the above checks fails, i is clearly faulty. The GUID of a faulty client is immediately disabled so that it cannot participate in peer-to-peer transactions or infrastructure downloads; also, the system operator is notified.

6.4.6 Plausibility checking

When a client's log passes the consistency check, we know that its recorded sequence of messages is consistent with the log of other clients. However, the messages in the log do not necessarily correspond to a valid execution of the client software. To detect misbehaving nodes, RCA checks each log to see if it satisfies the following invariants, which capture the essence of RCA's swarming protocol. Specifically, clients

1. may only exchange data with peers or edge servers that were suggested to them by the infrastructure;
2. may only serve data they have already downloaded;
3. must not modify blocks before serving them;
4. must serve blocks they have available (i.e., blocks they store and for which peering is enabled); and
5. may only request blocks they do not already store.

If the log does not satisfy all of the invariants, RCA disables the GUID of the client and notifies the system operator. Otherwise, RCA identifies, for each uploaded data block, the content provider that owns the corresponding file, and it tallies, for each content provider, the number of bytes that were uploaded on its behalf, minus any bytes that would exceed the bandwidth in the client's resource certificate.

6.4.7 Statistical tests and quarantine

RCA's control plane continually maintains statistics about the download and upload activities of each client, such as its IP address, its geolocation, or the number of bytes downloaded and uploaded during the last k days. The control plane uses this data and a set of statistical tests to identify anomalous clients.

When a client i is flagged as anomalous, the infrastructure quarantines i and redirects any future download requests from i to the infrastructure nodes. The client will continue to receive service, however; RCA merely ensures that any interactions with i involve at least one trusted endpoint, so i's actions can be accounted accurately. (Logs produced by i before the quarantine will still be accepted, provided that they pass all the other tests.) In other cases, the infrastructure merely notifies a human operator for resolution.

There are many kinds of statistical tests that could be useful. For example, tests can be based on an client's own actions (e.g., downloading unusually many bytes), deviations from the actions of related clients (e.g., sudden load spike in a certain geolocation, but not in nearby geolocations), changes in file popularity, etc. More advanced tests could be based on machine learning or techniques from the botnet literature, e.g., BotGrep (Nagaraja et al., 2010). In Section 6.5.7, we describe and validate a small set of statistical tests for the NetSession system.

6.4.8 Limitations

RCA's tamper-evident log is only guaranteed to detect inconsistencies in message exchanges when at least one of the two endpoints is an honest node; if the infrastructure (unknowingly) pairs up two colluding clients, one of them can claim to have downloaded a large part of the file from the other without actually having done so. Controlling client pairing, applying statistical tests, and quarantining help to mitigate this limitation.

A second limitation is related to the use of anomaly detection. Since the operator usually does not know which clients are malicious, he can only base the statistical test on the observed behavior of all clients, which is only safe as long as the fraction f of clients controlled by a single adversary is small. If f is large, the adversary can slowly change the behavior of his clients over the course of several weeks or months,

analogous to a frog-boiling attack (Chan-Tin et al., 2009); this might prompt the operator to adjust the statistical tests, which would progressively relax the constraints on the adversary. However, we expect that in practice, few adversaries would have both the required number of clients and the necessary patience.

6.5 Evaluation

To evaluate RCA, we implemented a clone of the NetSession client and infrastructure software, called NetSession-Base, which includes all functionality required for our experiments. NetSession-Base is complete enough to run on the Internet. However, we perform most of our experiments in a network emulation environment, which can run hundreds of NetSession-Base clients on a single machine. The network emulator models bandwidth (upstream and downstream) and propagation delay, but not packet loss. Emulations are driven by a trace that defines the node characteristics (geolocation, link capacities, IP and GUIDs) as well as the workload, i.e., the downloads and their precise timing.

We then added RCA's defenses, including the tamper-evident log, the consistency and plausibility checks, the statistical checks, and the client quarantine. We use RSA with 1024-bit keys for the cryptographic signatures. We will refer to the system with defenses enabled as NetSession-RCA.

6.5.1 Validation

The goal of our first experiment is to verify that our clone matches the behavior of the Akamai NetSession system closely enough so that we can use the NetSession-Base system as a baseline in subsequent experiments. For this purpose, we used Akamai's NetSession client to download a 760 MB file in the live system, and used Wireshark to capture the network traffic from and to the client.

From the captured network traffic, we then compiled a trace that replicates this

download in the emulator, such that the same proportion of data are downloaded from the same number of peers and the infrastructure. We then ran NetSession-Base using this trace, and we measured the client's control and data traffic exchanged with each peer and the infrastructure. The results were all within 1% of those obtained with Akamai's NetSession.

6.5.2 Experimental setup

For the following experiments, we used a 30-day trace from Akamai's live NetSession system recorded in December, 2010. The trace includes an identifier and size for each object requested, the time when the download was initiated and completed, and the number of bytes downloaded from other peers. Our network emulator cannot scale to the entire workload recorded in the trace, so we used a sample that includes all downloads initiated by a randomly chosen subset of 500 clients.

We assigned client link capacities by randomly sampling from a measured distribution of download and upload speeds in residential broadband networks (Dischinger et al., 2007).

The NetSession traces do not record how many and which peers were actually used in a download, or how many bytes were obtained from each. In any case, because our emulation only includes a small sample of the actual peers, it would not include many of the peers who actually uploaded to one of the peers in the sample. Instead, our emulation assumes that all peers in our sample can serve all files to other peers. The infrastructure suggests a random set of emulated peers for each download, and the swarming protocol dynamically requests content from this set of peers based on the observed bandwidth. Since our evaluation is not concerned with the dynamics of the swarming protocol, this approximation does not affect the results. To be conservative, we fixed the overall ratio of bytes downloaded from peers versus bytes downloaded from the infrastructure to 80%; based on our observations

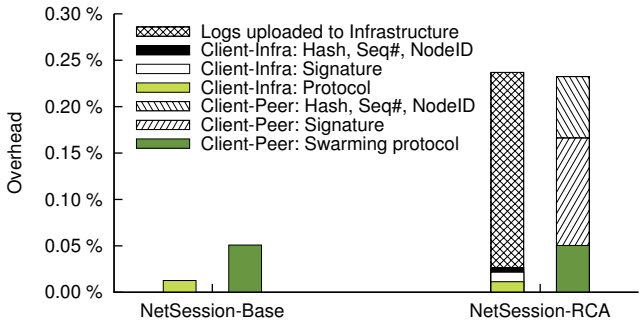


FIGURE 6.2: Network traffic overhead, normalized to the size of the downloaded content. For each system, the figure shows the communication with the infrastructure (left bar) and with peers (right bar).

from the trace, this will overestimate the overhead of our system.

6.5.3 Cost: Traffic

To quantify the additional bandwidth requirements of NetSession-RCA, we measured a) the total number of bytes downloaded as actual payload by all clients, and b) total number of bytes transmitted in the system. The difference between the two numbers is an estimate of the bandwidth overhead of the system relative to the actual payload bytes; it was 0.06% for NetSession-Base and 0.47% for NetSession-RCA. This amounts to a 7.8-fold increase in bandwidth overhead for NetSession-RCA.

While the relative increase in overhead is substantial, it is important to note that the absolute bandwidth requirement is still modest. The average per-peer bandwidth requirement is only 192 KB/day for NetSession-RCA and 26 KB/day for NetSession-Base. In return, NetSession-RCA provides much more fine-grained and reliable information about peer behavior than NetSession-Base.

Figure 6.2 shows a more detailed breakdown of the results. Both NetSession-RCA and NetSession-Base exchange some control messages with peers and with the infrastructure; the corresponding amount of traffic is small and identical in both systems. RCA adds an authenticator and an acknowledgment for each message. The

overhead is higher for the infrastructure traffic because the number of messages is higher: in addition to the data blocks, this traffic also contains a number of small control messages. Finally, clients must upload their logs to the infrastructure.

6.5.4 Cost: CPU

NetSession-RCA requires more client CPU than NetSession-Base, because it must generate and verify the signatures in authenticators. Because NetSession is intended to run in the background without inconveniencing the user, this additional computation must not consume more than a small fraction of the CPU.

To estimate the cost, we measured the number of signature generations and verifications performed by clients as part of the download activity. We then benchmarked RSA-1024 signature generation and verification on a single core of an Intel Xeon X5650 CPU, and we used these benchmarks to estimate the additional CPU load that would be caused by these operations. The maximum additional CPU load over all clients was never more than 0.5%.

6.5.5 Cost: Log storage and log upload

NetSession requires each client to maintain a log, and to periodically upload this log to the control plane. However, NetSession-RCA's log is considerably more detailed because it keeps track of individual messages, whereas NetSession-Base's log merely records occasional download progress reports. In both cases, the exact size depends on the client's activity.

To quantify the amount of log data, we ran NetSession-Base and NetSession-RCA with log uploading disabled; thus, each of 500 clients kept its *entire* 30-day log on its local disk. We then examined the log sizes at the end of the experiment. The average log size was 2.5 MB; the 5th and 95th percentiles were at 1.4 MB and 3.4 MB, respectively. The largest log was 86 MB. If we (conservatively) estimate the

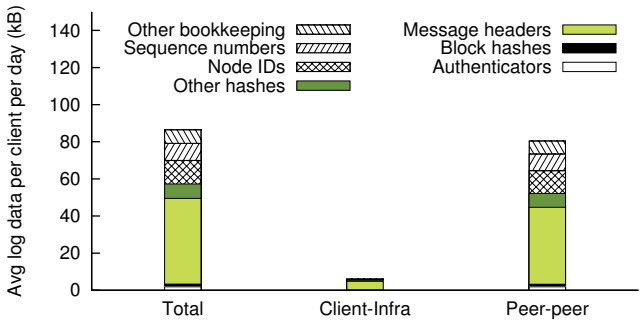


FIGURE 6.3: Average log size per client

average download activity per client at 1 GB per month, a larger deployment with 100 million GUIDs would generate about 1.8 TB of logs per day. This corresponds to only 18 kB of logs per client per day.

Figure 6.3 provides a detailed breakdown of the log contents collected from the clients. A comparison with Figure 6.2 shows that the log is considerably smaller than a complete message trace; this is because a) the log contains only a hash of each data block rather than the actual bytes (which are known to the infrastructure anyway), and b) the log does not contain every single authenticator, but only the most recent one for each client.

Each client periodically uploads its log to the control plane and then deletes the entries once they have been acknowledged. Thus, the amount of storage that is needed locally on each client depends on the upload interval. With daily uploads, less than a MB of storage is required. Since every logged byte must eventually be uploaded, the amount of network traffic generated by the uploads is largely independent of the upload interval.

6.5.6 Cost: Log processing

Once the logs have been uploaded, the infrastructure must perform the consistency and plausibility checks described in Section 3.2. The required processing time depends on what actions are recorded in each log, but we expect it to be correlated

Table 6.2: Statistical tests used with NetSession-RCA, along with the additional load (#bytes served) they place on the infrastructure due to quarantined clients.

