ON P2P NETWORKS AND P2P-BASED CONTENT DISCOVERY ON THE INTERNET

by

GHULAM MEMON

A DISSERTATION

Presented to the Department of Computer and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

December 2013

DISSERTATION APPROVAL PAGE

Student: Ghulam Memon

Title: On P2P Networks and P2P-Based Content Discovery on the Internet

This dissertation has been accepted and approved in partial fulfillment of the requirements for the Doctor of Philosophy degree in the Department of Computer and Information Science by:

Prof. Jun Li	Chair
Prof. Andrzej Proskurowski	Core Member
Prof. Reza Rejaie	Core Member
Prof. Michal Young	Core Member
Prof. Kevin Butler	Core Member
Prof. Yuan Xu	Institutional Representative

and

Kimberly Andrews Espy	Vice President for Research & Innovation/ Dean of the Graduate School

Original approval signatures are on file with the University of Oregon Graduate School.

Degree awarded December 2013

DISSERTATION ABSTRACT

Ghulam Memon

Doctor of Philosophy

Department of Computer and Information Science

December 2013

Title: On P2P Networks and P2P-Based Content Discovery on the Internet

The Internet has evolved into a medium centered around content: people watch videos on YouTube, share their pictures via Flickr, and use Facebook to keep in touch with their friends. Yet, the only globally deployed service to discover content - i.e., Domain Name System (DNS) - does not discover content at all; it merely translates domain names into locations. The lack of persistent naming, in particular, makes content discovery, instead of domain discovery, challenging. Content Distribution Networks (CDNs), which augment DNSs with location-awareness, also suffer from the same problem of lack of persistent content names. Recently, several infrastructure-level solutions to this problem have emerged, but their fundamental limitation is that they fail to preserve the autonomy of network participants. Specifically, the storage requirements for resolution within each participant may not be proportional to their capacity. Furthermore, these solutions cannot be incrementally deployed. To the best of our knowledge, content discovery services based on peer-to-peer (P2P) networks are the only ones that support persistent content names. These services also come with the built-in advantage of scalability and deployability. However, P2P networks have been deployed in the real-world only recently, and their real-world characteristics are not well-understood. It is important to understand these real-

world characteristics in order to improve the performance and propose new designs by identifying the weaknesses of existing designs. In this dissertation, we first propose a novel, lightweight technique for capturing P2P traffic. Using our captured data, we characterize several aspects of P2P networks and draw conclusions about their weaknesses. Next, we create a botnet to demonstrate the lethality of the weaknesses of P2P networks. Finally, we address the weaknesses of P2P systems to design a P2P-based content discovery service, which resolves the drawbacks of existing content discovery systems and can operate at Internet-scale.

This dissertation includes both previously published/unpublished and co-authored material.

CURRICULUM VITAE

NAME OF AUTHOR: Ghulam Memon

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon, Eugene, OR

Karachi Institute of Information Technology, Karachi, Pakistan, affiliated with the University of Huddersfield, UK

DEGREES AWARDED:

Doctor of Philosophy in Computer and Information Science, 2013, University of Oregon

Bachelor of Science in Software Development, 2002, Karachi Institute of Information Technology, affiliated with the University of Huddersfield

AREAS OF SPECIAL INTEREST:

Computer Networks, Peer to Peer Networks, Content Networks, Security, Distributed Systems

PROFESSIONAL EXPERIENCE:

Graduate Research Fellow, University of Oregon, September 2006 - December 2013

Summer Intern, Bell Labs, Holmdel, NJ, June 2012 - September 2012

Summer Intern, Bell Labs, Holmdel, NJ, June 2010 - September 2010

Summer Intern, Thomson Research Lab, Princeton, NJ, June 2007 - September 2007

Research Programmer, San Diego Supercomputer Center, San Diego, CA, June 2004 - August 2006

Teleworker Software Engineer, Synentia, Dubai, UAE, September 2003 - December 2003

Software Engineer, Synentia, Karachi, Pakistan, November 2002 - August 2003

GRANTS, AWARDS AND HONORS:

Clarence & Lucille Dunbar Scholarship, 2010

Erwin & Gertrude Juilfs Scholarship, 2009

Symposium on Networked Systems Design & Implementation Travel Grant, April 2009

Gold Medalist, Karachi Institute of Information Technology, 2002

PUBLICATIONS:

G. Memon, J. Li and M. Varvello, "Compass: A Content Discovery Service on the Internet," University of Oregon, Tech. Rep. CIS-TR-2013-14, 2013.

G. Memon, J. Li and R. Rejaie, "Tsunami: A Parasitic, Indestructible Botnet on Kad," *Peer-to-Peer Networking and Applications*, DOI. 10.1007/s12083-013-0202-x, April 2013.

G. Memon, R. Rejaie, Y. Guo and D. Stutzbach, "Montra: A Large-Scale DHT Traffic Monitor," *Elsevier Computer Networks, Special Issue on Measurement-based Optimization of P2P Networking and Applications*, vol. 56, no. 3, February 2012, pp. 1080–1091.

G. Memon and J. Li, "Survey of Future Internet Architectures," University of Oregon, Tech. Rep. CIS-TR-2011-03, 2011.

G. Memon, R. Rejaie, Y. Guo and D. Stutzbach, "Large-Scale Monitoring of DHT Traffic," in *Proceedings of International Workshop on Peer-to-Peer Systems (IPTPS)*, Boston, MA, April 2009.

C. Youn, C. Baru, K. Bhatia, S. Chandra, K. Lin, A. Memon, G. Memon and D. Seber, "GEONGrid Portal: Design and Implementations," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 12, May 2007, pp. 1597–1607.

E. Jaeger-Frank, A. Memon, I. Altintas, G. Memon, D. Seber, B. Ludaescher and C. Baru, "Integrating Seismic Events Focal Mechanisms With Image Services In Kepler," in *Proceedings of 25th Annual ESRI International User Conference*, San Diego, CA, July 2005.

G. Memon, A. Shaikh, K. Lin, I. Zaslavsky and C. Baru, "Generating Composite Thematic Maps From Semantically Different Shapefiles," in *Proceedings of 25th Annual ESRI International User Conference*, San Diego, CA, July 2005.

Q. Memon, Z. Shaikh and G. Memon, "Private Textual Network Using GSM Architecture," in *Proceedings of the IEEE International Multi Topic Conference*, Karachi, Pakistan, December 2002, pp. 546–550.

ACKNOWLEDGEMENTS

For the last 7 years this dissertation has been the entire focus of my life, much like a child, which has slowly grown up. As the saying goes, it takes a village to raise a child. So, it took a village to complete this dissertation as well. Unfortunately, I can only list a few people here, but anyone who ever taught me anything, including family, friends, teachers, advisors, co-workers, employers, played a role in the completion of this dissertation, and I am thankful to all.

I am grateful to my thesis committee for helping me in improving the structure of the dissertation, for their insightful comments on the technical and editorial aspects of the dissertation, and for always providing a listening ear whenever I needed. In particular, I am thankful to my advisor, Prof. Jun Li for teaching me that the best research can only be done when it is chosen because of researcher's interest, and not because of the current trend. Furthermore, I am thankful to him for allowing me to choose the work that I like, and then teaching me how to convert an interest into a well-defined problem. He further taught me how to motivate the problem, and how to approach different solutions.

During the early years of my Ph.D., I worked under the guidance of Prof. Reza Rejaie. I am thankful to Prof. Reza Rejaie for introducing me to research, for teaching me how to conduct research rigorously, for helping me in finding needles in large amounts of data, and for always pushing the envelope further by asking tough questions. Prof. Rejaie also taught me that a balance must be maintained between ambition and reality. I will always remain grateful for Prof. Rejaie's sage advice.

I am thankful to current and former members of NETSEC and MIRAGE research labs for being the sounding board for my research ideas. In particular, I am thankful

to Dr. Daniel Stutzbach, Dr. Amir Rasti, Dr. Nazanin Maghraei, Masoud Valafar, Mojtaba Torkjazi, Dr. Toby Ehrenkranz, Dr. Shad Stafford, Jason Gustafson, Paul Elliot, Mingwei Zhang, and Josh Stein for their insightful comments on my research and for reading and commenting on my papers.

I am grateful to the wonderful staff of the Computer and Information Science department. In particular, words cannot describe the help and support that Star Holmberg has provided during my Ph.D. She went above and beyond to help out whenever I discussed any issue with her, and always provided a calm and comforting listening ear. I will always remain grateful to her for her support.

Finally, I am thankful to my family, because without their support I would have never been able to complete my dissertation. In particular I am thankful to my parents for their financial and emotional support throughout my educational years. I am also thankful to them for instilling the love of knowledge in me, and for inspiring me to reach the highest possible level of education. I am also thankful to Mrs. Margaret Smeekens Vanhorn for always standing by my side through thick and thin. When my papers got accepted, or I accomplished something, she has been my biggest fan. On the other hand, whenever my papers got rejected, or when things became tough, she has provided unconditional support. Above all, win or lose, Margaret's never-ending supply of cookies has always been there to either cheer me up or to add to the celebration.

My work on this dissertation was supported in part by the National Science Foundation (NSF) under Grant No. Nets-NBD-0627202 and CNS-0644434. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

To my parents,

Akbar and Zarina Memon

TABLE OF CONTENTS

Chapter Page

Chapter Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

LIST OF TABLES

Table Page

CHAPTER I

INTRODUCTION

Peer to Peer (P2P) networks represent the most popular form of distributed resource sharing. The key concept behind P2P networks is that several users (peers), distributed globally, share various *resources* amongst themselves without the presence of a central management authority. Examples of the resources are files, CPU cycles and network bandwidth. The most common use of P2P networks is to share files.

P2P systems[1] represent a significant paradigm shift in the way distributed systems are designed. The most interesting aspect of P2P systems is their built-in scalability - i.e., each peer contributes its own set of resources. This property is in sharp contrast with traditional distributed systems, which rely on designated nodes for resources. Furthermore, the P2P approach relieves the nodes in the distributed system from relying on a central point of failure. Since the P2P approach of designing distributed systems is relatively new compared to the centralized approach, our knowledge and understanding is still growing. Thus, it is important to study the deployed instances of P2P systems, and then improve the design of P2P systems based on our accumulated knowledge.

Today's deployed P2P systems focus on content discovery, and for good reason. Content discovery is an important problem, because today's Internet is primarily used for consuming content. However, to the best of our knowledge, P2P networks have not been used for an Internet-scale content discovery service. Specifically, P2P networks serve tens of millions of files [1] only, whereas the Internet consists of at least 1 trillion web pages [2]. In this dissertation, we demonstrate that current P2P networks exhibit

[1]In this dissertation, we use the term P2P networks and P2P systems interchangeably.

fundamental weaknesses, and if those weaknesses are addressed, then a P2P network can be an ideal candidate for designing an Internet scale content discovery service.

1.1. Thesis Statement

Peer-to-peer systems suffer from fundamental weaknesses, which if addressed, can make P2P systems useful in the design of an Internet scale content discovery service.

1.2. Understanding Peer-to-Peer Networks

P2P networks are broadly divided into unstructured and structured networks. Unstructured networks were the first generation of P2P networks, which naively relied on flooding for communication. However, the research community found flooding to be an unsuitable means of communication, because the discovery of a resource is not always guaranteed. In order for flooding to be sustainable, unstructured P2P networks impose a hop count limit beyond which flooding will not continue. If the desired resource exists beyond the hop limit of flooding, then it will never be discovered. Unstructured networks are fairly well-studied by the research community [3–6]. Structured P2P networks constitute the second iteration of P2P networks, in which the peers adhere to a geometrical structure (e.g., circle [7] or tree [8]), and the resource discovery is always guaranteed. The structure that these networks create is completely separate from the underlying network, and is created in the form of an overlay. All structured networks guarantee that any query for a resource will be resolved in $O(log_2 n)$ hops, where n is the total number of nodes in the network. In this dissertation we focus on structured P2P networks, and in the remainder of the dissertation we use P2P networks and structured P2P interchangeably.