#	Which peers are quarantined?	Parameters	+Load
T1	IP has downloaded $> N_1$ bytes during the last k_1 days	$k_1 = 20$ $N_1 = 15.2G$	1.00%
T2	IP has been used by $> g_2$ GUIDs during the last k_2 days	$k_2 = 1$ $g_2 = 2$	0.57%
T3	GUID has downloaded $> N_3$ bytes during the last k_3 days	$k_3 = 20$ $N_3 = 10.4G$	0.99%
T4	GUID has downloaded $> N_4$ files during the last k_4 days	$k_4 = 20$ $N_4 = 140$	0.92%
T5	GUID failed to validate $> N_5$ bytes during the last k_5 days	$k_5 = 1$ $N_5 = 200K$	1.00%
	Tests T1–T5 combined		2.81%

with the overall log size.

To estimate the overhead, we performed the consistency and invariant checks on each of the logs produced by the clients. We measured the total processing time, as well as the fraction of time spent on consistency and invariant checks. Since we expect cryptographic operations to be a major factor, we separately measured the time spent verifying signatures (recall that the infrastructure does not generate signatures).

On average, about 0.78 MB's worth of log data was processed per second on a single CPU. 9% of the time was spent on consistency checking and 91% on invariant checking; overall, signature verifications accounted for about 3% of the processing time. Log processing can easily be parallelized, e.g., using MapReduce.

Based on these results, we estimate that a deployment with 100 million GUIDs would require about 28 extra machines to process the logs. For comparison, NetSession's current log processing system requires around 10 machines. Both estimates assume that the machines are fully utilized; in practice, log processing is one of several jobs that runs on a larger cluster.

6.5.7 Examples of statistical tests

Developing a full set of statistical tests for NetSession would require a detailed characterization of its workload, which is beyond the scope of this chapter. However, we use the set of simple tests in Table 6.2 to illustrate the general principle. These tests are fully automated; they are designed to constrain clients who collude to overreport uploads or deliver bad service to peers. We expect that more sophisticated tests will be based on detailed workload characteristics, e.g., channel switching patterns (Cha et al., 2008) or the clients' response to the quarantine.

More aggressive tests reduce the amount of misbehavior an adversary can get away with, but they also increase the load on the infrastructure due to false positives. To give a rough impression of how aggressive our simple tests could be, we used the 12/2010 trace to determine, for each test, the set of parameters that a) causes at most 1% additional load on the infrastructure, and among those, the one that b) constrains the adversary the most. These parameters, and the resulting load increases, are also shown in Table 6.2. Note that a given client can trigger more than one test; hence, the load increase from a set of tests is lower than the sum of the increases from the individual tests.

6.5.8 Effectivity

As a sanity check for the NetSession-RCA implementation, we injected a set of sample attacks. Specifically, we injected five inflation attacks and one corruption attack: In *blatant liars*, one client uploaded a fabricated log claiming to have downloaded 1 TB. In *collusion*, two clients continuously requested files and then reported that they had downloaded them from each other, regardless of which clients the infrastructure suggested. In *flash mob*, five clients joined the system simultaneously and rapidly requested rare files, 'faking' downloads whenever the infrastructure paired up two of them. In *leechers*, five clients joined the system simultaneously and downloaded

random files as quickly as possible. In *Sybil attack*, one client joined the system with five GUIDs; the first GUID downloaded a rare file, and the others then tried to download the same file from each other. Finally, in *confused clients*, one client uploaded malformed log entries.

In all cases, the system behaved as expected. Logs from faulty clients were discarded, clients with abnormal behavior were quarantined, and the uploads affected by flash mob and Sybil attackers were effectively capped. In particular, *blatant liars* and *confused clients* were caught by our consistency checks (Section 6.4.5), *collusion* was detected by the plausibility checks (Section 6.4.6), *flash mob* was identified by resource certificates (Section 6.4.3), and finally *leechers* and *Sybil attack* were flagged by statistical checks (Section 6.4.7).

6.6 Conclusion

In this chapter, we have examined a fundamental challenge in commercial P2P-infrastructure hybrids: how to reliably account for the actions of untrusted clients. In current hybrid systems, malicious peers can report fictitious content downloads and degrade the system's quality of service. We described and evaluated RCA, a system that leverages the unique characteristics of P2P-infrastructure hybrids to limit the loss of accounting accuracy and service quality resulting from faulty or malicious clients. RCA reliably discovers all misreporting and protocol violations by individual clients, and it can automatically quarantine potentially colluding clients, at a moderate cost in terms of bandwidth and load on the infrastructure.

7

CDN as Security Overlay Network

7.1 Introduction

Modern businesses are suffering from great revenue losses due to pervasive cyber attacks such as distributed denial of service (DDoS), SQL injection, and cross site scripting. Successful attacks could result in leakage of commercial secrets and clients' privacy, unavailable or degradation of online services. These attacks are growing both in size and scale. For example, Arbor Networks reported that the average size of a DDoS has increased by 20% from a year ago to 1.77 Gbps in the first quarter of 2013 (Kerner, 2013).

Unfortunately, the Internet architecture is not inherently designed to offer any defense against even the simplest forms of attack such as SYN-flooding. In response to which, many researches have been done to mitigate attacks such as DDoS flooding. These efforts can be roughly divided into three categories.

The first category try to detect and filter out bad traffic before links get congested. For example, Ioannidis and Bellovin (2002) suggested a way to notify upstream routers to drop attack packets in order to mitigate DDoS attacks as close to source as

possible. Liu et al. (2011) proposed an architecture where access routers are set aside to listen on feedbacks from bottlenecked routers and police network traffic. However, this approach requires router support and fundamental networking changes, and is usually not powerful enough to withstand attacks that utilize large scale of botnets.

The second category require senders to obtain permissions before putting packets onto networks (Machiraju et al., 2002; Yaar et al., 2004; Anderson et al., 2004; Yang et al., 2005). Some of them are based on the proof-of-work paradigm and rely on resource testing (Parno et al., 2007b). The problem with this approach is its introduction of significant computational overhead on both clients and the origin servers for doing cryptographies. Furthermore, it is usually too cost-inefficient for individual content or service providers to afford the additional server and bandwidth reserved to tackle with sudden burst of attacks.

The third approach, namely security overlay network, are proposed in order to address the above concerns. The security overlay are infrastructures provided by CDN that shield victim server from attackers. Bad traffic are filtered out by the overlay nodes and not forwarded to the secret origin server being protected. This approach greatly benefit from scale of economics, such as shared expertise and high cost efficiency for individual businesses.

In this chapter, we conduct a close examination of the CDN security overlay designs as a cyber attack combating mechanism. We show that the approach is a better solution for business entities because it can offload attack traffic at the edge and thus cloak the origin servers from the attackers. We further discuss extensions to maximize benefits provided by this approach.

The remaining of this chapter is organized as follows: Section 7.2 lays out the background and motivation for security overlays. Section 7.3 gives a description of design goals and architecture of a typical security overlay network. Section 7.4 conducts a measurement study of an example system. Based on the design goals and

results of the study, two improvements to the system are proposed in Section 7.5. Finally, we conclude the chapter in Section 7.6.

7.2 Background and Motivation

This section gives a case study of a series of real-world DoS attacks dubbed "Operation Ababil' and how it affects the performance of CDN and origin servers. The important role CDNs play in today's Internet security and the motivation for an "always-on" security overlay network are then discussed.

7.2.1 Operation Ababil

Operation Ababil is a sequence of Distributed Denial of Service (DDoS) attacks towards various American financial institutions. Despite suspicion of Iran government's involvement, the cyber attacks are largely believed to be carried out by a group named Qassam Cyber Fighters (QCF). The group has announced four waves of attacks by the time this dissertation is written. This chapter focus on their security natures rather than its economic consequences.

Regardless of phases, the majority of QCF's attack traffic are initiated from BroBot botnet, which consists of compromised WordPress and Joomla content management systems (CMS) and virtual private servers(VPS). Compared to vulnerable home computers and other botnets, the BroBot attracts QCF group with its high supply of vulnerable servers and availability of server bandwidth.

Phase 1 attacks was announced in September 2012 and ended approximately a month later. The attacks target one bank per day usually on the main or search pages of its website. UDP garbage flood of packets that contain large blocks of repeating 'A's (0x41) are used to saturate Internal DNS servers of various CDNs and financial organizations. The hacktivist group has also began to use HTTP dynamic contents to circumvent static caching defenses in the late part of the first phase.

Starting from December 12th, 2012 and ending on January 29th, 2013, phase 2 of Operation Ababil is characterized by more extensive BroBot usage and cunning firewall bypassing techniques. For example, query strings with valid argument names and random values are added to evade filtering. Random query string are appended to PDF file names of the victims' websites to form an attack request. Also, burst probes are employed to cheat rate-limiting controls. BroBot nodes are observed to be sending high volume of attack traffic (up to 18 million requests per second) during the short burst period and resume attack after a long dormancy of hours or even days.

Phase 3 lasts from late February to mid May of 2013. Compared with previous phases, this wave of attack are more focus on layer 7. During phase 4, which begins in late July of 2013 and is still ongoing, more complex obfuscation has been observed. Inside Botnet, more CMS vulnerability has been discovered and exploited; fake plugins have been used to infect multiple files.

All phases of Operation Ababil have caused disrupted or degraded service of multiple financial institutions such as Bank of America, Chase Bank and New York Stock Exchange and have presented researchers with new challenges on Internet security.

7.2.2 Perceived Impact on Akamai

CDNs served as the front line defense to many financial organization targeted by QCF and witnessed the phase evolutions of Operation Ababil. We used data and charts collected by Akamai during each phase of Operation Ababil to exemplify CDNs' perception of this attack series.

In the recent years, Akamai has observed rapid growth in both number and scale of DoS attacks (Figure 7.1). The attacks are originating from all geographical locations and a single attack can move between different geographies. A typical

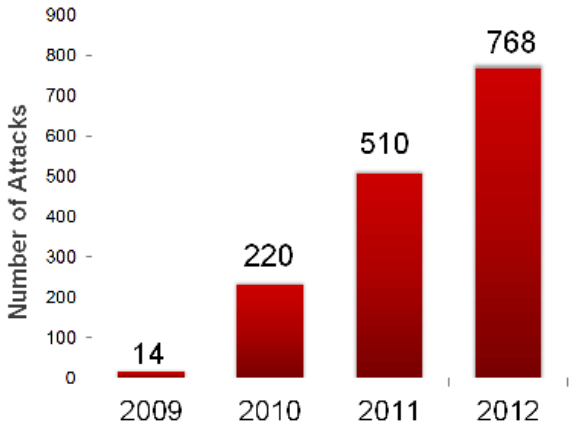


FIGURE 7.1: Attacks on Akamai customers.

attack size is around 10 Gbps while large attack can consume a total bandwidth of more than 100 Gbps.

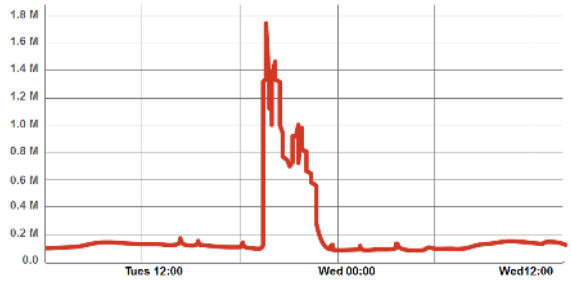
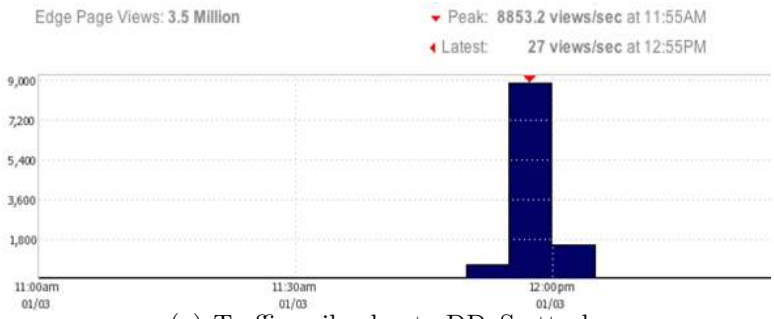


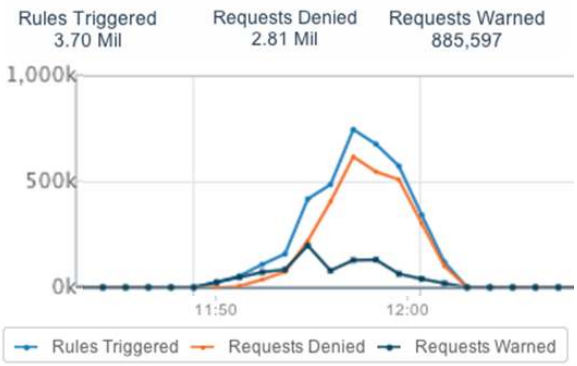
FIGURE 7.2: DNS traffic handled by Akamai.

Figure 7.2 show the impact of a phase 1 attack on Akamai's DNS servers back in September 2012. The attack lasts for about 4.5 hours and traffic during peak hour is 23 Gbps, nearly 10,000 time of normal traffic size. The attack packets are missing valid DNS headers and stuffed with abnormally large payload (around 1400 bytes). The packets are directed towards UDP port 53 paralleled SYN-flood against TCP port 53 in order to disrupt normal DNS service.

An example of phase 2 attack successfully mitigated by Akamai is presented in Figure 7.3 (Sitaraman et al., 2014). The website in concern was hit by an access rate of 9000 pages/sec during the attack as compared to a normal rate of 50 pages/sec, leaving a sudden spike in Akamai's security monitor. Over 90% of the attackers'



(a) Traffic spike due to DDoS attack.



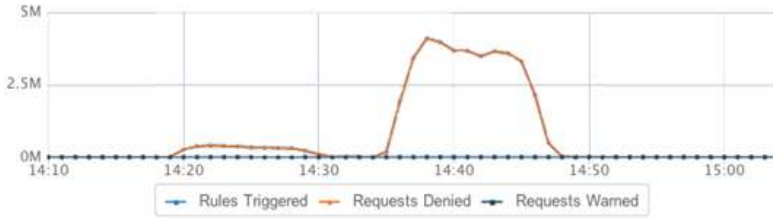
(b) Attack traffic filtered out by edge servers.

FIGURE 7.3: Phase 2 DDoS attack

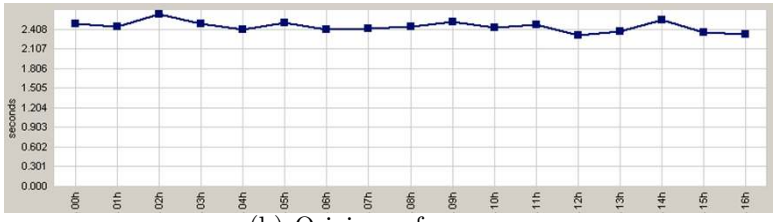
requests were denied by firewall rules and not forwarded, saving the origin from the significant surge of traffic.

On March 5, 2013 morning, Akamai faced one of the largest attack wave of phase 3 Ababil. Figure 7.4(a) is a real-time graph generated by Akamai security monitor, which provide direct insight on the attack patterns and size. As can be seen from the graph, the pear attack traffic is greater than 4 million requests/min, pouring 70 times normal traffic volume in to the CDN's edge servers. The assault lasted for around 20 minutes and more than 2000 agents are identified to have been participated, and 80% of the agents were new IP addresses that had not participated in earlier campaigns. The attack has been focused on PDF files with random query parameters, marketing pages for new customers and login pages of many financial organization website.

Fortunately enough, the security overlay provided by Akamai immediately realize



(a) Traffic spike due to DDoS attack.



(b) Origin performance.

FIGURE 7.4: Phase 3 DDoS attack

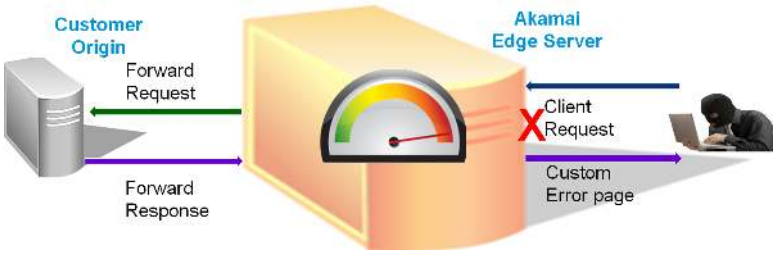
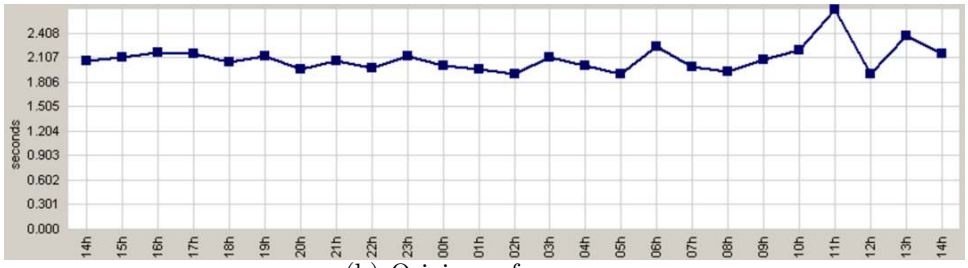


FIGURE 7.5: Rate control on forward requests.

the attack by counting the number of forward request to origin server (Figure 7.5), and 1700 new IP addresses are blacklisted without operator's involvement because of excessive forward requests. Most of the remaining attack traffic is blocked by various custom rules. As a result, the traffic forwarded to origin server remained below 1% of the link capacity; the origin availability stayed at 100% throughout the attack, and its application performance was as fast as normal (response time around 2.5 seconds), as can be seen from Figure 7.4(b).

On July 31, 2013, Akamai successfully crushed another DDoS attempt as part of Operation Ababil's phase 4 plan(Figure 7.6). The attack are slightly more intensive than the last example, with traffic size greater than 4.4 million requests/min. More

(a) Traffic spike due to DDoS attack.



(b) Origin performance.

FIGURE 7.6: Phase 4 DDoS attack

than 3000 bots are exploited. The attack targets many marketing webpages such as "details.do" as well as the DNS infrastructures. The attack immediately triggers alert of the live security monitor, at the attention of which Akamai promptly exchanged intelligence with affected customers, harvested and blocked new bot IP addresses upon approval from the customers. The result is zero negative availability or performance impact on the customers' origin server by the attack. Average response time remains 2 seconds before, after and throughout the attack.

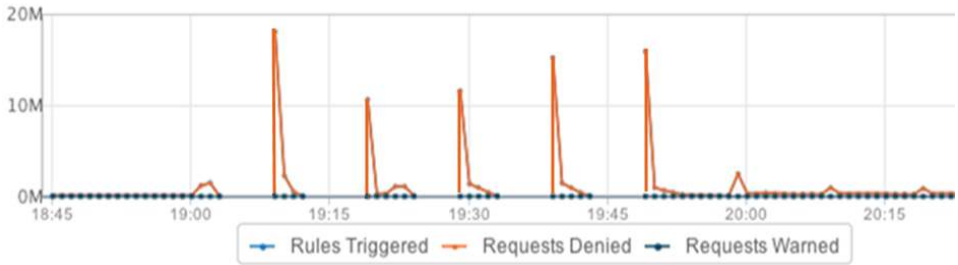


FIGURE 7.7: Observed probes employed by Operation Ababil.

The tactic of pre-attack reconnaissance employed by QCF described in Sec-

tion 7.2.1 has also been recognized in Akamai's security monitor. As shown in Figure 7.7, the short burst high speed probes used by attackers to test vulnerability of the site leaves several sharp peaks in the graph. If a site falters to the probes, QCF announce that they will attack the financial institution and return later with a full scale attack. If the site is resilient, they will move on and start probing others.

7.2.3 Lessons Learned

Operation Ababil and Akamai's observation leave us with invaluable experience to fight cyber attacks of new era. First of all, due to recent attack sizes, infrastructure capacity build out is no not economical, and may not work any more, a cloud-based security layer is critical.

Second, the burst speed of attacks has become too fast for reactive mitigation. Besides, the burst probe example tells us that quick reaction and not exposing vulnerability can prevent site from larger scale of attacks from hacktivists. Therefore, a proactive "always-on" defense is absolutely necessary.

Third, as Operation Ababil has evolved to focus more on layer 7 attack techniques, network layer defenses along are no longer sufficient. It is important for firewalls to have full visibility in to application layer request structure for accurate attack identification. Besides, balcklisting known IP attackers is not enough and cloud-based rate control needs to be established against layer 7 flooding attacks.

Fourth, evidence show that a majority of attack traffic are targeted towards login pages of bank customers. It is a good practice to combine rate controls with request validation strategies and edge validation offload.

Last but least, many of the assaults we observed so far are utilizing modified query string to "break" CDNs' caching and thus requiring CDNs to forward requests to the origin. It is crucial for CDN to ignore query strings that attached to a static resource.

7.3 System Design

To withstand challenges posed by modern cyber attacks such as the Operation Ababil and follow security considerations summarized in Section 7.2.3, we present a typical "always-on" defense solution that can be deployed on most of today's CDN security overlays.

7.3.1 Design Goals

The system is designed to provide the following functionalities:

1. Cloak origin servers from attackers and offload as much attack traffic from them as possible.
2. Offer fine-grained, customer specific rate control ability.
3. Detect and deflect malicious application level traffic.
4. Provide "always-on" protection and real-time defense, instead of analyzing for attacks offline or taking mitigation measures after performance degradation has been observed.

The first point sets the utter goal of the system. The second and third point specifies the network and application layer protection the system must provide, respectively. The last point suggests that the system must provide live service. Besides the functional features above, the system must also meet the following operational requirements by the content providers.

1. Flexibility to adjust defense policies per customer.
2. Ability to apply patches for new web site vulnerabilities.

In other words, the customers using the security product must have full control and freedom to adjust and formulate security measures in accordance with conditions specific to their own sites.

7.3.2 System Overview

The overlay network sit between end users and origin servers maintained by content providers to provide protection and offload attack traffic. Together with the CDN's account managers, a content provider can retrieve real-time security report and control firewall policies through its management portal. Figure 7.8 illustrates the high-level interaction between the security solution and different parties.