The deployment of structured P2P networks is relatively new, and needs careful monitoring to understand their properties. Better understanding of existing systems not only allows us to improve their performance, but also sheds light on the design of future systems. While extensive simulation experiments, analysis and small scale deployments [9, 10] have been conducted in the past, few studies have been done on existing systems with large numbers of peers. As a result, the real-world properties of large-scale structured P2P networks are not well understood.

Accurate measurement of P2P networks is a challenging task. The sheer size of the problem is daunting: an appropriate portion of peers need to be monitored in order to collect sufficient data to draw any meaningful results. An instrumented peer can be deployed to collect the measurement data. However, deploying a large number of monitoring peers requires a significant amount of computing resources and network bandwidth, because monitoring peers need to participate in the network in order to monitor it. The addition of a large number of monitoring peers may drastically alter the P2P network, potentially making the measurement results biased or even invalid [11].

We present a P2P network-monitoring technique, called Montra, that can monitor thousands of peers using modest resources, without significantly disturbing the P2P network's normal operation. We implement Montra in the form of a highly parallel, scalable Python-based client. We rigorously evaluate our implementation of Montra using an actual deployment over real-world P2P networks. We found that our implementation of Montra can accurately monitor around 32,000 peers using a moderately configured PC (an Intel Core 2 Duo with 1GB RAM) with 90% accuracy. While the program runs on a single machine, it spawns multiple threads to use multiple cores available on the machine. In addition, we conduct a detailed

characterization study of collected data, in order to understand the weaknesses that hinder the use of P2P networks for an Internet-scale content discovery service.

1.3. Content Discovery

Although originally designed for connecting machines, the Internet has evolved into a medium devoted to the production and consumption of content: users search for information over Google, watch videos on YouTube, and share their pictures via Flickr. Before proceeding further, it is important to define that for this dissertation what we mean by content. For our purposes, content is any collection of bytes that has a well-structured name. For example, a web page, a web-based video or image are all examples of content because all these objects have well-structured names in the form of URLs. *Content delivery* is a mechanism used to make content available to Internet users. The original Internet infrastructure possesses no native support for content delivery; this fundamental limitation has motivated several incremental as well as clean-slate solutions.

Content delivery consists of *content dissemination* and *content discovery*. Content dissemination is a mechanism that strategically decides where in the network to store one or more copies of some content. Content discovery refers to a mechanism that maps content, identified by a name or unique identifier, to its "best" location or IP address.

While there has been significant work in the area of content dissemination [12–15], we believe that the problem of content discovery is still far from being completely solved. Domain Name System (DNS) [16] is the only globally deployed content discovery system. But, DNS was not designed to discover content; it was designed to translate hostnames into IP addresses. Because of this fundamental design choice,

DNS suffers from limitations as a content discovery service. This dissertation proposes a new design for an Internet-scale content discovery service, based on P2P networks.

1.3.1. Requirements

We believe that the key requirement for a content discovery system is the ability to support persistent content names. Persistent content names are only bound to the content, and are independent of location, content ownership or domain names. As a result, even if content moves from one owner to another (e.g., different organizations) or is replicated to a different location, the name does not change. Thus, persistent names naturally support content mobility and content replication. Persistence in content names allows a content discovery service to treat content, and not domains, as first-class citizens. Furthermore, a content discovery system must be secure so that its infrastructure cannot be attacked and compromised. A content discovery system must also respect the autonomy of participating entities. In addition, a content discovery system must be flexible enough to integrate existing location-aware content dissemination systems. Finally, the content discovery system must be deployable over today's Internet.

1.3.2. A New Content Discovery Service

As a part of this work, we design a new content discovery system, called Compass, which fulfills all the requirements mentioned in Section 1.3.1.. The Compass content discovery system consists of content discovery agents (CDAs), which resolve content identifiers into their locations (i.e., IP addresses). Compass provides the following advantages, which are missing in other content discovery systems:

– Persistent Names: Lack of persistent names is the key drawback in some of the existing content discovery systems. Compass, however, supports arbitrary names that are not bound to any domain or provider.

– Secure: In order to prevent untrusted CDAs from being part of the system, Compass does not allow arbitrary CDAs to join. A newly joining CDA can only be part of Compass if an existing CDA authorizes the participation. Even if an existing CDA admits a malicious CDA, the newly joining malicious CDA can still be held accountable if it causes any damage to the Compass system.

– Scalable: Compass is scalable because it respects the autonomy of its CDAs, and does not dictate which resolution information they must store.

– Deployable: In terms of deployment, Compass CDAs are similar to today's DNS resolvers, and can be incrementally deployed over today's Internet.

– Location Awareness: Compass is capable of incorporating existing location awareness mechanisms.

1.4. Contributions

In this dissertation, we make the following specific contributions:

– Design of a new technique to monitor structured P2P networks, and characterization of captured traffic.

– Design of a new botnet to demonstrate the weaknesses of P2P systems.

– Requirements for the design of a new content discovery system

– Design and evaluation of a new content discovery system, which addresses the problems of existing content discovery systems.

1.5. Dissertation Organization

The remainder of this dissertation is organized as follows. In Chapter II, we provide detailed background on the operation of structured P2P networks. In Chapter III, we present a new technique for monitoring structured P2P networks, in order to understand their weaknesses. In Chapter IV, we present a new botnet design, in order to demonstrate the weaknesses of P2P systems. In Chapter V, we discuss our philosophy behind Compass design, its low-level design and security. In Chapter VI, we present implementation details of Compass, its evaluation and path to its deployment. We conclude the dissertation in Chapter VII.

1.6. Co-Authorship Acknowledgments

Portions of the text in this dissertation are based on published and unpublished co-authored material. Chapter III includes text from [17, 18]. Chapter IV includes text from [19]. Chapter V, and Chapter VI includes text from [20, 21].

CHAPTER II

PEER-TO-PEER NETWORKS

In this chapter we provide an overview of peer-to-peer (P2P) networks, specifically, structured P2P networks.

2.1. Overview

P2P networks are divided into 2 major types: *structured* and *unstructured*. In structured networks, a distributed algorithm ensures that all the peers adhere to a particular structure (e.g., circle [7] or tree [8]). The presence of a structure ensures that 2 peers can contact each other and share resources by simply following the path dictated by the structure. As a result, if a resource exists in the network then it will be found. The most popular form of structured networks is Distributed Hash Tables (DHTs) [22–26]. In an unstructured network, on the other hand, no such structure exists. Peers join the network by simply contacting random peers. Peers search for resources by flooding the network. In order for flooding to be sustainable, unstructured networks impose a hop count limit, beyond flooding does not proceed. As a result, even if a given resource exists in the network, there is no guarantee that it will be found because it may reside beyond the scope of flooding. The most popular example of unstructured networks is Gnutella [27].

In addition to lack of guarantee that a resource will be found, flooding in unstructured networks also have performance implications [28]. While both structured and unstructured P2P networks are efficient in finding popular content objects. However, structured networks are twice as effective in finding unpopular objects, when compared against unstructured networks. This effect is primarily

because in structured networks, each peer knows what is the next appropriate hop to which each packet should be forwarded. In unstructured networks, however, peers blindly flood the network. Since queries for a particular object stop after going through certain number of hops, if an object is unpopular, it is not necessarily reached. Furthermore, even for popular objects, the query resolution time in unstructured networks is almost twice as high as structured networks. This effect too is because of flooding, as peers do not know where to send the packet, and blindly flood the network. Finally, in structured networks, the query load that a peer observes depends upon its position in the content id space. On the other hand, in unstructured networks, the query load depends upon a peer's degree - i.e., the larger the degree the more traffic a peer will see because of flooding. Given the ill-effects of flooding, structured networks were created precisely to avoid flooding when issuing queries. Since structured networks are more technologically-advanced, and also unstructured networks have been studied fairly extensively [3–6], in this dissertation, we focus on structured networks - specifically DHTs.

2.2. Distributed Hash Tables

In this section we first give a brief background on DHTs, and then discuss the operational details of the most-widely adopted DHT, Kademlia, and its real-world implementation, Kad.

Structured Routing approaches came to light with the advent of Distributed Hash Tables (DHTs). A DHT, as the name indicates, stores $< key, value >$ pairs in a distributed fashion. In other words, a DHT is a single hash table, and each node in the DHT stores a portion of the hash table. In order to assign *ownership* of keys to a particular node, each node is assigned a unique ID that belongs to the same n

bit (commonly used values of n are 128 and 160) ID space as keys. The owner node for a key y is determined by finding a node x that is closest to y in the ID space. DHTs use different notions of closeness, such as numeric difference [7], longest prefix matching [29, 30] and XOR distance [8]. In a real-world deployment, DHTs cannot function by storing a given key on a single node, because that node might depart permanently and the value of the key will be lost. In order to avoid this scenario, DHTs store each key-value pair on m (common values of m are 10 and 20) closest possible nodes to the key, instead of just one node.

The usage of a structure implies adhering to a specific geometry for peer connections. DHTs use such geometries as Ring [7], Hypercube [31] and Tree [29], [30]. DHTs use the geometry to dictate a unique path from one peer to another. It is because of this uniqueness that DHTs can always find a value for a given key, as opposed to an unstructured approach. All the DHTs introduced so far guarantee that every key lookup will be resolved in $O(log_2 N)$ hops or less, where N is the total number of nodes in the network.

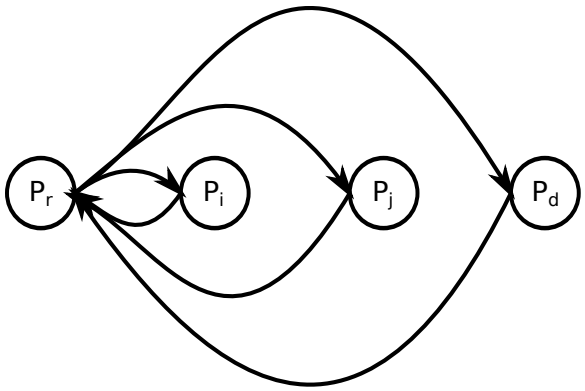


FIGURE 2.1. Iterative Routing Process in DHTs.

DHTs use *iterative routing* to find a peer whose ID is closest to a key K. The iterative routing process of DHTs is demonstrated in Figure 2.1.. It shows three different types of peers:

1. Peer P_r, which starts the iterative routing process.

2. Peer P_d, which is the closest peer to K.

3. Peers P_i and P_j, which are intermediate peers that route requests from peer P_r towards peer P_d.

Peer P_r initiates the routing process by consulting its own routing table and finds a peer P_i that is closest to the key K. Peer P_r sends a request for K to the peer P_i, as shown in Figure 2.1.. Peer P_i consults its own routing table, finds the closest possible peers to K, and informs the peer P_r about the newly discovered peers that are closer to K. This process continues until the peer P_r finds the peer P_d, which is the closest possible peer to K.

The innovation in the field of DHTs occurred in two separate rounds. In the first round, four independent works of DHTs were introduced almost simultaneously: Chord [7], CAN [31], Pastry [29] and Tapestry [30]. All the four DHTs provide the same guarantee that a query can be resolved in $O(log_2 N)$ or lesser hops. However, these DHTs differ from each other in their geometry, and closeness metrics: Chord structures its peers into a ring and uses numeric difference as closeness metric, CAN uses a hypercube structure and cartesian distance as closeness metric, Pastry and Tapestry uses a tree structure and longest prefix match as closeness metric. A later study by Gummadi et al. [32] compared all different geometries and found that the ring structure has advantage over other geometries in terms of resilience.

The second round of DHT innovation advanced the first round by addressing requirements of a real-world deployment. Specifically, these DHTs incorporate replication of keys over multiple nodes, and fault tolerance by contacting multiple nodes at each hop, instead of just one. The DHTs introduced in the second round

are OpenDHT [33], Accordion [34] and Kademlia [8]. From all the DHTs introduced so far, Kademlia is the only that has been used in real-world applications. In the remainder of this chapter we focus on explaining the details of the Kademlia DHT, because in later chapters we use various deployments of Kademlia to further our understanding of P2P systems.

2.2.1. Kademlia

Kademlia [26] is the most widely-adopted DHT, because it improves on some of the weaknesses of earlier DHTs [22–25], and in the process makes DHTs a more practical and usable approach.

Kademlia belongs to the class of prefix matching DHTs, just like Pastry [23] and Tapestry [24]. In general, prefix matching naturally maps to a tree structure (e.g., IP addresses). Because of this inherent mapping, Kademlia's routing tables collectively form a binary tree which is similar to Pastry and Tapestry. But unlike Pastry and Tapestry, Kademlia uses one uniform routing algorithm and distance metric from start to end of the search operation, hence adding simplicity to the design. Kademlia uses XOR metric to calculate distance between 2 IDs in ID space, which is simply a form of longest prefix matching.

The practicality of Kademlia comes from its ample use of redundancy. The basic idea is that instead of making one and only one peer responsible for a given key, Kademlia allows a given key-value pair to be stored at multiple nodes. This pro-active approach allows Kademlia to handle churn effectively.

Kademlia comes with built-in fault tolerance, which is achieved by using *k-buckets*, where k is the number of entries in the bucket. Each k-bucket corresponds to a single row in the routing table of other DHTs. Unlike earlier prefix matching

DHTs, instead of maintaining 1 contact for every prefix in the routing table, Kademlia maintains k contacts. Increasing k increases redundancy in Kademlia and hence improves lookup performance at the expense of more bandwidth usage.

Kademlia uses parallel lookups to perform routing, i.e., instead of picking 1 next hop contact at each hop, Kademlia picks α contacts. The provision of parallel lookups prevents request originators from a timeout if the next hop peer is not available. The value of α varies for different implementations of Kademlia. Choosing a high value for α results in larger bandwidth consumption and lower latency. On the other hand, a smaller value of α denotes a relatively higher latency threshold. In case of Kad (described next), the value of α is 3.

2.2.1.1. Kad

Kad is the real-world implementation of Kademlia, which is employed by eMule file-sharing software [35]. Kad uses 128 bit ID space for peers and keys. Since Kad is used for file-sharing, it uses two different types of keys: file-ID and keyword-ID. File-ID is the cryptographic hash of the entire file that is being shared. The value for file-ID is the pair (IP address, TCP port number) of the peer that is sharing the file. Each file is also associated with several keywords. Keyword-ID is the cryptographic hash of a given keyword, and its value is file metadata (e.g., file name, file type, file size). In the remainder of this section, we refer to keywords and files as content, and file-ID and keyword-ID as content-ID.

Kad uses different kinds of messages to perform its regular operation. Kad allows a publisher to *publish* content and a consumer to *search* specific content. A *publish* operation includes a *lookup* phase and a *storage* phase. A *search* operation includes a *lookup* phase and a *retrieval* phase.

Lookup Phase: The lookup phase is common between publish and search operations. This phase is aimed at finding 10 closest possible alive nodes to the target content-ID. This phase uses REQUEST messages, which carry the target content-ID and requested number of contacts x. Whenever a peer receives a REQUEST message, it checks its own routing table to find x closest contacts to the target content-ID. The peer then embeds these contacts in a RESPONSE message and sends the message to the peer that sent the REQUEST message.

Once 10 alive nodes are found, Kad clients send either SEARCH or PUBLISH message for keyword or file, to these nodes. The four different types of requests are described below:

- PUBLISH_FILE (PuF) for publishing a file, where the *key* field of the message is the hash of the file, and the *value* field of the message is the publisher's IP address and TCP port number.

- PUBLISH_KEYWORD (PuK) for publishing a keyword, where the *key* field of the message is the keyword's hash, and the *value* field is the meta-data of those files that include the keyword in their names. The meta-data for each file includes their name, hash, size and type.

- SEARCH_FILE (SeF) for searching a file, where the *key* field of the message is the hash of the file. A response to this message carries the IP address and TCP port number of the file's publisher.

- SEARCH_KEYWORD (SeK) for searching the files associated with a keyword, where the *key* field of the message is the hash of the keyword. The response to this message carries the meta-data of relevant files, including their hash. For

each desired file, a consumer then can use the retrieved file hash to issue an SeF request to get the file.

The reason that a Kad client stores content at multiple nodes by sending multiple PUBLISH messages is to ensure availability by replication of content. Also, it sends multiple SEARCH messages to retrieve maximum possible results.

In addition to search and publish operations, a Kad client must also maintain its routing table. A client's routing table is maintained by adding newly discovered alive peers and discarding dead peers. In order to check if a newly discovered peer is alive or an existing peer is dead, a Kad client uses heartbeat messages, called HELLO messages. The client sends a HELLO REQ message to check if a peer is alive. If it receives a HELLO RES message then it knows that the peer is alive. Otherwise the peer is considered dead. Both HELLO REQ and HELLO RES messages carry the node ID of the peer that sends the message.

2.2.1.2. Azureus

Azureus is a popular BitTorrent client. Azureus was the first client to support trackerless swarms by using the Kademlia DHT. In this section we describe how Azureus supports its trackerless operation by using the Kademliad DHT.

Azureus uses a DHT to ensure that Azureus clients can find swarm peers in the absence of a tracker[1]. Since DHTs use key value pairs, the Azureus DHT generates the key by hashing the *info torrent hash*, found in the .torrent files. The info torrent hash is used by a given peer when communicating with the tracker and other peers. The DHT uses a hash of this info hash in order to preserve anonymity of peers downloading

[1]BitTorrent uses a centralized node, called tracker, which coordinates the discovery of file blocks among peers

the file. Peer IDs are assigned based on a peer's external (non-NAT[2]) IP address and port combination. The ip port string is constructed as "$< ip >:< port >$". The peer's ID is the SHA1 [37] hash of ip:port string. A peer finds is external address by contacting different peers. Each peer reports its perceived IP address to every peer contacted. The target peers compare the reported IP address and the transport-level IP address. If the addresses do not match then the receiving peer reports the correct IP address to the packet sender. From that point the sender uses that IP address as its perceived IP address.

Since a peer's ID relies on its IP and port combination, it must always use same combination during one session. As a result, peers send repeated keep-alive messages to keep the outgoing port on the NAT box open.

The Azureus DHT supports 2 operations:

- Put: The peer puts the information about its IP address and TCP port in the DHT. This information will be used by other peers for contacting this peer and for downloading files.

- Get: A peer uses the Get request to retrieve TCP information for the peers of a given swarm. The returned information contains the IP address and TCP port of swarm members.

The Put and Get operations are equivalent to publish and search operations in Kad. Just like Kad, put and get operations are preceded by a lookup operation. The goal of the lookup operation is to find n closest peers to a given ID. Azureus set n=20, ensuring greater redundancy, whereas Kad sets n=10.

[2]NAT stands for Network Address Translation [36]

Unlike kad, put and get operations are not user-driven. The operations are automatically repeated every 30 seconds. The operations are always back-to-back i.e. each peer first performs a put and then a get.

The lookup operation, which precedes the put and get operations, starts with a *FindValue* request. A FindValue request inquires a peer whether it possesses a given value. If the peer does not possess the value, it responds with *FindNode* response that contains other nodes to be inquired (i.e., next hops). If during the lookup operation, a peer receives FindNode response, it repeats the process and sends FindValue request to the peers contained in FindNode response.

Both put and get operations end when either desired number of contacts have been reached or timeout has occurred. Put operation is more exhaustive and finishes only when values are published at 20 contacts. On the other hand, the get operation ends when 30 peers in a swarm are found.

2.3. Recap

In this chapter we provided an overview of P2P networks, with emphasis on DHTs.

CHAPTER III

MONTRA: LARGE-SCALE DHT TRAFFIC MONITOR

3.1. Overview

Text from this chapter was published in the *Proceedings of International Workshop on Peer-to-Peer Systems*, in April 2009 [17], and *Elsevier Computer Networks, Special Issue on Measurement-based Optimization of P2P Networking and Applications*, in February 2012 [18], Prof. Reza Rejaie, Dr. Yang Guo, Dr. Daniel Stutzbach and I were the primary contributors for this work.

Almost a decade ago, Distributed Hash Tables (DHTs) were presented as an elegant approach to design structured Peer-to-Peer (P2P) networks [7, 31]. Distributed Hash Tables (DHTs) are increasingly incorporated into large-scale Peer-to-Peer (P2P) systems such as eMule [38] and Azureus [39]. As in any other large scale and distributed system, it is very valuable to capture a representative view of system traffic without disrupting its basic operations. Capturing system traffic enables one to identify design flaws, detect performance bottlenecks and determine how users use (or possibly abuse) the system in practice. For example, one may monitor traffic in a DHT to ensure that popular files are properly distributed across the ID space or to check whether botnets command and control messages are not embedded into DHT messages (i.e., a public DHT is not used as the command and control channel of a botnet). Better understanding of existing systems not only allows us to improve their performance, but also sheds light on the design of future DHT systems.

Capturing a representative view of traffic for a large-scale DHT is nevertheless challenging. A common approach is to add instrumented peers that passively

participate in the DHT and log exchanged messages [40, 41]. Deploying a small number of monitoring peers may not provide a representative and sufficiently detailed view of DHT traffic. Alternatively, inserting a large number of instrumented peers artificially increases the number of peers [42], which may disrupt the normal operation of the targeted DHT and may lead to biased measurement results.

We present a new technique for scalable monitoring of DHT traffic, called Montra. The key idea in Montra is to make monitors minimally visible. This minimizes the disruption of monitors on the system and significantly reduces the required resources for each monitor. We implement Montra in the form of a highly parallel, scalable, python based client and validate it over Kad and Azureus DHTs. We show that our implementation of Montra can concurrently monitor around 32,000 peers using a moderately configured PC (an Intel Core 2 Duo with 1 GB RAM), while dropping 0.009% percent of packets.

The remainder of the chapter is organized as follows: We review other DHT monitoring techniques in Section 3.2.; we describe the design of Montra and related issues in Section 3.3.; validation of Montra is described in Section 3.4.; we present characterization of traffic in our target DHTs using Montra in Section 3.5., and we identify the weaknesses of P2P systems in Section 3.6..

3.2. Related Work

Over the past several years, the area of P2P measurement and characterization has attracted a lot of attention and focus. However, only in the past few years have DHTs been widely-deployed, and, hence, there have been only a few measurement studies of DHTs. Falkner et al. [40] and Qiao and Bustamante [41] are the studies that we are aware of that focus on observing the traffic in DHTs. Both studies use

deployment of passive instrumented peers to collect traffic samples. Falkner et al. use 258 peers (250 from PlanetLab and 8 at the University of Washington) while Qiao and Bustamante use 4 geographically distributed sites to collect traffic samples. Both the studies focus on different DHTs from our study, which focuses on Kad.

Falkner et al. focus on the Azureus DHT. Azureus is a popular BitTorrent client and employs a Kademlia-based DHT to allow peers downloading the same content to locate one another without requiring a rendezvous point ("tracker") to be known *a priori*, similar to Kad's File Publish and File Search operations. Their study [40] focuses on core DHT characteristics such as the session length of peers, policies for including new peers in the DHT, and the availability of content.

Qiao and Bustamante [41] focus on the Kademlia-based Overnet DHT, which is also used for file sharing and provided the inspiration for Kad. Their study characterizes those DHT features that are related to finding files, such as the success and failure of queries and the overhead of query traffic. Qiao and Bustamante also evaluate the existence of keyword based hot-spots and conclude that while such hot-spots do exist, they do not affect the peers.