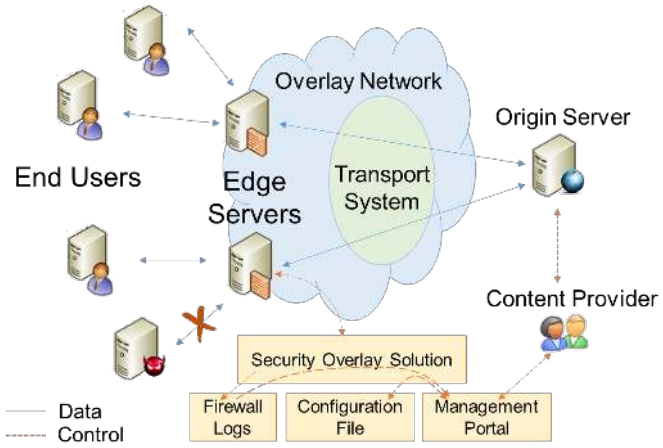


FIGURE 7.8: Overview of a security overlay solution.

To quickly filter out Internet noise and crude network layer attacks with minimum CPU consumption, a simple firewall is configured on all edge servers as a first defense line. The firewall acts on basic IP table rules and drops all traffic towards non-service ports. It can mitigate attacks that do not target service ports such as 40 and 443. SYN cookies (Bernstein, 1996) are also enabled on the edge servers to provide coarse SYN flood protection.

In order to deter more advanced attack, another firewall is set up on the applica-

tion layer. Instructed by a predefined rule set, the firewall inspects various fields in both HTTP requests to identify potential attacks before they are served. Customers have two options for the CDN to handle HTTPS request. In the first option, the CDN simply acts as a proxy and forward encrypted packets to the origin. the CDN is working in limited mode in this option and perform only simple tasks such as rate control and IP blacklisting. In the other option, the customer allow the CDN to hold a private key for SSL encryption between the client and the edge server. The private key does not need to be the same as the one held by the customer. With this option, the firewall is able to provide full protect for HTTPS requests as well.

Rules are basic units in the firewall to define characteristics of an attack. Each rule is a set of preconditions on request messages. For example, an SQL injection (SQLi) rule could be defined as violated if quote marks are discovered in the argument values of a query string. The precondition specified by a rate limiting rule could be that the same IP address send more than 50 ordinary requests or 10 requests with extremely large payload size within any 1 second's period. Note that there could be tens of rules used to define the same attack type (e.g., SQLi), and preconditions used in different rules may overlap. A classification of rules adopted by the firewall is described in Section 7.3.3.

Violation of a rule does not necessarily assures an attack, but adds suspicion to the request in concern. Each rule is associated with an anomaly score and an attack category such as SQLi, XSS, or protocol violation. Within each category, a total score is obtained by summing up the anomaly scores of all violated rules. If the total exceeds a predefined threshold, the request is then classified as an attack. Otherwise, the request is simply logged for future reference and continue to be served by the edge. Multiple thresholds are defined for different categories of attack and their values are specified in the customer's configuration file.

Once an attack is identified, an *action* needs to be taken against it. If the action

is to *alert*, the CDN server will not deny the request, but only generate an alert and continue processing the HTTP request. If the action is to *deny*, the request will simply be dropped, resulting in an HTTP 403 response.

In order to adopt security measures that best suit its site, each content provider can supply a *configuration* file that defines a set of *policies*. A policy specifies rules to be checked, criteria to define an attack and action to take once the attack is identified. Attack definition criteria can take different forms. The customer can define violation of certain rules as an attack disregarding the anomaly score. It can also take advantage of score testing system and supply thresholds explained before. For example, a customer can define attacks as requests that violate any rule described in the configuration file. Alternatively, it can define them as requests that have more than an SQLi score of 20 or a cross site scripting (XSS) score of 30.

A policy is mapped to target resources by *policy matching conditions*, which are based on path, filename and file extension of the requested resource. This allows content providers to have fine grained security control on different categories of their contents, and complete flexibility to make policy adjustment and exceptions.

Besides the layer 3 and layer 7 firewalls, the security overlay can also include some other features such as DNS infrastructure defense, and connectivity constraint enforcer. The discussion about these components is beyond the scope of this dissertation.

7.3.3 Firewall Rules

A typical application firewall rule set maintained by the CDN are consisted of three major components, namely the core rule set (CRS), rate control rules and custom rules. The CRS covers a wide range of common attacks such as **SQLi**, **XSS**, **Trojans** on top of:

Bad Robots: Requests generated by attack tools such as HULK, DirtJumper that

do not follow web crawling rules.

Protocol Anomalies: Empty or missing important fields such as *Host*, *User-Agent* in HTTP header.

Protocol Violations: Requests inconsistent with HTTP protocol such as having a body content with GET or HEAD method, not supplying *Content-Length* with POST method.

HTTP Policy: Some customers may post special constraints on HTTP requests as part of their site policy, such as not allowing accessing URL with certain extension, forbidding certain HTTP method, or requiring HTTP/1.1.

Request Limits: Restriction on length and size argument names and values in the query string.

Outbound: Information or source code leakage from the edge or origin server.

Miscellaneous: Unclassified attacks such as PHP code injection, Unix system command injection, HTTP response splitting attack.

Rate control rules perform two functions. First, it blocks requests from geographies where the customer provides no service. Second, it blacklist IP based on known intelligence and rate limit policies. These rules are the main fighters against DDoS attacks.

The custom rules are formulated by two sources. On one hand, the CDN's security group may consistently investigate newly emerged hack tools, recent attack attempts and release hot fixes by laying out new custom rules as "virtual patches". On the other hand, customers are also able to formulate their private security rules specific to their own site.

7.4 Measurement Study

In this section, we analyze the traffic blocked by an example implementation of system described in Section 7.3 — Akamai's "Web Application Firewall" (WAF).

The core rule set of WAF are developed on the base of ModSecurity's CRS (Ristic, 2010) with adjustments and extensions.

7.4.1 Data Set

The data used for this study is taken from WAF logs in April 2014. Each log entry is a record of client request that violates one or more WAF rules. Major information stored in each entry include:

1. Timestamp.
2. Source IP, port and geolocation information.
3. Target customer and host name. The customer can be further mapped into *Akamai vertical* and *market segment*, which are classification of content providers by their industries. Market segments are general industries such as media & entertainment, commerce, and high tech, while Akamai vertical provides finer grained division such as gaming, media, hotel & travel, automotive and software. There are a total of 7 market segments and 21 Akamai verticals touched by the data set.
4. HTTP headers such as User-Agent, HTTP Version, Referrer.
5. Rules violated. For each violated rule, the suspicious part of request is dumped. A detailed violation reason is also provided such as "argument value contains restricted string 'and'" or "IP 12.34.56.78 exceeds 38 message units/sec".

On a typical day, around 30 million requests are blocked and each request triggers 1.5 rules on average. Table 7.1 shows a break down of rules triggered in each risk

[1] Include protocol anomaly, protocol violation, HTTP policy and request limits.

[2] Mainly custom rules.

Table 7.1: Number of WAF rules triggered during a day.

Rule category	# Requests denied	# Distinct IPs blocked
SQLi	428,274	73,003
XSS	35,013	15,424
Bad robots	70,116	520
Invalid HTTP[1]	1,787,654	642,032
Outbound	5,213,408	68,357
Rate limit	8,736,078	13,649
IP/Geo block	16,205,770	228,181
Others[2]	12,074,042	584,441
Total	30,489,702	1,468,913

group on April 30th, 2014. A request or a source IP can show up and thus be tallied in different risk groups, resulting in a total smaller than the sum of all above rows.

According to a recent press release (WAFAccuracy, 2011), the WAF rules are able to achieve an 96% accuracy based on MCC scoring, which is high enough to assume that most requests recorded in our data set are real attacks.

During the study, we found that different rule category exhibits very distinct characteristics. Due to space limitations, the rest of this section focuses on four of the most popular attack types in today's Internet: SQLi, XSS, bad robots, and rate limit, the last of which is a good proxy for and therefore used interchangeably with DDoS attacks in this chapter.

7.4.2 Noises

During this study, we manually inspected around 100 attack interception records to verify the authenticity and nature of each attack. Different attack records can be snapshots of the same wave of attack. The records we picked come from two sources:

1. Randomly sampled records from different rule categories.

2. Records pertaining to periods, customers or attack origins that demonstrate unusual patterns (such as traffic burst period, content provider being persis-

tently attacked, etc.)

While the majority of records we investigated looks like illegitimate traffic, we cannot confirm authenticity of many records due to lack of request message body and understanding of its purpose. However, we are able to verify the part of the log that does not constitutes attacks. We classify the non-attack records into the following categories.

False positives: False positives are legitimate requests that are expected in a customer's website normal operation but mistakenly intercepted by WAF. Sometimes they are results of paranoid or unsound rules. For example, there is a rule defining "%0a" and "%0d" in HTTP header as hints for HTTP response splitting attack, while these ASCII characters can be found in various types of normal cookies. As another example, there is a customer who banned requests of any resources that contain ".cookie" out of concern about XSS attacks. However, he did not realize his website relies on "jquery.cookie.js", a javascript of jQuery plugin. As a result, all his end-users are blocked by WAF from accessing that file and thus unable to utilize certain jQuery features.

Other times false positives can be caused by situations specific to certain web pages. For example, we discovered that an ASP webpage of a customer has been alarmed as an XSS attack target by a large number of origins with unusual frequency. The WAF log reported that the request bodies contain various keywords such as "substring", "alert", "javascript" that are highly correlated with XSS attacks. A closer examination of the ASP page reveals that the dynamic web page is actually deployed by the customer to collect bugs on their website and the users are submitting faulting javascripts. Not surprisingly, without knowledge about the ASP's use cases, WAF thinks the clients are uploading scripts to compromise another website and prevented these "malicious" attempts without hesitation. The poor customer never

had a chance to receive the bug report and thus fixing faulty web pages.

Akamai keeps tracing and fixing false positives by loosing unnecessary rule conditions or setting up policy exceptions for specific customers.

Customer misconfiguration: Customer misconfiguration are caused by buggy codes or faulty website configuration on the content provider's side. For example, we observed that a vast number of source IPs are requesting a web page containing query strings whose values recursively refer to the URL itself. The requests are classified as XSS attempts and blocked by WAF. However, it is not caused by malicious clients but rather by a javascript bug on the customer's website which generates the recursive URL. It is appropriate for Akamai to block these malformed requests but blacklisting source IPs are not necessary.

Crawler activities: Despite Akamai's efforts to let through certain well-known legitimate web crawler, we are still able to see a considerable number of their activities in the WAF log, most in the bad robots category. By reverse DNS the source IPs, we are able to compile a list of index spiders. On the top of list is Google, followed by a few telecommunication companies.

7.4.3 Repeated Attackers

We first investigate how patient attackers are. In order to answer this question, we picked a two-week period and stores all IP address that appear in the source field for each during the first day. Starting from the second day, we count the number of source IPs that appeared the day before and the number of IPs that are ever seen in the previous days. The measurement study is done for each separate rule category. Because the graphs of different categories are similar, we only show our results for SQLi in Figure 7.9.