Montra bears some similarities to Mistral [42], developed by Steiner et al. as an attack to disrupt a DHT through the strategic position of malicious peers. While the two have radically different goals, both rely on crawling the DHT and using protocol features to add monitors to the routing tables of existing peers. However, Montra makes use of MVMs to monitor the network without causing disruption and gracefully cope with churn. Mistral, on the other hand, is designed to cause disruption.

Other researchers have focused on non-traffic measurement studies of DHTs. For example, Steiner et al. developed Blizzard [11], which is a crawler for Kad. The data collected from Blizzard is used to analyze such peer properties as population, session

time, and change in ID as well as IP addresses. Stutzbach et al. [43], explored lookup performance in Kad by studying several variations of the Kademlia lookup algorithm over the real Kad network and introduced a Kad variation of their P2P crawler, Cruiser [44].

Earlier traffic monitoring studies targeted unstructured P2P systems. These studies can be divided into two classes: application level and network level. Application level studies employ a few instrumented peers and characterize the traffic observed by those peers. Examples of such studies include: [45], [46], and [47]. Network Level studies, on the other hand, monitor and characterize P2P traffic by observing traffic passing through a router at an ISP. For example, Gummadi et al. [48] monitor Gnutella and Napster traffic entering and exiting a campus network by placing a traffic monitor at the edge of the campus network, Sen and Wang [49] monitor traffic at a backbone ISP, and Karagiannis et al. [50] monitor two different OC48 (2.5Gbps) links of a tier 1 ISP.

Montra is the first tool for efficiently monitoring a large number of peers while minimally disrupting the existing DHT network. Using Montra, we monitored the destination traffic to approximately 32,000 peers simultaneously, more than two orders of magnitude greater than the 258 instrumented peers employed by Falkner et al. [40].

3.3. Methodology

A common approach for capturing traffic in a DHT is to randomly place a few instrumented peers within the network. Each instrumented peer passively monitors and logs traffic. Using a small number of monitors may not provide a representative view of traffic. Alternatively, inserting a large number of monitors may be infeasible

due to the required computational resources. More importantly, such a brute force technique artificially increases the number of peers, and could disturb the traffic pattern [42].

To resolve this conundrum, we notice that traffic can be classified into two categories:

- *Destination traffic*: A lookup request received by peer P_i is destination traffic if P_i's ID is the closest to the target ID in the message using DHT's closeness metric (e.g., XOR distance in Kad).

- *Routing traffic*: Any request that is not destination traffic, is routed toward its destination by peer P_i and is considered as routing traffic for peer P_i.

The rate of destination traffic at peer P_i depends on the popularity of the content that is mapped to P_i's assigned ID space, i.e., destination traffic is primarily user-driven. However, the rate of routing traffic at each peer is determined by that peer's connectivity and overall content popularity. We focus on measuring destination traffic because it represents user behavior and more importantly, because it is much more tractable. In the rest of this section, first, we first describe how to monitor the destination traffic at a single Kad peer. We then discuss how to minimize the visibility of monitors in order to avoid any disruption in the target DHT. Next, we describe how Montra handles peer churn. After that, we comment on applicability of Montra to other DHTs. Finally, we describe an extension to Montra for obtaining content metadata from Kad.

3.3.1. Monitoring a Single Kad Peer

Suppose peer P_t is an arbitrary target peer in Kad. Let A_t be the portion of Kad address space assigned to P_t. Typically P_t is the closest node to the identifiers in A_t. Let P_m be the monitoring peer of target peer P_t. When peer P_t receives a request message from a request originator, P_r, it responds with the set of peers from its routing table that are closest to the requested ID. P_m captures this destination traffic in the following three steps as shown in Figure 3.1.:

1. At the start of the monitoring process, the monitor P_m introduces itself in the DHT in following 2 steps:

   (a) P_m places itself next to the target peer P_t in the ID space by setting its ID as $ID(P_m) = ID(P_t)$ XOR 1

   (b) P_m adds itself to the target peer's routing table by exchanging Hello messages to become visible to the target peer. [1]

2. When P_t receives a request from the peer P_r, with a destination in A_t, P_t replies with P_m's address because according to P_t's routing table, P_m is one of the closest peers to the requested ID.

3. When P_r learns about P_m, it sends the same request to P_m. Thus, P_m receives a copy of requests destined for A_t. P_r sends a request to P_m because it is looking for several alive peers close to the target ID, in order to publish content at or retrieve search results from multiple peers.

[1] Further details on the required message exchange between the target and monitor peer can be found in [1].

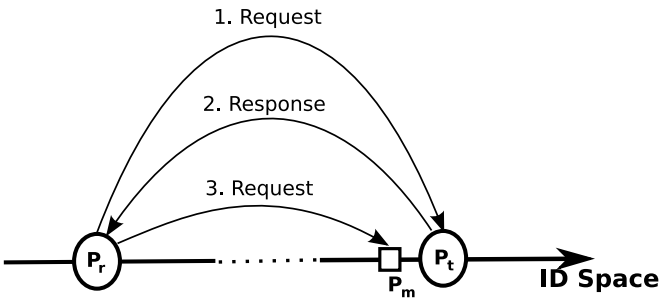


FIGURE 3.1. Message exchanges for adding a monitor
and capturing destination traffic

3.3.2. Minimally Visible Monitors

As mentioned earlier, large number of monitors, while necessary to collect a complete view of the system, may change and/or disrupt the system. We solve this problem by introducing *Minimally Visible Monitors (or MVMs)*. The basic idea is to minimize the visibility of each monitor, P_m, by maintaining its presence only at its target peer, P_t. P_m only responds to the messages issued by P_t and silently ignores messages from all other peers. Peers that may learn about MVM, from P_t, consider P_m as departed peer and do not add it to their routing tables. As a result, an MVM neither routes traffic nor stores content. Since MVMs are essentially invisible, placing a large number of MVMs does not cause disruption and/or change the system. Furthermore, the lightweight nature of MVMs helps in deploying large number of monitors while using minimal resources.

3.3.3. Identifying Destination Traffic

In addition to receiving the majority of destination traffic, P_m may also receive a small fraction of P_t's routing traffic. This occurs when P_m is one of the top c contacts in P_t's routing table for some routing requests. For example, this scenario may occur at one hop before a request reaches its destination. A single MVM cannot distinguish between routing traffic and destination traffic. It has no information whether any other peer closer to the target ID received the same message. In order to accurately distinguish destination traffic from routing traffic, we monitor peers in a continuous region of the ID space. The continuous ID space is called the *monitoring zone*. A zone is specified by x high order bits of the ID (or *prefix*) that is common among all peers in that zone. For example, 0x$a4$ is a prefix for an 8 bit zone ($x = 8$). Any requests for an ID that has the same zone prefix and enters the monitored zone has a destination in that zone. Therefore, by monitoring all the peers in the entire zone, we capture all destination traffic for the zone. Although a request may be observed by multiple MVMs in a zone, during the post processing phase the closest MVM that received the request is considered its actual destination.

3.3.4. Coping with Churn

To accurately monitor the destination traffic at all peers within a given zone in the presence of churn, it is important to quickly identify newly arriving peers and attach a monitor to them. Interestingly, we do not need to remove MVMs for departing peers. Since each MVM is only visible to its target peer, an MVM does not receive any traffic after its target peer departs. Using Stutzbach et al.'s [44] high speed crawler, we crawl the target zone back-to-back in order to ensure timely discovery of new and departing peers. Then, we attach a new monitor to each new peer and place

the monitors of departed peers into the pool of idle monitors for efficient resource management. Note that we only attach monitors to those peers that are not behind NAT boxes. Peers behind NAT boxes do not participate in routing lookup messages.

3.3.5. Generality of Montra

While this work focuses on monitoring Kad, we believe that our proposed technique can be adapted to other real-world DHTs (e.g., Azureus [39] [2], Mojito [51] [3]). Real-world DHTs must ensure the availability of content in the presence of churn. A common technique to achieve this is to lookup multiple peers close to the target ID for both searching and publishing content, as described in Section 2.2.1.1.. Montra leverages the need to "lookup multiple nodes" to capture each lookup message. Note that actual publish and search messages are not observed by an MVM since they are sent directly to the target peer. However, the lookup message often contains some information (e.g., number of requested contacts in Kad lookup) that reveals whether it is associated with a publish or search message.

3.3.6. Extracting Content Metadata in Kad

While monitoring Kad, the following information can be extracted from the captured Request messages: *(i)* the type of request (publish or search), *(ii)* the requested content ID, and *(iii)* the ID of the destination peer. From this information, however, we are unable to determine whether a request (search or publish) is for a keyword or a file. Neither can we learn about the characteristics of requested files (e.g., size).

[2] Azureus uses an implementation of Kademlia to operate in trackerless mode.

[3] Mojito is an implementation of Kademlia and is used by Limewire.

We extend our measurement technique to collect more information as shown in Figure 3.2.. When an MVM receives a Request message from the request originator during the lookup phase, it may send a response to the originator. The response does not carry any next hop contacts, but it informs the request originator that the MVM is alive and can receive a request during the second phase. As a result, the MVM receives the (search or publish) requests during the Search/Publish phase that carry the following additional information about the files or keywords being requested: *(i)* the type of content (file or keyword) being requested/published, and *(ii)* the size of the file when the requested content is a file.

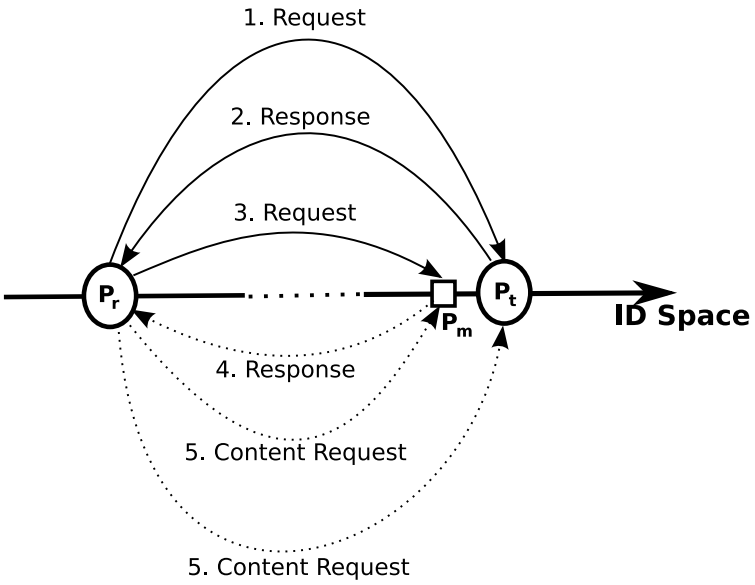


FIGURE 3.2. Extended Technique

This extension, shown with dotted lines in Figure 3.2., results in a slight disruption to regular operation of the system. More specifically, content that is published at the 10 closest peers, is instead published at 9 closest peers. In addition, sending Response to a Request message increases the visibility of the MVM. To minimize the side effects of this extension, MVMs respond to requests for a given

content ID only once, in order to obtain the above additional information. MVMs do not respond to any further request for the previously observed content ID.

This extension is specific to Kad. We believe Montra, in its original form, can be used to capture traffic from any real-world DHT. However, we also envision such DHT-specific extensions to Montra to extract more fine-grained information from different DHTs.

3.3.6.1. Extracting Content Metadata in Azureus

In **Azureus**, the most challenging aspect is placing MVMs sufficiently close to the target peers. Peer IDs in Azureus are based on a combination of IP address and port number. To cope with this problem, we use a pool of 115 IP addresses from the 128.223.8/24 subnet of the Computer Science Department at the University of Oregon. Each IP address can use a UDP port from the range (1025, 65535). This combination of IP addresses and port numbers yields more than 7 million Azureus peer IDs, providing adequate flexibility to place an MVM near any of the 300,000 peers in Azureus.