The figure shows that there is a considerable amount of attackers who participated in more than a day's assault. The percentage of attackers that has ever seen before

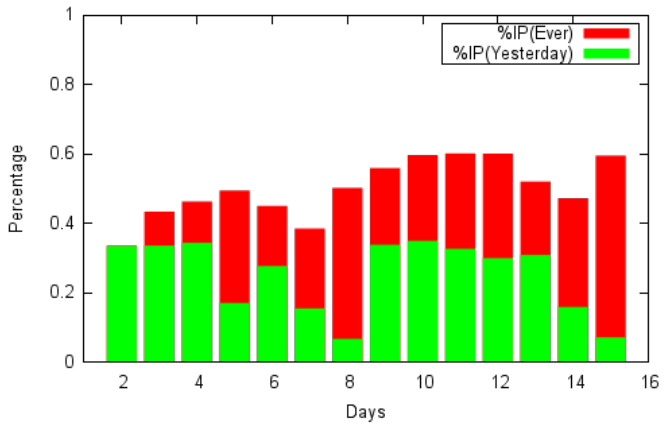


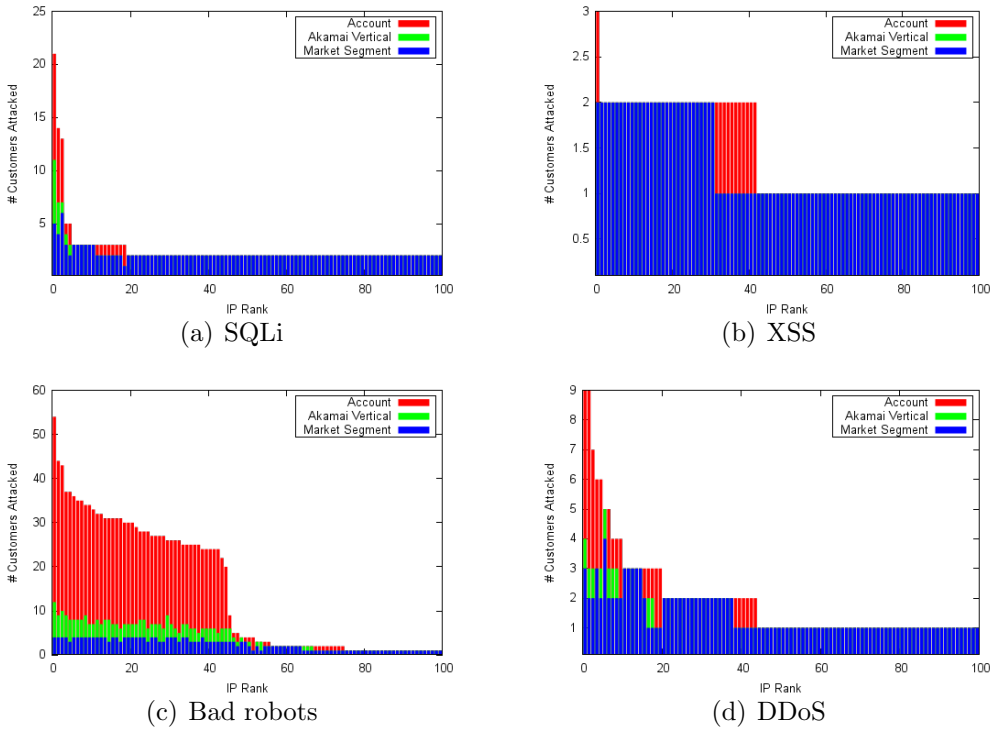
FIGURE 7.9: Repeated SQLi attackers during two weeks. One bar stands in front of the other.

converges at around 50% as more days are measured. This supports the effectiveness of IP blacklisting even with the advancement of cyber attack techniques today. The number of IPs that participate in two consecutive day's attacks floats around 20% of total IP address population.

7.4.4 Attack Pervasiveness

One interesting aspect of attacks that we have special advantage to learn from a CDN's perspective is their outreach and pervasiveness. To partly alleviate DHCP effect, we aggregate user's IP address into /24 subnets. The subnets are then sorted by the number of customer accounts they attacked during a day and presented in Figure 7.10.

As can be seen from the graph, the number of targeted accounts varies across different rule categories. Each bad robots attack targets a wide range of customers probed by the attacker, which is expected by its definition. The SQLi category characterized by a few attacks extensively exploring vulnerability on the Internet and the majority concentrated on a single site. DDoS and XSS attacks are similar in their dedication to paralyze very few victims.



(a) SQLi (b) XSS

(c) Bad robots (d) DDoS

FIGURE 7.10: Number of targeted customers per /24 subnet

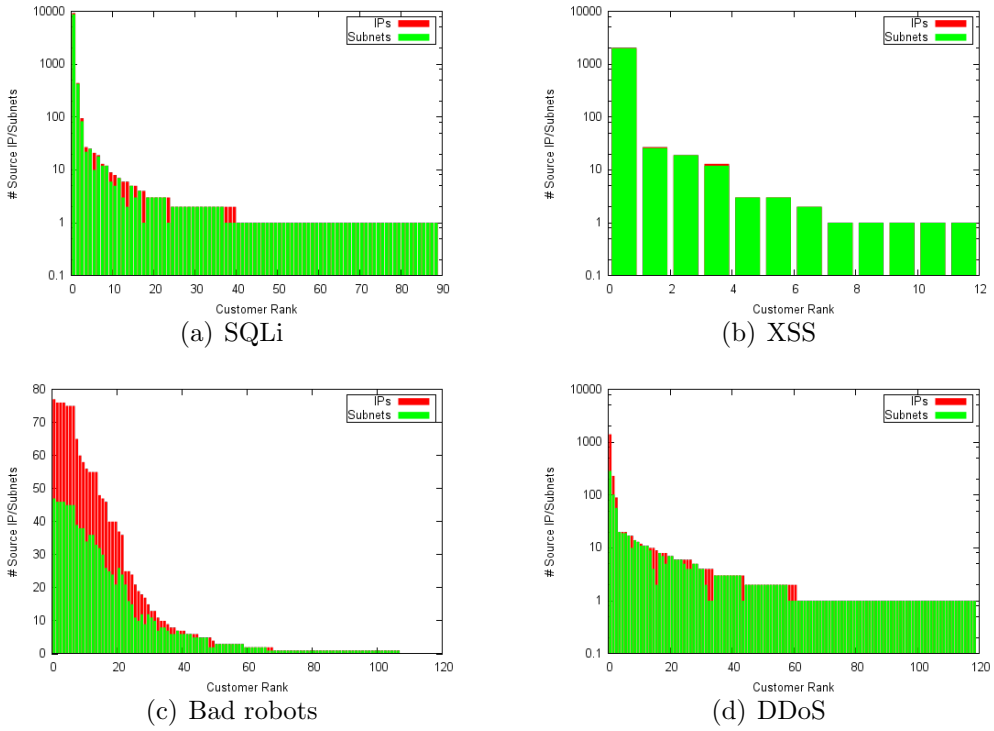
It is also worth noticing from the graph that regardless of extensiveness of attacks from the same subnet, they are usually directed towards the same industry. This means a customer can potentially gain extra protection value from CDN by raising site security level when it is alerted of attacks towards its peers.

7.4.5 Attack Origins

Another interesting fact one might be interested to learn is the other way around, i.e. how many machines or botnets are involved in an attack towards a single site. Lacking more supporting evidence to distinguish attacks waged by different entities, we make a crude assumption that all malicious requests towards the same customer during a 60-minute window are originated from the same attacker.

Under this over-simplistic assumption, we counted the maximum number of different origin IP addresses and /24 subnets observed for each customer during a day.

Figure 7.11 shows all customers affected by each of the four rule categories, ranked by number of sources tallied in the above mentioned way.



(a) SQLi (b) XSS

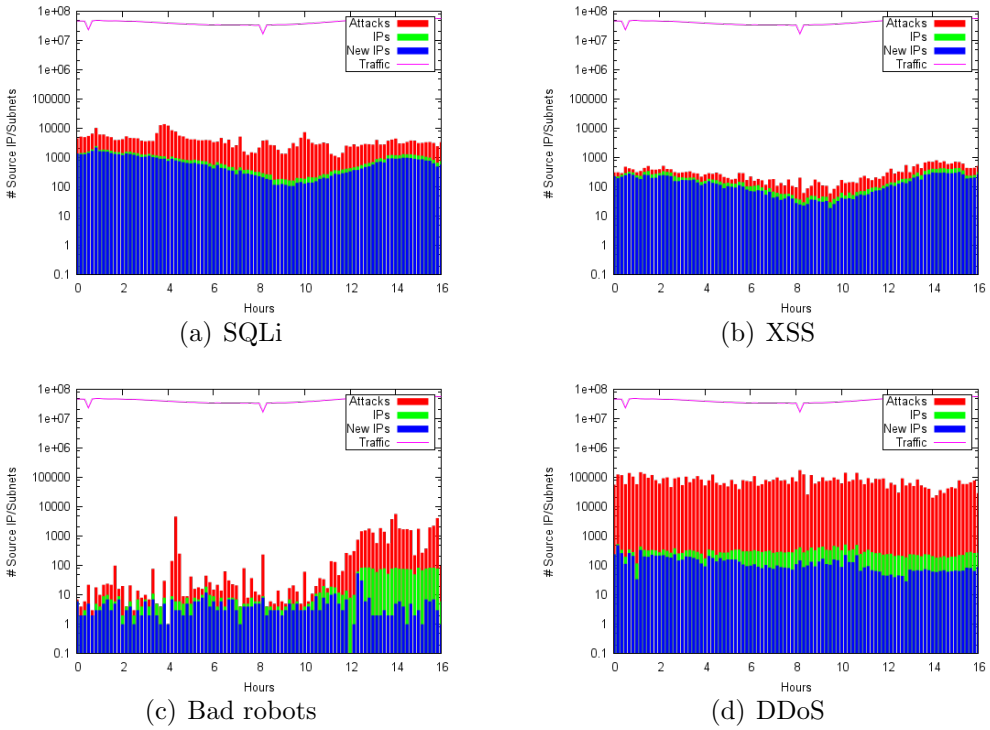
(c) Bad robots (d) DDoS

FIGURE 7.11: Number of maximum hourly attack sources for each customer. Note that the vertical axis is in log scale except for bad robots category.

Reading from the graph, each rule category have some customer targeted by a vast number of machines during the peak hour. Top customers from both SQLi and DDoS categories have received attacks from more than 1000 IP addresses during an hour. However, there are still around half of the customers in each category face attacks from no more than one IP addresses each hour, suggesting distributed attacks do not dominate all attack traffic. Last but not least, it is observed that bad robot attacks usually utilize multiple machines within the same subnet, probably due to web crawling activity from the same data center.

7.4.6 Attack Size

In Figure 7.12, we tallied all attack requests in each 10-minute window during a day. We present three overlapping bars to denote the number of new IPs, total number of IPs and number of requests inside each window. New IPs are defined as IP addresses that involve in the attack during the time window, but have never been seen before during the day. The total traffic, including both good requests and the ones blocked by WAF, are also provided in the graph as a reference curve.



(a) SQLi (b) XSS

(c) Bad robots (d) DDoS

FIGURE 7.12: Attack volume during a day.

By its nature and definition, DDoS attack are characterized by huge traffic originating from each source. To the opposite is XSS, where each IP address only makes one attempt during 10 minutes' period. It is also noticed that SQLi attack has the most IP addresses involved, with each making around 5 attack attempts before quiting.