Finally, we note that Montra cannot capture *get* requests from the Azureus DHT. A get request combines lookup and retrieval phase, and terminates after finding 8 other peers participating in the same swarm, as described in Section 3.5.. Thus, with high probability, a get request may terminate before reaching the closest possible peer to the swarm ID, i.e., the true destination. Montra cannot capture those requests that do not reach their true destination. In the rest of the chapter, all references to Azureus requests refer to *put* requests.

3.3.7. Monitoring Overhead

The messages that are sent to individual MVMs increase the overall traffic associated with the target DHT. These extra messages can be viewed as monitoring overhead. We can estimate the percentage of this overhead as a function of DHT's relevant properties. Suppose that the average distance (i.e., the number of rounds of routing) between two peers in a DHT is d and the level of redundancy in routing messages (i.e., the number of routing messages sent in each round) is r. Therefore, the total number of routing messages to identify proper destinations during the lookup phase is on average $r*d$. Ideally, out of these messages only one is sent to an MVM. Thus, the monitoring overhead is $\frac{1}{r*d}$. This simple model enables us to assess the basic overhead of Montra for a given DHT. For example, in Kad, the level of routing redundancy is $r = 3$, and the average number of hops between two peers is d>3.2 [43]. This suggests that the basic overhead of Montra for Kad is 10.41%.

In practice, multiple MVMs close to the destination ID may receive the same routing messages during the lookup phase. This in turn increases the overhead of Montra but the effect is DHT specific. We quantify this effect by examining the distribution of the number of duplicate messages among the MVMs in a single 6-bit zone in Kad and repeat this analysis for ten 6-bit zones. Figure 3.3. presents all ten distribution of number of duplicate messages received by MVM in each zone by showing the minimum and maximum y values (for each x value). In essence, the derived CDF for each targeted zone is between the min and max lines in Figure 3.3.. The min and max lines are very close to each other in Figure 3.3.. This suggests that the distribution of the number of duplicate messages per zone is very stable (i.e., it is zone independent). Figure 3.3. shows that for 90% of lookup events, 3 or less MVMs receive routing messages, while the average number MVMs that receive

routing messages is 2.17. This implies the actual overhead of Montra for Kad is 10.41% * 2.17 or 22.6%. The extended version of Montra for Kad has slightly more overhead since it exchanges additional messages with other peers. However, since these additional messages are sent only once per content ID, this overhead is not significant. The examination of MVM logs revealed that this addition is only 13%.

In conclusion, the monitoring strategy in Montra increases the DHT traffic. The actual amount of increase depends on the details of the targeted DHT. It is important to note that this overhead is not likely to increase with the size of the DHT since both the value of d and the number of duplicate messages are likely to increase. This suggests that the total overhead is likely to be independent of peer population. Furthermore, we are not aware of any other DHT traffic monitoring approaches that are capable of monitoring the traffic in large scale DHTs without disrupting the system or causing a significantly larger overhead than our approach.

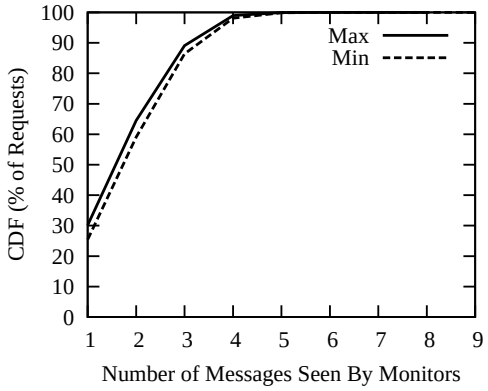
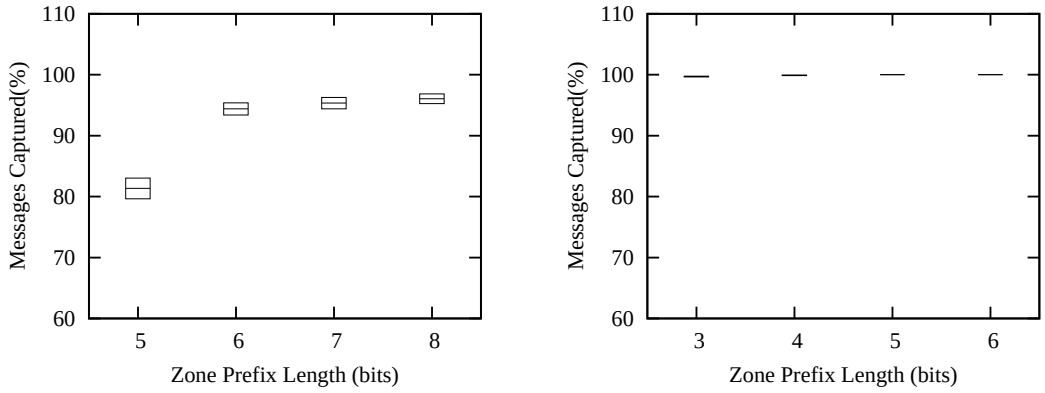


FIGURE 3.3. Routing Overhead in Kad

3.4. Validation of Montra

In this section, we quantify the percentage of traffic that Montra can capture. First, we deploy MVMs in a randomly selected zone in a DHT. Next, we run an



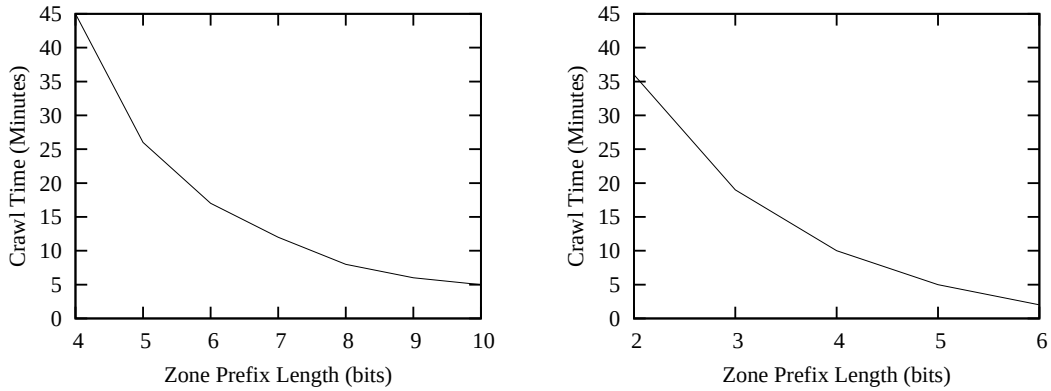
(a) Kad (b) Azureus

FIGURE 3.4. Montra Validations

instrumented peer with a randomly selected ID outside the monitored zone. The instrumented peer sends a large number of lookup requests towards random IDs within the monitored zone. Finally, we calculate the percentage of issued requests that are successfully captured by Montra. In addition to evaluating the accuracy, we also want to identify the largest possible zone that Montra can accurately monitor. Therefore, we explore accuracy as a function of zone size.

The validation results for the Kad and Azureus DHTs are shown in Figures 3.4.a and 3.4.b, respectively. Both figures show the zone prefix length on the x-axis and the percentage of successfully captured requests on the y-axis. The zone size doubles each time the zone prefix decreases by one bit. The top and bottom lines of each box reflect the 95% confidence interval and the middle line is the mean.

Figure 3.4.a reveals that Montra captures almost 95% of destination traffic in Kad when the prefix length is 6 bits or larger. However, as the prefix length falls below 6 bits, Montra's accuracy drops significantly. For longer prefix lengths, Montra does not capture 100% traffic because *stale peers* exist in the routing table of alive peers. Stale peers are those departed peers whose departure has not been detected and are



(a) Kad (b) Azureus

FIGURE 3.5. Crawl Times

thus not removed. Stale peers prevent the lookup traffic from reaching the closest possible destination. If the traffic does not reach its true destination, then Montra cannot capture it. For larger zones, Montra's accuracy is affected significantly because of the large crawl time to discover all peers. As the prefix length decreases, it takes longer to crawl the ID space, as shown in Figure 3.5.a. Longer crawling times lead to longer delays in discovering new peers and attaching monitors to them which in turn reduces accuracy. Based on these results, we monitor 6-bit zones.

Figure 3.4.b indicates that Montra intercepts almost all Azureus destination traffic regardless of zone size. Unlike Kad, neither the effect of stale peers nor crawl time is evident. Given the trend of crawl time shown in Figure 3.5., we believe the effect of crawl time would become visible if we evaluated Montra for larger Azureus zones. However, we were unable to test Montra over Azureus prefixes shorter than 3 bits due to the volume of incoming traffic (around 50 Mbps).

The effect of churn is absent in Azureus because of the lack of timeouts. Azureus peers allow the requests to reach the true destination, and thus route around any stale entries in the path. If a request reaches its true destination, then Montra can

capture it. In summary, these results reveal that monitoring 3-bit zones in Azureus is the most appropriate given our bandwidth limitations.

3.5. Characterization of Traffic

In this section, we first describe the collected data sets then present different characteristics of Kad and Azureus DHTs.

3.5.1. Data Sets

The granularity of Montra (described in Section 3.4.) allows us to monitor $\frac{1}{64}$ of Kad (32,000 peers) and $\frac{1}{8}$ of Azureus (37,000 peers) ID space at any point of time. In a series of observations, we monitored each of the 64 Kad zones and 8 Azureus zones. Each set of observations is 6 hours in length. To ensure robustness to any time-of-day effects, the observations began at different times of the day (3pm, 9pm, 3am, or 9am) chosen at random.[4] The Kad dataset covers May 2007 through October 2008 while the Azureus dataset covers June 2009. In this section, we leverage these datasets to examine several characteristics of traffic in the Kad and Azureus DHTs

3.5.2. Characteristics

Plotting the Cumulative Distribution Function (CDF), or its complement (CCDF) for every dataset on one graph to compare them is impractical due to the large number of datasets. In order to present the data succinctly, we plot the minimum and maximum y values (for each x value) across all distributions, resulting in two lines per graph that succinctly capture the data. The CDF and CCDF for each dataset falls somewhere between the minimum and maximum lines. In general, we found that

[4]All times in this chapter are in Pacific Time.

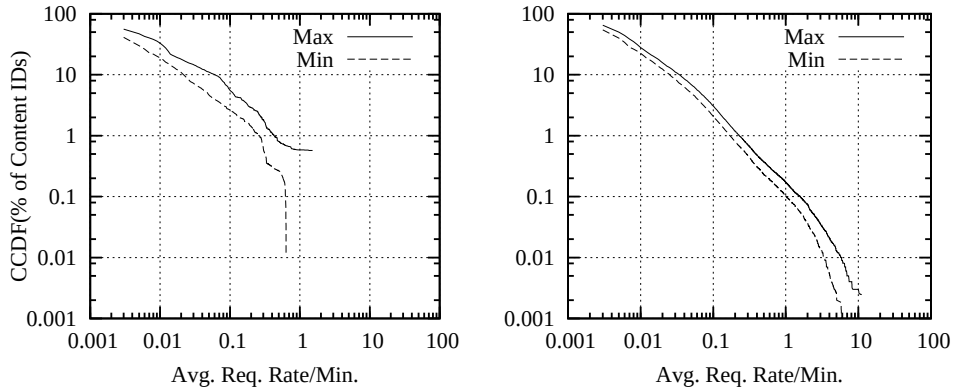


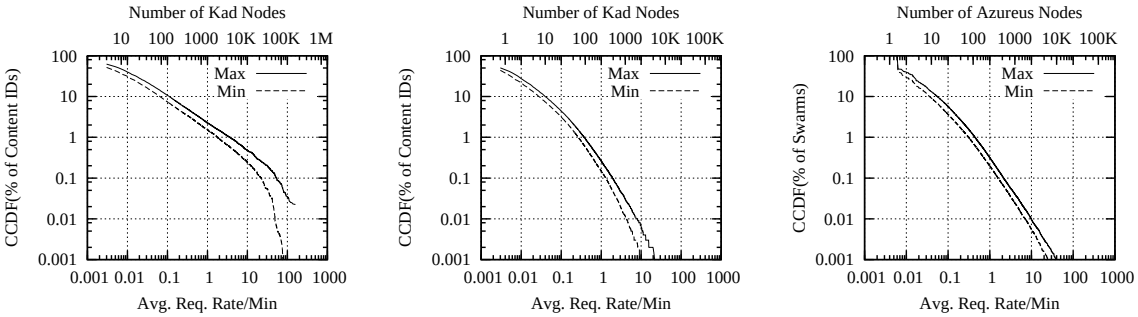
FIGURE 3.6. Kad Search Rates: a. Keywords, b. Files

the minimum and maximum lines are relatively close together, demonstrating little variation between zones.

3.5.2.1. Message Rates

To begin our study, we examine the rates at which different content IDs are requested. As discussed in Section II, there are two basic types of messages monitored by Montra: *(i)* search and *(ii)* publish. Montra cannot capture *search* requests from the Azureus DHT. A get request combines lookup and retrieval phase, and terminates after finding 8 other peers participating in the same swarm. Thus, with high probability, a get request may terminate before reaching the closest possible peer to the swarm ID, i.e., the true destination. Montra cannot capture those requests that do not reach their true destination.

Figure 3.7. shows the distributions of publish requests over the set of IDs, as CCDFs in log-log scale. Kad and Azureus both send automatic publish messages at 4-hour intervals, allowing us to estimate the number of peers publishing each piece of content. Given the observed request rate and the known re-publish interval (based on source code inspection of the eMule client), we can compute an estimate of the number



(a) Kad Publish Keyword (b) Kad Publish File (c) Azureus Publish File

FIGURE 3.7. Publish Request Rates

of Kad nodes in the system who possess the resource. For example, some popular files are published at the rate of 30 requests per minute. Since each source sends a Publish File message once every four hours, this rate is equivalent to approximately 72,00 concurrent Kad nodes who possess a file copy. The number of peers are presented as a second x-axis along the top of each graph.[5] Figure 3.7.a shows that 10% of published keyword IDs have a request rate of more than 0.1 per minute, while 0.1% have a rate of more than 30 per minute. For all three types, the distribution is heavily skewed, with the vast majority of messages targeting a tiny fraction of the observed content IDs. In other words, a tiny fraction of content is widely popular, while the majority of content is found on only a few peers. This heavily skewed distribution is consistent with observations of other file-sharing systems [52].

Figure 3.6. shows the distribution of search requests over the set of IDs, as CCDFs in log-log scale. Comparison of Figures 3.7. and 3.6. shows the rate of publish requests is much higher than the rate of search requests. For example, some keywords have a publish request rate greater than 100 requests per minute, while the highest observed keyword search request rate is less than 2 requests per minute.

[5]Departure and arrival of peers will cause some error in these values, but they are a useful first-order approximation.

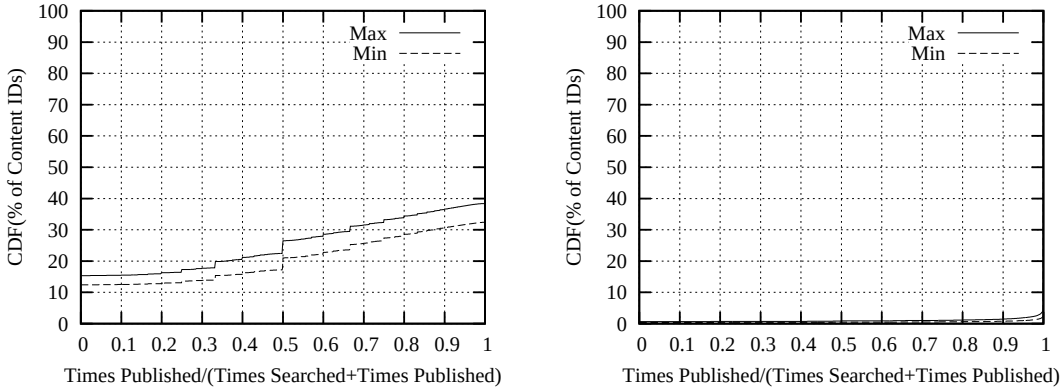
Publish requests are automatically generated, while search requests are manually generated. Thus, the majority of request messages are generated by Kad nodes for the purpose of protocol maintenance. The traffic generated by direct user activity only accounts for a small percentage of total traffic.

3.5.2.2. Relation Between Published & Searched Content

A DHT, in essence, provides a distributed rendezvous mechanism for content contributors (publishers) and consumers (searchers). Publish requests for a given content demonstrate its availability, whereas search requests represent the demand. This section examines the balance between availability and demand for individual content. If a file is excessively published but never searched, then that file has lost its popularity and is now easily available in the system. On the other hand, if a file is searched by a large population but is never published then that file has gained popularity recently and is not available in the system yet. This section examines the balance between availability and demand for content.

We explore this balance using the formula $(\frac{P}{S+P})$ for each content ID, where P is No. of Publish Requests and S is No. of Search Requests observed during a measurement window. In order to find true availability of a given content we discard duplicate publish requests from the same publisher. We use IP, port combination to identify content publishers. Duplicate search requests, unlike duplicate publish requests, show true demand. Therefore, we do not discard search requests. Figure 3.8.a and 3.8.b show the CDF of $(\frac{P}{S+P})$ per content ID for files and keywords, respectively. Interestingly, Figure 3.8.a reveals that 15% of files are searched but never published $(\frac{P}{S+P} = 0)$. Most likely, these files became recently popular and thus are not widely-available in the system. Figure 3.8.a also show that 60% of files are published

but never searched during our measurement window ($\frac{P}{S+P} = 1$). These files are very well-spread in the system and therefore are rarely searched. Figure 3.8.b shows that 95% of keywords are published but never searched during our measurement window ($\frac{P}{S+P} = 1$). This shows that a very small fraction of keywords is actually used for search and a lot of resources are wasted on publishing additional keywords.



(a) Publish vs Search Requests for Files (b) Publish vs Search Requests for Keywords

FIGURE 3.8. Relation Between Published & Searched Content

3.5.2.3. Radius of Hot IDs

We define hot IDs as extremely popular content IDs. The skewed distributions shown in Figure 3.7. and 3.6. demonstrate that hot IDs exist. The largest impact of a given hot ID is observed by its destination peer, which will receive several orders of magnitude more traffic than a typical peer. However, peers close to the destination are also affected because they route the traffic. We define the radius of a hot ID region as the distance between the destination peer and the farthest peer that observes a dramatically increased request rate. In this subsection we empirically characterize the radius of hot ID regions in Kad.

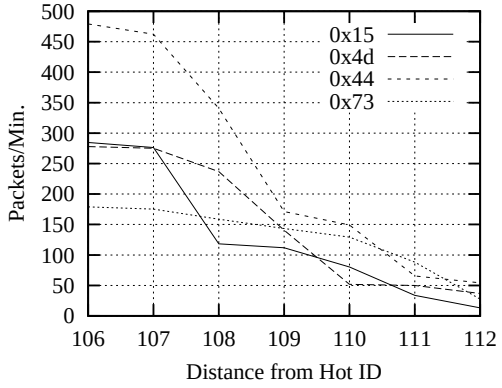


FIGURE 3.9. Radius of Hot IDs

We examine the radii of the four most popularly published keywords from four different zones. We choose publish keyword requests since they are the largest source of DHT traffic in Kad. We measure the radius by placing 7 instrumented Kad clients at varying distances from each hot ID. An instrumented client is a regular Kad client modified to log all incoming request messages. The key idea is to place enough instrumented clients at varying distances such that we can observe the variation in traffic rate. We place the first instrumented client at a distance of 106 bits. This client has 22 high order bits in common with the hot ID (106 + 22 = 128), which in most cases is sufficient to make it the closest peer to the ID. We place the remaining 6 clients at increasing distances of 107 bits, 108 bits and so on. Figure 3.9. shows the radii for four measured hot IDs. The graph legend shows the 6-bit zone prefixes from which the hot IDs are chosen. The figure shows that the traffic declines steeply as a peer moves further from the hot ID, from 250–500 messages per minute down to 5–50 messages/minute. The sharp decrease is because of the exponential increase in number of peers that can route the traffic towards the destination.

We were not able to conduct this analysis for the Azureus DHT because of the lack of peer IDs that can be varied bit by bit. The pool of 7 million peer IDs,

mentioned in Section 2.2.1.2., is enough to monitor the destination traffic of individual peers. However, it is an extremely small percentage of total Azureus peer IDs. Thus, we cannot select the peer IDs with enough precision.

3.5.2.4. Swarm Participation per Peer

Next, we explore the number of swarms in which each Azureus peer participates. Figure 3.10. shows the CCDF of the number of swarms per peer on a log-log scale. The x-axis shows the number of swarms and the y-axis shows the percentage of peers. We identify a peer using its IP address, instead of IP:port combination, so that anomalous peers using a large number of ports cannot bias the results. The data shows that most peers participate in a small number of swarms, while a tiny percentage of peers participate in a large number of swarms. For example, 50% of peers participate in six or fewer swarms, while 0.001% of peers participate in 1,000 or more swarms.

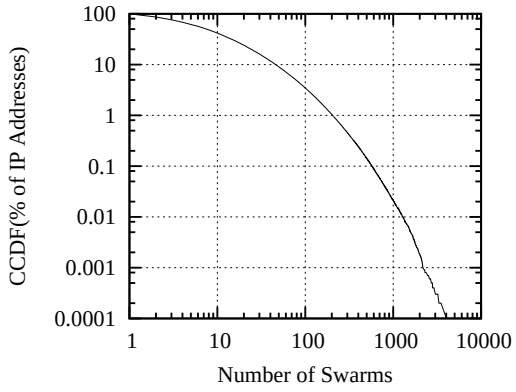
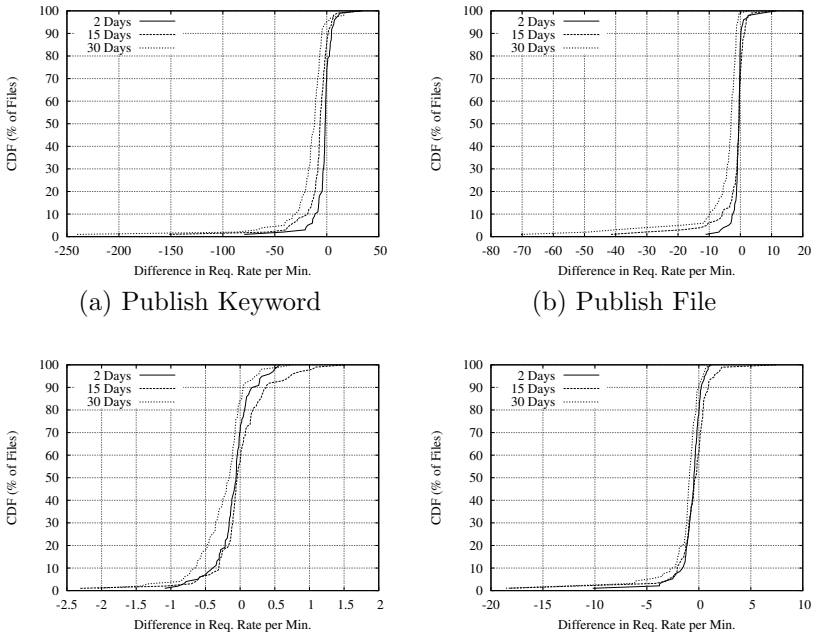


FIGURE 3.10. Swarm Participation

This analysis cannot be performed for Kad because NAT peers use "buddies" for publishing files. If several Kad peers publishing the same file choose the same buddy

then the population of the file will be underestimated. On the other hand, Azureus peers avoid the effect of NAT, as mentioned in Section 2.2.1.1..