The figure also suggests that attacks from same IP usually last less than 10 minutes, as the "new IPs" bars almost overlap the "IPs" bars. The only exception is bad robots, where an IP blocked by WAF can come back later in the day. DDoS attacks somewhat stand between the two, with the majority IP addresses seen in each time window being new ones and minor revisiting source IPs.

7.5 Discussion

Based on the conclusion from our observation of Operation Ababil and measurement study of WAF, this section discusses potential real-time attack mitigation strategies that could be incorporated into CDN security overlays.

7.5.1 Authentication Data Pre-Screening

One of the lessons we learned in Section 7.2.3 is that rate controls needs to be complemented by request validation strategies edge validation offload in order to combat the tendency of attack focus on various customers' login pages. The validation provided by firewalls so far are mostly upon traffic patterns. Very loose rate limiting can be applied in fear of blocking good clients. Therefore, a smart attacker can issue moderate amount of legitimate login requests with randomly guessed user names and passwords from a vast botnet to penetrate the CDN's shield and overload the origin server's bandwidth and computation resource.

Pre-screening of authentication data from CDN's side comes with no trivial solution due to two major constraints. On one hand, according to some CDN insiders, most banks do not wish to impose any efforts on their clients during the whole authentication process. Challenge-response test systems such as CAPTCHA will deteriorate user experience. Proof-of-work schemes such as speak-up (Walfish et al., 2006) and Portcullis (Parno et al., 2007b) may cause public discomfort about using the bank's service, when their competitors do not require this significant extra

consumption of end users' hardware resource. On the other hand, customers are cautious about potentially revealing their clients' credentials or commercial secrets, and therefore reluctant to share their private key or user database with the CDN.

In response to these requirements, we propose here a method for CDN to inspect authentication information in the login requests and offload unauthorized ones from the origin server. The scheme works for both HTTP and the second connection establishment model of HTTPS as discussed in Section 7.3.2, where the CDN uses its own private key to establish a secure connection with the end user and can see any message communicated between the two parties.

In order to anonymize user credentials from CDN, a javascript is embedded in the login web page to encrypt authentication data with the bank's public key using asymmetric cryptography such as RSA, before submitting the request. In the meanwhile, the bank provides CDN with a list of all possible cipher texts that could contain legitimate identities. The list can be exchanged on a daily basis is not required to be up-to-date all the time. When the CDN receives the request, it performs an integrity check as the system described in Section 7.3.2 currently performs. Malformed requests are directly dropped. The cipher text is then matched against the list compiled by the bank as the pre-authentication step. If a match is found, the request is forwarded to the origin for actual authentication. Otherwise, the request will be dropped to a low priority channel with other requests who failed the cipher text check, which is under stringent rate limiting control.

When the request arrives at the origin server, the bank decrypts the data and attaches an authentication token to the cookie if the credentials are valid. If the previous pre-authentication is a failure and the CDN sees the authentication cookie, it will add the cipher text onto the list. Once a user holds the authentication token, he can freely communicate with the site with a loose rate limiting policy by the CDN. If the result is negative, the bank respond with an authentication failure. The CDN

will also be able to see the response and remove the credentials' cipher text from the list if the pre-authentication is a success.

7.5.2 Client Reputation Accounting

As suggested in Section 7.4.3, a considerable percentage attackers will reappear days whether they are successful or not. The case study in Section 7.2.1 also showed the tendency of hacktivists to partition their attack activities with intervals of dormancies.

While IP blacklisting is effective against certain repeated attackers, CDNs have to use this tool with extreme caution and only in the case of absolute certainty. Permanently blocking an innocent IP addresses by accident means unavailability and lose of a customer for the content provider. On the other hand, if an attacker has successfully caused performance degradation on the edge or origin server but left uncaught, even if the rate limiting control and various WAF rules comes into play and block the source temporarily, he may come back after a long period dormancy and inflict the same damage again.

These observations and concerns motivate a design of system to keep track of misbehaving end users. The design proposed here are based on the existing client reputation studies and functions in a way similar to FortiGate (2013). Both IP address and device fingerprinting can be used as client identity. A separate profile is maintained for each suspicious end user. We define certain types of attack contexts such as bots, DDoS'ers, and scanners. Each violation contribute penalty scores towards one or more such categories in the client's profile according to the configuration file and based on the rules triggered. The score testing system described in Section 7.3.2 will also help decide the penalty points accrued.

Besides the above mentioned sources, some observations during our manual inspection of WAF data set will also help decide likelihood of a real attacker. For

example, we noticed that the same person triggering a rule many times with different payloads has a high chance to be a real attacker; and if the he targets multiple sites, the chance is even higher. As another example, sending PHP pages to a site that does not run PHP at all is a clear clue of an attack.

After accumulating knowledge about a client based on his behaviors, a more complete picture is formed on his profile. The CDN and content providers can then make informed decisions and impose more appropriate policies on the client based on his intent and evilness exhibited from the profile.

7.5.3 Security Analysis

We start with analyzing security properties with design in Section 7.5.1. In order for attackers to penetrate edge server and reach origin server, the attacker need to included encrypted credentials in the login request to pass integrity check; otherwise the package is dropped by WAF.

If the attacker does have an account with the protected customer, either by stealing or legally obtaining it, any anomalies in the future can be associated with the account, which is then at the bank's disposal. Moreover, the attacker will not be take advantage of multiple botnet nodes to saturate origin server's capacity because simultaneous login into the same account from multiple location is correlated with identity theft and will likely cause freezing of the account, preventing further requests from getting through freely.

If the attack does not have a valid account, the request will arrive origin at the rate specified by the rate limiting policy of the low priority channel, which is supposed to be set too low to make any observable impact on origin's performance.

We now discuss the case where the list of credentials held by the CDN is slightly out-of-date. There are two types of anomalies that can happen. First, a credentials' entry can be in the CDN's list but not in the bank's. This can only be cause by

the bank's modifying or deleting an account. In the first case, the bank will still be able to trace down the misbehaving account as long as it maintains a history. Under both circumstances, the bank will issue an authentication failure in the response, at the notice of which CDN will remove the corresponding entry and further visit with those credentials will be put on rate limit. Second, an entry can be in the bank's database but no the CDN's. In this case, the account is valid and should not be on rate limit channel in the future. This is realized by CDN's observation of the authentication token, which is issued by the bank upon successful authentication.

Throughout the whole process, the CDN only sees credentials encrypted credentials by an asymmetric public key, the authentication tokens and the failure notice. The authentication result is not a privacy for the bank as long as the CDN cannot figure out the credentials. The irreversibility of client credentials and the bank's private key from cipher texts the CDN sees is guaranteed by proper choose of encryption algorithm and the property of asymmetric cryptography.

Next, we prove the viability of the client reputation accounting system. The score used by the system is in its essence an evaluation of the accumulated damage a client has attempted to inflict the edge and the origin server with, be it done during a single-shot burst, steady attack over time, or probes interspersed with dormant periods as in Operation Ababil. Therefore, the hacktivist cannot hide his intent by maneuvering the timing of his attacks. He can neither manipulate the attack strength of each attack such as splitting sensitive query strings into two requests to explore the vulnerability of a site without exceeding the sensitivity threshold currently defined by WAF, because the two scores are still credited towards his profile despite they are not classified as attacks individually. The only exploit that would succeed is assaults that do not trigger any existing rules or have undervalued scores compared with actual damage they cause, which is technically difficult and will diminish with the everlasting adjustment and improvement of rules.

7.6 Conclusion

In this chapter, we have studied Operation Ababil, a most recent large-scale attack series towards financial institutions, and its impact on Akamai CDN. Our observation show that the CDN is able to offload the majority of attack traffic during each phase of the attack series and leave origin servers' availability and performance untouched. We then follow up with a design of general security overlay solution to shield content providers.

Our study of four popular attacks recorded by a commercial CDN firewall product shows that attackers have a tendency to revisit a site even if his first attempts are blocked. We noticed several unique attack patterns between different attack types. Based on these insights, we presented two components to enhance the existing CDN security mechanisms and keep DDoS traffic further away from origin servers.

8

Conclusion

In this dissertation, we have approached content delivery, one of the most important component of today's Internet, from three qualitative angles, namely its efficiency, reliability and security.

We first investigates three means to improve content delivery's cost efficiency. The first way is through hybrid CDN. We examined a large commercial hybrid CDN and show how such architectures can achieve better performance with the same amount bandwidth consumption of infrastructure, by offloading high percentage of traffic to the peers. We also cleared a previous concern about hybrid architectures' impact on inter-ISP traffic by showing that the cross AS traffic is balanced even with hybrid CDN.

The second way to achieve more efficient content delivery is through CDN-ISP collaboration, which enables in-network server allocation and informed user-server assignment. We argued that most of the existing content delivery architectures can benefit from these two capabilities offered by CDN-ISP collaboration.

The third way to efficient content delivery is the concept of information-centric network. We summarized the main benefits of ICN as well as its fatal weakness —

the requirement of significant network change. By close examination and trace-based simulation, we discovered that the benefits claimed by many ICN architectures can still be achieved without going through the proposed major change to network. Especially, pervasive caching and nearest-replica routing are not fundamentally necessary.

We then turned to reliability problems introduced by the above emerging content delivery trends. We used an actual attack to demonstrate potential severity of accounting reliability problems in hybrid CDNs. We then suggested peer review based system to address these challenges. Our evaluation show that the system can discover all peer misbehaviors in a hybrid CDN with moderate overhead.

Finally, we studied the security challenges faced by today's Internet and how CDN can mitigate overwhelmingly large attack traffic before it reaches origin. We start by presenting a CDN's perception of a famous DDoS attack series and show the importance of an "always-on" security overlay. We then conduct a measurement study of a commercial security solution deployed on overlay networks and proposed two enhancement schemes based on the results of measurement study.

Appendix A

Pseudo-Code for Tamper Evident Log

```
 1: pso,pro,psi,pri : set                                   ▷ Previous hash value
 2: pending : vector of list of (seq, msg)                  ▷ Pending msgs
 3: sendqueue : vector of list of msg                       ▷ Send queue
 4: lsa,lra : vector of (seq, hash, sig)                    ▷ Authenticators from clients
 5: log : list of (h, s, t, c)                              ▷ Log
 6: top-hash : hash                                         ▷ Current top of the hash chain
 7: top-seq: int                                            ▷ Most recent sequence number
 8: function LOG-APPEND(t, c)
 9:     (top-seq, save) ← (top-seq + 1,top-hash)
10:     top-hash ← H(save || top-seq || t || c)
11:     log ← log + (top-hash, top-seq, t, c)
12:     return (save, top-hash, top-seq)
13: function SEND(m, i, j)
14:     sendqueue[j].APPEND(m)
15:     MAYBE-SEND-NEXT(j)
16: function MAYBE-SEND-NEXT(j)
17:     if not sendqueue[j].empty and |pending[j]| < max then
18:         m ← sendqueue[j].FIRST
```

```
19:         (h,pso[j],s) ← LOG-APPEND(SEND, j || m || pso[j])
20:         NET-SEND(j, SEND || m || h || s || σ_i(s || pso[j]))
21:         pending.APPEND(s, m)
22: function RECV(m, i, j)
23:     if m is SEND || x || h_1 || s || sig then
24:         h ← H(h_1 || s || SEND || x || pso[j])
25:         if VALIG-SIG(sig, seq || h, j) then
26:             (lra[j], psi[j]) ← ((seq || h || sig), h)
27:             (h_2,pro[j],s_2)←LOG-APPEND(RECV,j || x || pro[i])
28:             NET-SEND(j, ACK || s || s_2 || h_2 || σ_i(s_2 || pro[j]))
29:     else if m is ACK || s || s_2 || h_3 || sig_3 then
30:         if ∃mout: pending.first is (s,mout) then
31:             h ← H(h_3 || s_2 || RECV || i || mout || pri[j])
32:             if validsig(sig_3, s_2 || h || sig_3) then
33:                 (lsa[j],pri[j]) ← ((s_2, h, sig_3), h)
34:                 pending.remove(s, mout)
35:                 MAYBE-SEND-NEXT(j)
```

Bibliography

Aditya, P., Zhao, M., Lin, Y., Haeberlen, A., Druschel, P., Maggs, B., and Wishon, B. (2012), "Reliable client accounting for hybrid content-distribution networks," .

Adya, A., Bolosky, W. J., Castro, M., Cermak, G., Chaiken, R., Douceur, J. R., Howell, J., Lorch, J. R., Theimer, M., and Wattenhofer, R. P. (2002), "FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment," in *Proc. OSDI*.

Agarwal, S. and Lorch, J. R. (2009), "Matchmaking for online games and other latency-sensitive P2P systems," in *Proc. SIGCOMM*.

Ager, B., Mühlbauer, W., Smaragdakis, G., and Uhlig, S. (2011), "Web Content Cartography," in *IMC*.

Ager, B., Chatzis, N., Feldmann, A., Sarrar, N., Uhlig, S., and Willinger, W. (2012), "Anatomy of a Large European IXP," in *SIGCOMM*.

Ahlgren, B., D'Ambrosio, M., Dannewitz, C., Eriksson, A., Golić, J., Grönvall, B., Horne, D., Lindgren, A., Mämmelä, O., Marchisio, M., Mäkelä, J., Nechifor, S., Ohlman, B., Pentikousis, K., Randriamasy, S., Rautio, T., Renault, E., Seittenranta, P., Strandberg, O., Tarnauca, B., Vercellone, V., and Zeghlache, D. (2010), "Second NetInf Architecture Description," `http://www.4ward-project.eu/`.

Ahlgren, B., Dannewitz, C., Imbrenda, C., Kutscher, D., and Ohlman, B. (2012), "A survey of information-centric networking," *Communications Magazine, IEEE*, 50, 26 –36.

Akamai (2014), "Facts & Figures," `http://www.akamai.com/html/about/facts_figures.html`.

Anderson, T., Roscoe, T., and Wetherall, D. (2004), "Preventing Internet denial-of-service with capabilities," *ACM SIGCOMM Computer Communication Review*, 34, 39–44.

Antoniades, D., Markatos, E., and Dovrolis, C. (2009), "One-click Hosting Services: A File-Sharing Hideout," in *IMC*.

Aranda, P. A., Zitterbart, M., Boudjemil, Z., Ghader, M., Garcia, G. H., Johnsson, M., Karouia, A., Lazar, G., Majanen, M., Mannersalo, P., Martin, D., Nguyen, M. T., Sanchez, S. P., Phelan, P., Ponce de Leon, M., Schultz, G., Sllner, M., Zaki, Y., and Zhao, L. (2010), "Final Architectural Framework," `http://www.4ward-project.eu/`.

Arianfar, S., Koponen, T., Raghavan, B., and Shenker, S. (2011), "On preserving privacy in content-oriented networks," in *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, ICN '11, pp. 19–24.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. (2009), "Above the Clouds: A Berkeley View of Cloud Computing," UC Berkeley Technical Report EECS-2009-28.

Barreno, M., Nelson, B., Sears, R., Joseph, A. D., and Tygar, J. D. (2006), "Can machine learning be secure?" in *Proc. AsiaCCS*.

Bazzi, R. A. and Konjevod, G. (2005), "On the establishment of distinct identities in overlay networks," in *PODC*, pp. 312–320.

Beaver, D., Kumar, S., Li, H. C., Sobel, J., Vajgel, P., and Inc, F. (2010), "Finding a Needle in Haystack: Facebook's Photo Storage," in *In Proc. of OSDI*.

Bernstein, D. J. (1996), "SYN cookies," .

Bhagwan, R., Savage, S., and Voelker, G. M. (2003), "Understanding Availability," in *Proc. IPTPS*.

Breslau, L., Cao, P., Fan, L., Phillips, G., and Shenker, S. (1999), "Web caching and Zipf-like distributions: evidence and implications," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*.

CCNIndustry (2012), "Emerging Network Consortium Brings Industries Together to Innovate with Content-Centric Networking (CCN)," `http://www.mach.com/en/News-Events/Press-Room/Press-Releases/Emerging-Network-Consortium-Brings-Industries-Together-to-Innovate-with-Content-Centric-Networking-CCN`.

Cha, M., Rodriguez, P., Crowcroft, J., Moon, S., and Amatriain, X. (2008), "Watching television over an IP network," in *Proc. IMC*.

Chan-Tin, E., Feldman, D., Kim, Y., and Hopper, N. (2009), "The Frog-Boiling Attack: Limitations of Anomaly Detection for Secure Network Coordinates," in *Proc. SecureComm*.

Chandola, V., Banerjee, A., and Kumar, V. (2009), "Anomaly detection: A survey," *ACM Comput. Surv.*, 41, 15:1–15:58.

Cleary, J., Donnelly, S., Graham, I., McGregor, A., and Pearson, M. (2000), "Design Principles for Accurate Passive Measurement," in *pam*.

Cohen, B. (2003), "Incentives Build Robustness in BitTorrent," in *P2PEcon Workshop*.

Denning, D. E. (1987), "An intrusion-detection model," *IEEE Trans. on Software Engineering*, 13, 222–232.

Dhungel, P., Ross, K. W., Steiner, M., Tian, Y., and Hei, X. (2012), "Xunlei: Peer-Assisted Download Acceleration on a Massive Scale," in *PAM*.

Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R., and Weihl, B. (2002), "Globally Distributed Content Delivery," *IEEE Internet Computing*, 6, 50–58.

DiPalantino, D. and Johari, R. (2009), "Traffic Engineering versus Content Distribution: A Game-theoretic Perspective," in *INFOCOM*.

Dischinger, M., Haeberlen, A., Gummadi, K. P., and Saroiu, S. (2007), "Characterizing Residential Broadband Networks," in *Proc. IMC*.

Dobrian, F., Awan, A., Stoica, I., Sekar, V., Ganjam, A., Joseph, D., Zhan, J., and Zhang, H. (2011), "Understanding the Impact of Video Quality on User Engagement," in *SIGCOMM*.

Douceur, J. R. (2002), "The Sybil Attack," in *Proc. IPTPS*.

Fayazbakhsh, S. K., Lin, Y., Tootoonchian, A., Ghodsi, A., Koponen, T., Maggs, B., Ng, K., Sekar, V., and Shenker, S. (2013), "Less pain, most of the gain: incrementally deployable ICN," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pp. 147–158, ACM.

FortiGate (2013), "Monitoring your network using client reputation," http://docs.fortinet.com/d/fortigate-monitoring-your-network-using-client-reputation.

Fotiou, N., Nikander, P., Trossen, D., and Polyzos, G. C. (2010), "Developing Information Networking Further: From PSIRP to PURSUIT," in *BROADNETS*, pp. 1–13.

Frank, B., Poese, I., Smaragdakis, G., Uhlig, S., and Feldmann, A. (2012), "Content-aware Traffic Engineering," *CoRR*, abs/1202.1464.

Freedman, M. J. (2010), "Experiences with CoralCDN: A Five-Year Operational View," in *Proc. NSDI*.

Freedman, M. J. and Morris, R. (2002), "Tarzan: a peer-to-peer anonymizing network layer," in *Proc. ACM CCS*.

Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., and Boneh, D. (2003), "Terra: A Virtual Machine-Based Platform for Trusted Computing," in *Proc. SOSP*.

Gasti, P., Tsudik, G., Uzun, E., and Zhang, L. (2012), "DoS and DDoS in Named-Data Networking," *CoRR*, abs/1208.0952.

Gerber, A. and Doverspike, R. (2011), "Traffic Types and Growth in Backbone Networks," in *OFC/NFOEC*.

Ghodsi, A., Shenker, S., Koponen, T., Singla, A., Raghavan, B., and Wilcox, J. (2011), "Information-centric networking: seeing the forest for the trees," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, pp. 1:1–1:6.

Gill, P., Arlitt, M., Li, Z., and Mahanti, A. (2007), "YouTube Traffic Characterization: A View From the Edge, IMC," in *In: Proc. of IMC*.

Goldenberg, D., Qiuy, L., Xie, H., Yang, Y., and Zhang, Y. (2004), "Optimizing Cost and Performance for Multihoming," in *SIGCOMM*.

Guo, L., Chen, S., Xiao, Z., Tan, E., Ding, X., and Zhang, X. (2005), "Measurements, Analysis, and Modeling of BitTorrent-like Systems," in *Proc. IMC*.

Haeberlen, A., Kuznetsov, P., and Druschel, P. (2007), "PeerReview: Practical Accountability for Distributed Systems," in *SOSP*.

Huang, C., Li, J., Wang, A., and Ross, K. W. (2008a), "Understanding Hybrid CDN-P2P: Why Limelight Needs its Own Red Swoosh," in *NOSSDAV*.

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

Inc., S. (2011), "Global Broadband Phenomena," Research Report `http://www.sandvine.com/news/global_broadband_trends.asp`.

Ioannidis, J. and Bellovin, S. M. (2002), "Implementing pushback: Router-based defense against DDoS attacks," .

Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., and Braynard, R. L. (2009), "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pp. 1–12.

Jacobson, V., Thornton, J. D., Smetters, D. K., Zhang, B., Tsudik, G., claffy, k., Krioukov, D., Massey, D., Papadopoulos, C., Abdelzaher, T., Wang, L., Crowley, P., and Yeh, E. (2010), "Named Data Networking (NDN) Project," `http://named-data.net/techreport/TR001ndn-proj.pdf`.

Jiang, W., Zhang-Shen, R., Rexford, J., and Chiang, M. (2009a), "Cooperative Content Distribution and Traffic Engineering in an ISP Network," in *SIGMETRICS*.

Jiang, W., Zhang-Shen, R., Rexford, J., and Chiang, M. (2009b), "Cooperative content distribution and traffic engineering in an ISP network," in *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, SIGMETRICS '09.

Jokela, P., Zahemszky, A., Esteve Rothenberg, C., Arianfar, S., and Nikander, P. (2009), "LIPSIN: line speed publish/subscribe inter-networking," in *SIGCOMM '09*, pp. 195–206.

Karagiannis, T., Rodriguez, P., and Papagiannaki, K. (2005), "Should Internet service providers fear peer-assisted content distribution?" in *Proc. IMC*.

Kerner, S. M. (2013), "DDoS Attacks: Growing, but How Much?" .