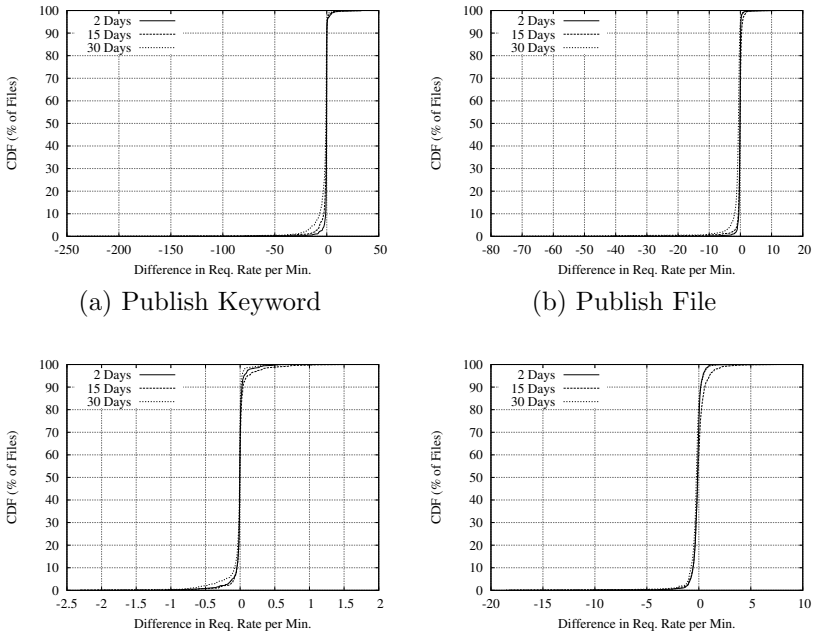
(a) Publish Keyword (b) Publish File

(c) Search Keyword (d) Search File

FIGURE 3.11. Change in popularity for top 100 Kad files

3.5.2.5. Evolution of Availability & Popularity of Individual Files

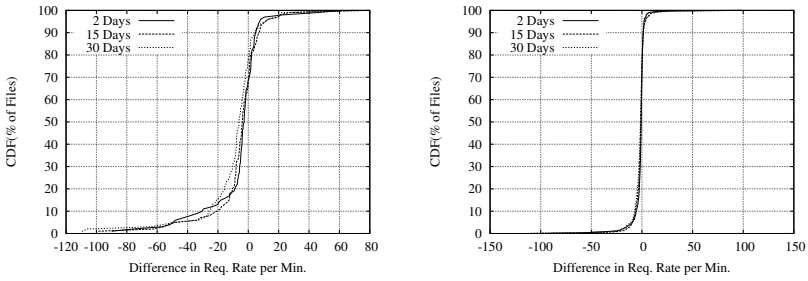
In order to understand how content availability and popularity evolve over time, we study the temporal property of content request rates for the Kad and Azureus DHTs. For both the DHTs we monitor a given zone for 6 hours every 2 days for 30 days. We refer to each 6-hour dataset as a *snapshot*. For each snapshot, we can leverage the publish rate of individual keywords or files to determine their availability. Similarly, we can use the rate of search keyword or search file messages to determine their popularity (i.e., demand) at any point of time. If we consider the first snapshot as a reference, comparing later snapshots with the reference reveals any change in



(a) Publish Keyword (b) Publish File

(c) Search Keyword (d) Search File

FIGURE 3.12. Change in popularity for top 1000 Kad files



(a) Top 100 (b) Top 1000

FIGURE 3.13. Change in popularity for top 100 and 1000 Azureus files

the availability or popularity of available files over time as well as any new files that might have become available.

Figure 3.11.a and3.11.b show the distribution of change in availability of individual keywords and files while Figure 3.11.c and Figure 3.11.d show the change in popularity of top-100 keywords and files, respectively. Each figure depicts the corresponding changes over three intervals 2, 15 and 30 days. The negative values on x-axis show a decrease in availability and vice versa. Figure 3.11.a and Figure 3.11.b reveal that the availability of content gradually decreases over time. For example, after 2, 15 and 30 days 40%, 80% and 95% of keywords become less available, respectively. The publish rate for a few keywords drops by approximately 250 requests per minute after 1 month that suggests these keywords lose roughly 300,000 publishing peers. A majority of keywords lose 10 to 50 requests per minute (14,400 to 72,000 publishers) after 1 month. The evolution of popularity for top-100 keywords and files in Figure 3.11.c and 3.11.d exhibit a similar decreasing trend with time but with the varied pace.

We have conducted the same analysis for Azureus. Figure 3.13.a and 3.13.b show the change in the availability of top 100 and top 1000 files in Azureus. Similar to Kad, these results exhibit a gradual decline in the availability of files in Azureus. However, the availability of files in Azureus drops slower than Kad. More specifically, after 30 days the availability of almost all files in Kad drops while such a drop is seen for only 80% of files in Azureus. Furthermore, the examination of top 1000 files in Azureus confirms that top 100 files exhibit a significantly larger change in their availability than in Kad.

To examine the evolution of content availability and popularity over a larger population of files, Figure 3.12. presents the same analysis (as in Figure 3.11.) for

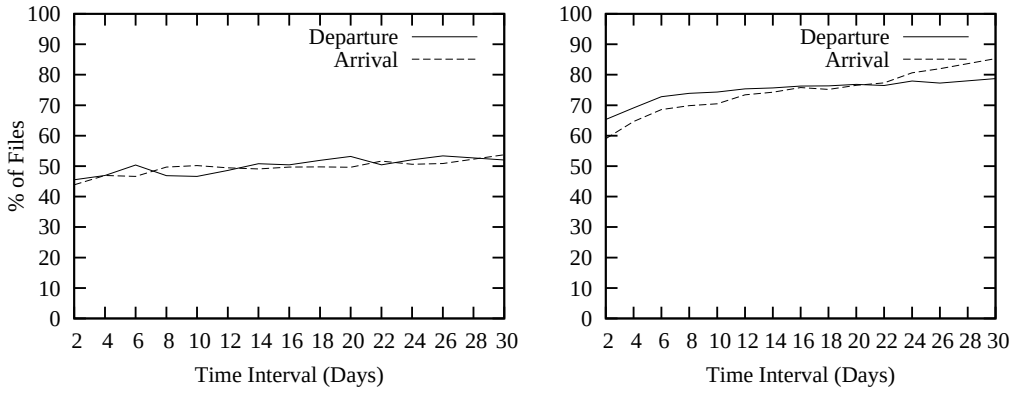


FIGURE 3.14. Arr. and Dep. of Files: a. Kad, b. Azureus

top 1000 files and keywords. This figure show very little change in availability and popularity of top-1000 files and keywords even after 30 days. This suggests that most of the fluctuation in availability and popularity is associated with the top 100 files and keywords.

3.5.2.6. Churn in Aggregate Available Content

In this subsection, we focus on the change in aggregate available content in the DHT over time. We compare any snapshot with the first one (as a reference) to determine what fraction of files have been added or removed from the system. The results for Kad and Azureus DHTs are shown in Figure 3.14., normalized by the total number of files in the reference.

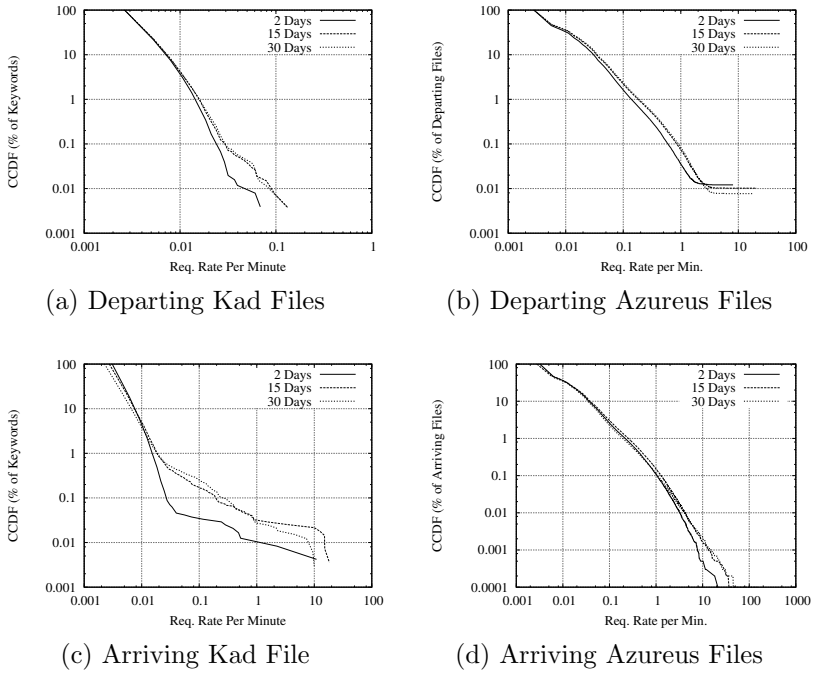
Two qualities are notable about these results. First, in each system the number of newly arrived files balances the number of departing files. Thus, the overall number of files available remains roughly the same even though the selection of files varies dramatically. Second, almost all files that were still present after 2 days were also present after 30 days (resulting in the nearly horizontal lines). Rather than seeing a gradual decay of the original set of files, a large number of files departed during the

first 2 days but few files departed thereafter. Presumably, this effect is caused by the skewed number of peers sharing each file, shown in Figures 3.7.b and 3.7.c. The many files shared by just a few peers become unavailable quickly, while the files shared by many peers remain available.

Figure 3.15. depicts CCDF of the publish rate for files that depart or arrive in each DHT in log-log scale within a particular window of time to provide a micro-level view of file churn. Figure 3.15.a and 3.15.b show that the publish rate for a majority of departing files is low over different windows of time. This implies that files even with extremely low availability can remain in the system for more than 30 days. For example 5-10% of departing published files have the original request rate of 0.01 or less. This result explains why highly available content does not suddenly disappear from the DHT. Figure 3.15.c and 3.15.d indicate that the distribution of publish rate for newly arriving files is very skewed over all time intervals in both the DHTs. A majority of files have a very low publish rate but a small fraction of files reach a very high availability even within a couple of days. For example, roughly 1% of newly arriving files in Kad reach the highest publish rate of 10 requests per minutes (as shown in Figure 3.7.b) even within 2 days.

3.5.2.7. File size

Figure 3.16. shows the envelope of distribution of files sizes for all the published and searched files. The closeness of the min and max lines shows that these distributions are consistent across different zones. Figure 3.16. shows that 55% of searched files are larger than 100 MB. Figure 3.16. also shows that almost 50% of published files are less than or equal to 10 MB. In other words, while many small files are available, users more often request to download large files. Presumably, the



(a) Departing Kad Files (b) Departing Azureus Files

(c) Arriving Kad File (d) Arriving Azureus Files

FIGURE 3.15. Dist. of Publish Rate for Departing and Arriving Files in Kad and Azureus

greater availability of small files is because they can be downloaded more quickly, enabling them to spread more rapidly through the system in response to demand.

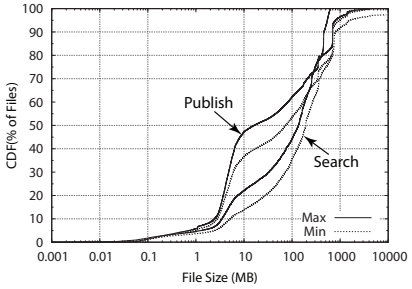


FIGURE 3.16. Distribution of Size of Published and Searched Files

We could not conduct this analysis for Azureus DHT because the Azureus protocol messages do not carry file metadata information.