Kohavi, R., Henne, R. M., and Sommerfield, D. (2007), "Practical Guide to Controlled Experiments on the Web: Listen to Your Customers not to the HiPPO," in *KDD*.

Krishnan, R., Madhyastha, H., Srinivasan, S., Jain, S., Krishnamurthy, A., Anderson, T., and Gao, J. (2009), "Moving Beyond End-to-end Path Information to Optimize CDN Performance," in *IMC*.

Labovitz, C., Lekel-Johnson, S., McPherson, D., Oberheide, J., and Jahanian, F. (2010), "Internet Inter-Domain Traffic," in *SIGCOMM*.

Laoutaris, N., Smaragdakis, G., Rodriguez, P., and Sundaram, R. (2009), "Delay Tolerant Bulk Data Transfers on the Internet," in *SIGMETRICS*.

Laoutaris, N., Sirivianos, M., Yang, X., and Rodriguez, P. (2011), "Inter-Datacenter Bulk transfers with NetStitcher," in *SIGCOMM*.

Leighton, T. (2009), "Improving Performance on the Internet," *CACM*, 52, 44–51.

Lewin, D. M., Maggs, B., and Kloninger, J. J. (2006), "Internet Content Delivery Service with Third Party Cache Interface Support. U.S. Patent Number 7,010,578," .

Li, L., Xu, X., Wang, J., and Hao, Z. (2013), "Information-Centric Network in an ISP," `http://tools.ietf.org/html/draft-li-icnrg-icn-isp-01`.

Liu, H. H., Wang, Y., Yang, Y., Wang, H., and Tian, C. (2012), "Optimizing Cost and Performance for Content Multihoming," in *SIGCOMM*.

Liu, X., Yang, X., and Xia, Y. (2011), "NetFence: preventing internet denial of service from inside out," *ACM SIGCOMM Computer Communication Review*, 41, 255–266.

Lu, Z., Wang, Y., and Yang, Y. R. (2012), "An Analysis and Comparison of CDN-P2P-hybrid Content Delivery System and Model," *JCM*, 7, 232–245.

Machiraju, S., Seshadri, M., and Stoica, I. (2002), "A scalable and robust solution for bandwidth allocation," in *Quality of Service, 2002. Tenth IEEE International Workshop on*, pp. 148–157, IEEE.

Mackay, W. E. (1991), "Triggers and barriers to customizing software," in *Proc. CHI*.

Maier, G., Feldmann, A., Paxson, V., and Allman, M. (2009), "On Dominant Characteristics of Residential Broadband Internet Traffic," in *IMC*.

Margolin, N. B. and Levine, B. N. (2008), "Financial Cryptography and Data Security; Chapter "Quantifying Resistance to the Sybil Attack"," Springer-Verlag.

McQuillan, J. M. and Walden, D. C. (1977), "The ARPA network design decisions," *Computer Networks (1976)*, 1, 243–289.

Nagaraja, S., Mittal, P., Hong, C.-Y., Caesar, M., and Borisov, N. (2010), "BotGrep: finding P2P bots with structured graph analysis," in *Proceedings of the 19th USENIX conference on Security*.

Networking, C. G. V. and Index., C. (2013), "Forecast and Methodology, 2012-2017." `http://www.cisco.com`.

Nordstrom, E., Shue, D., Gopalan, P., Kiefer, R., Arye, M., Ko, S., Rexford, J., , and Freedman, M. J. (2012), "Serval: An End-Host Stack for Service-Centric Networking," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, NSDI'12.

Nygren, E., Sitaraman, R. K., and Sun, J. (2010), "The Akamai Network: A Platform for High-performance Internet Applications," *SIGOPS Oper. Syst. Rev.*, 44.

OCN (2012), "Announcing the Netflix Open Connect Network," `http://blog.netflix.com/2012/06/announcing-netflix-open-connect-network.html`.

Otto, J. S., Sánchez, M. A., Rula, J. P., and Bustamante, F. E. (2012), "Content Delivery and the Natural Evolution of DNS - Remote DNS Trends, Performance Issues and Alternative Solutions," in *imc*.

Parno, B., Wendlandt, D., Shi, E., Perrig, A., Maggs, B. M., and Hu, Y.-C. (2007a), "Portcullis: Protecting connection setup from denial-of-capability attacks," in *Proc. SIGCOMM*.

Parno, B., Wendlandt, D., Shi, E., Perrig, A., Maggs, B., and Hu, Y.-C. (2007b), "Portcullis: protecting connection setup from denial-of-capability attacks," *ACM SIGCOMM Computer Communication Review*, 37, 289–300.

Paxson, V. (1999), "Bro: A System for Detecting Network Intruders in Real-Time," *Com. Networks*.

Perino, D. and Varvello, M. (2011), "A reality check for content centric networking," in *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, ICN '11, pp. 44–49.

Peterson, R. S. and Sirer, E. G. (2009), "Antfarm: efficient content distribution with managed swarms," in *Proc. NSDI*.

Piatek, M., Isdal, T., Anderson, T., and Krishnamurthy, A. (2007), "Do Incentives Build Robustness in BitTorrent?" in *Proc. NSDI*.

Poese, I., Frank, B., Ager, B., Smaragdakis, G., and Feldmann, A. (2010), "Improving Content Delivery using Provider-Aided Distance Information," in *IMC*.

Punkbuster (2000), "Major Features in PunkBuster," `http://www.evenbalance.com/index.php?page=info.php`.

Qureshi, A., Weber, R., Balakrishnan, H., Guttag, J., and Maggs, B. (2009), "Cutting the Electric Bill for Internet-scale Systems," in *SIGCOMM*.

Rayburn, D. (2011), "Telcos and Carriers Forming New Federated CDN Group Called OCX (Operator Carrier Exchange)." *StreamingMediaBlog. com*.

Ristic, I. (2010), *ModSecurity Handbook*, Feisty Duck.

Rosenberg, J., Mahy, R., Matthews, P., and Wing, D. (2008), "Session Traversal Utilities for NAT (STUN)," RFC 5389 (Proposed Standard).

SAIL (2010), "Scalable and Adaptive Internet Solutions (SAIL)," `http://www.sail-project.eu/`.

Saltzer, J. H., Reed, D. P., and Clark, D. D. (1984), "End-to-end arguments in system design," *ACM Trans. Comput. Syst.*, 2, 277–288.

Saroiu, S., Gummadi, K., and Gribble, S. (2002), "A Measurement Study of Peer-to-Peer File Sharing Systems," in *Proc. MMCN*.

Seuken, S. and Parkes, D. C. (2011), "On the Sybil-Proofness of Accounting Mechanisms," in *Proc. NetEcon*.

Sirivianos, M., Park, J. H., Yang, X., and Jarecki, S. (2007a), "Dandelion: Cooperative Content Distribution with Robust Incentives," in *Proc. USENIX ATC*.

Sirivianos, M., Park, J. H., Chen, R., and Yang, X. (2007b), "Free-riding in BitTorrent Networks with the Large View Exploit," Tech. Rep. CECS-07-01, University of California, Irvine.

Sitaraman, R. K., Kasbekar, M., Lichtenstein, W., and Jain, M. (2014), "Overlay Networks: An Akamai Perspective," .

Slashdot (2011), "Slashdot: Major Outage At the Amazon Web Services," `http://slashdot.org/story/11/04/21/1515238/major-outage-at-the-amazon-web-services`.

Spring, N., Mahajan, R., Wetherall, D., and Anderson, T. (2004), "Measuring ISP topologies with rocketfuel," *IEEE/ACM Trans. Netw.*, 12.

Tariq, M., Zeitoun, A., Valancius, V., Feamster, N., and Ammar, M. (2009), "Answering What-if Deployment and Configuration Questions with Wise," in *SIGCOMM*.

TRA (2013), "Your Gadgets Are Slowly Breaking the Internet," `http://www.technologyreview.com/news/509721/your-gadgets-are-slowly-breaking-the-internet/`.

Triukose, S., Al-Qudah, Z., and Rabinovich, M. (2009), "Content delivery networks: protection or threat?" in *Computer Security–ESORICS 2009*, pp. 371–389, Springer.

Vu, L., Gupta, I., Nahrstedt, K., and Liang, J. (2010), "Understanding overlay characteristics of a large-scale peer-to-peer IPTV system," *ACM Trans. Multim. Comp. Comm. Appl.*, 6, 31:1–31:24.

WAFAccuracy (2011), "Improvements to Akamai Kona Site Defender Shown to Yield One of Industry's Most Accurate WAFs," `http://www.akamai.com/html/about/press/releases/2014/press_020414.html`.

Walfish, M., Vutukuru, M., Balakrishnan, H., Karger, D., and Shenker, S. (2006), "DDoS defense by offense," in *ACM SIGCOMM Computer Communication Review*, vol. 36, pp. 303–314, ACM.

Wolman, A., Voelker, G. M., Sharma, N., Cardwell, N., Karlin, A., and Levy, H. M. (1999), "On the Scale and Performance of Cooperative Web Proxy Caching," in *ACM Symposium on Operating Systems Principles*, pp. 16–31, ACM New York.

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

Yaar, A., Perrig, A., and Song, D. (2004), "An endhost capability mechanism to mitigate DDoS flooding attacks," in *Proceedings of the IEEE Symposium on Security and Privacy*.

Yang, X. and de Veciana, G. (2004), "Service Capacity of Peer to Peer Networks," in *INFOCOM*.

Yang, X., Wetherall, D., and Anderson, T. (2005), "A DoS-limiting network architecture," *ACM SIGCOMM Computer Communication Review*, 35, 241–252.

Yin, H., Liu, X., Zhan, T., Sekar, V., Qiu, F., Lin, C., Zhang, H., and Li, B. (2009), "Design and deployment of a hybrid CDN-P2P system for live video streaming: experiences with LiveSky," in *Proceedings of the 17th ACM international conference on Multimedia*, pp. 25–34, ACM.

Yu, H., Kaminsky, M., Gibbons, P. B., and Flaxman, A. (2006), "SybilGuard: defending against Sybil attacks via social networks," in *Proc. SIGCOMM '06*.

Yu, H., Gibbons, P. B., Kaminsky, M., and Xiao, F. (2008), "SybilLimit: A Near-Optimal Social Network Defense against Sybil Attacks," in *Proc. IEEE S&P*.

Yu, H., Shi, C., Kaminsky, M., Gibbons, P. B., and Xiao, F. (2009), "DSybil: Optimal Sybil-Resistance for Recommendation Systems," in *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, pp. 283–298, Washington, DC, USA, IEEE Computer Society.

Zhao, M., Aditya, P., Chen, A., Lin, Y., Haeberlen, A., Druschel, P., Maggs, B., Wishon, B., and Ponec, M. (2013), "Peer-assisted content distribution in Akamai netsession," in *Proceedings of the 2013 conference on Internet measurement conference*, pp. 31–42, ACM.

Biography

Yin Lin was born in Fuzhou, China on September 24th, 1986. He defended his PhD thesis at Duke University in June 2014. He received his master degree on Computer Science from the same school in 2012 and his B.S. degree in Software Engineering from Shanghai Jiao Tong University, China in 2009. He was rewarded James B. Duke fellowship during years of his graduate study. He is expected to join VMware at Palo Alto after he graduates.