	Publish Files	Publish Keywords	Search Files	Search Keywords
Italy	22.687	25.343	12.836	17.637
Spain	21.306	21.025	10.594	10.522
France	12.502	13.532	9.838	9.247
Brazil	5.586	7.115	4.591	3.838
China	4.449	3.126	30.105	19.699
Other	33.470	29.859	32.036	39.057

(a) Kad

	Publish Files
US	25.52
France	8.59
Canada	8.33
Great Britain	6.39
Spain	5.11
Other	46.06

(b) Azureus

TABLE 3.1. Percentage of DHT Traffic Originating from Top 5 Countries.

3.5.2.8. Geographical Characteristics

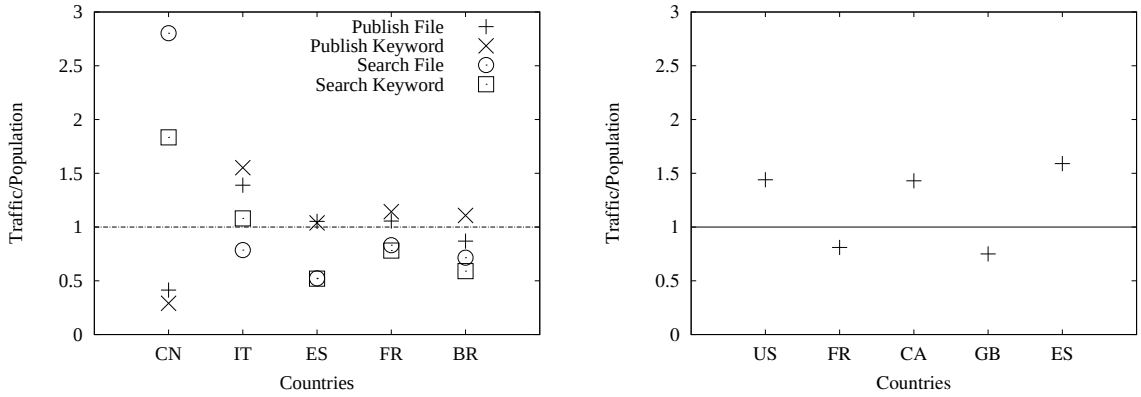
In this subsection, we briefly explore the geographical characteristics of Kad and Azureus traffic. For this analysis, we use 24-hour traffic traces in order to avoid the time-of-day effect. In order to understand the geographical characteristics of captured traffic, we classify the requests by type and country of origin. The results are summarized in Table 3.1. for Kad and Azureus. Table 3.1.a reveals that Italy is the biggest contributor of content (files as well as keywords) to Kad while China is the biggest consumer of content. Almost one third of request (of different type) are originating from other countries. Table 3.1.b presents the breakdown of Azureus traffic across different countries which looks different from Kad. US is the main contributor of traffic in Azureus. Spain and France are among the top 5 contributors but with a much smaller share than Kad. Furthermore, the contribution of traffic

across different countries is less skewed in Azureus since almost half of the traffic is originating from other countries compare to 66% in Kad.

Ideally, when a peer downloads a file, it automatically publishes the file back. Thus, one expects that the contribution and consumption of each country is relatively balanced and it is proportional with the population of peers in that country. However, our results in Table 3.1. do not support this hypothesis. To explore this issue more closely, we quantify the normalized contribution and consumption with the following ratio:

$$\frac{\textit{\% of Total Traffic Generated by the Country}}{\textit{\% of Peers Located in the Country}}$$

Figure 3.17.a depicts the normalized contribution and consumption of top five countries. The normalized value of one indicates the perfect balance between traffic and population for each country. Figure 3.17.a reveals that Italy, Spain, France and Brazil publish proportionally more content and consume less content than fraction of their population whereas China is the exact opposite. Closer examination of our results led to two reasons for the excess consumption by Chinese peers. First, Figure 3.17.a revealed that the contribution-deficit by Chinese peers is very close to the contribution surplus by Italian. This must be due to the fact that Chinese users behind NAT boxes use Italian peers as buddies. Thus in reality, neither Chinese peers are under-contributing nor Italian peers are over-contributing. Second, we identified a single peer in China who searches significantly more often than normal users. When we eliminate this particular peer from the analysis, then the normalized consumption and contribution of China in Figure 3.17.a becomes very close to 1. Figure 3.17.b depicts the normalized consumption and contribution of top 5 countries in Azureus. . This figure shows that US, Canada and Spain contribute more while France and Great Britain contribute less content than their fair share. However, unlike Kad, lower



(a) Kad (b) Azureus

FIGURE 3.17. Traffic to Population Ratios

contribution in Azureus is not an anomaly. Rather it simply shows while lot of peers from France and Great Britain are connected to the DHT, not all are involved in file exchange. Content under-contribution is an anomaly only when it is complimented by content over-consumption.

3.5.2.9. Scale of the Study

The large-scale nature of this study allows us to monitor every single peer in every single zone of the DHT. Naturally, such extensive reach in DHTs is hard to achieve by deploying a few monitors. The scale of our study allows us to understand how different characteristics (e.g., traffic rate) vary across different zones of the DHT. Furthermore, our study helps us in discovering all content-IDs, including both popular and unpopular ones, and we can easily find where popular content is located and how well it is distributed across the DHT.

3.6. Weaknesses of Peer-to-Peer Systems

Our characterization study shows that structured P2P networks suffer from the following weaknesses:

- Heterogeneity: By their very nature, P2P networks allow heterogeneous hosts to participate in the network: a node in a P2P network can be a single desktop machine, or an entire data center. However, P2P networks treat each node as equal, and are not particularly careful about the bandwidth and storage requirements of a node - i.e., a desktop machine may receive lot more traffic than the entire data center. Section 3.5.2.1. demonstrates that there are certain nodes across DHTs that receive orders of magnitude more traffic than other nodes in the network.

- Churn: P2P networks exploit the self-organization property to accommodate churn - i.e., nodes may disappear without prior notice, and may never come back. Section 3.5.2.1. and Section 3.5.2.2. reveal that in their attempt to accommodate churn, P2P networks generate lot of extra traffic, and still fail to serve all the requests.

  Current P2P networks are required to accommodate churn because they assume that regular end hosts will participate in the network. However, real-world network services (e.g., DNS, email) assume servers that are online 24/7, and failures are transient. Thus, if a P2P network is to be used for Internet-scale content discovery, then the design must assume transient failures instead of churn.

- Stretch: The structure of P2P networks dictate where the discovery information for a particular content item must be stored, and the discovery process uses

this structure to resolve a content name into its location. However, the structure of P2P networks is completely disconnected from the underlying network structure, which implies that a content name resolution query may incur high latency, even if the resolution information is only one hop [53] away in the P2P network.

– Membership Management: P2P networks do not control who may or may not join the network. As a result, an entity may inject large number of controlled nodes to cause significant damage to the network, or exploit the network for malicious purposes. Our work on Montra demonstrates that this weakness can be used to monitor traffic, and pollute name to location mappings. Furthermore, our work on Tsunami (described in the next chapter) demonstrates that the same property can be used to create a botnet inside a P2P network.

3.7. Recap

In this chapter we presented Montra, which is a lightweight, scalable technique for capturing DHT traffic. Montra deploys large number of minimally visible monitors, and can capture destination traffic with 90% accuracy. We also characterized the captured traffic, and drew conclusions about weaknesses of structured P2P systems, which hinder the use of structured P2P networks for Internet-scale content discovery.

CHAPTER IV

TSUNAMI: A PARASITIC, INDESTRUCTIBLE PEER-TO-PEER BOTNET

4.1. Overview

Text from this chapter was published in *Peer-to-Peer Networking and Applications*, in February 2013 [19]. My advisor Prof. Jun Li, Prof. Reza Rejaie and I were the primary contributors for this work. In this chapter we present the design and evaluation of a new botnet, called Tsunami, in order to demonstrate the lethality of lack of membership management in P2P systems.

Botnets—i.e., networks of compromised computers called bots—are one of the most challenging network security problems of today's Internet. Their threat is clear: a botmaster can use a command and control (C&C) channel to direct thousands or even millions of bots to launch large-scale security attacks (e.g., DDoS, spam campaign).

The most critical component of botnets is their C&C channel. While botnets have employed several different types of C&C channels, all C&C channels so far rely on a central server or bootstrap nodes to operate, and such "bottleneck" nodes can be the weakest points of a botnet. For example, a centralized botnet (e.g., an IRC botnet [54] or a HTTP botnet [54]) can stop functioning if its central server is located and shut down, and a peer-to-peer botnet (e.g., Nugache [55], Phatbot [56]) will not be able to recruit new bots if its bootstrap nodes are captured. Although there have been various forms of botnets, such as a hybrid botnet (e.g., Storm [57], Alureon [58]) that combines the centralized botnet and the peer-to-peer botnet, or a botnet that

creates many redundant central servers (e.g., Conficker [59]) or bootstrap nodes (e.g., Nugache), they still rely on certain types of bottleneck nodes.

We introduce a new type of botnet in this dissertation, called Tsunami, which does not have a central server or bootstrap nodes, or any types of bottleneck nodes, and every bot is equal to each other. If bots are not assigned special roles (e.g., botmasters), then elimination of particular class of bots cannot destroy the botnet. Without bottlenecks to target, the only way to bring down such a botnet is to remove all the bots, which can become a vast amount of work if such a botnet consists of a very large number of bots. Moreover, this hard-to-destroy feature holds true even if one may identify the bots on this botnet or discover its C&C traffic.

Instead of building *dedicated* C&C channels for communication, Tsunami acts as a parasite, and exploits a widely deployed, benign distributed system as its host. It sprinkles its bots randomly across the benign host system, and uses the built-in communication channels of the host system for its own C&C. For this research, we choose *Kad* to be the host system for Tsunami.

While a centralized botnet may also piggyback on other communication platforms, such as the IRC botnet or the HTTP botnet, this new botnet is more advanced by avoiding the need of a central server as required by a centralized botnet. It will still have a botmaster, of course, but unlike the botmaster in a centralized botnet and many P2P botnets, the botmaster here does not need to be always online. It can simply inject its command and run away.

This new botnet is also more dangerous than today's peer-to-peer botnets, which all have their own dedicated network. Not only does the new botnet avoid the need of the bootstrap nodes as required by these peer-to-peer botnets, it also avoids the need of building its own dedicated C&C channels.

We study how Tsunami can accomplish a C&C channel without points of failure by running on top of Kad as a parasitic botnet. It is a parasitic botnet because it consumes resources (e.g., bandwidth), and may not contribute anything back to the network. Specifically, we explore how Tsunami can use Kad to issue its commands to millions of bots, receive responses from them, and conduct surreptitious but damaging functions as needed. Also, using Kad as the host system, we can study the security of Kad, including how a P2P network such as Kad can be exploited and become an involuntary host of a botnet.

We further evaluate Tsunami to show that when 5% of Kad nodes are Tsunami bots, a command can reach virtually all of them in less than six minutes. Even when Tsunami bots experience churn (bots arrive and depart all the time), with 8% Kad nodes that are bots, a Tsunami botmaster can still reach 80% bots in 3–4 minutes.

In addition to designing the botnet, we also study how to defend against Tsunami. Since capturing and destroying Tsunami bots is extremely hard and costly, we study how we may capture the Tsunami traffic, mislead the bots, and thus disrupt their C&C communication.

The rest of this chapter is organized as follows. We first summarize the related work in Section 4.2.. Then, we describe the design of Tsunami in Section 4.3. and two examples of launching attacks via Tsunami in Section 4.4.. We evaluate Tsunami's performance in Section 4.5., and describe our current implementation of Tsunami in Section 4.6.. Section 4.7. discuses our defense strategies against Tsunami.

4.2. Existing Botnet Construction Techniques

Most existing botnets can be disrupted because botmasters and bots maintain explicit or implicit knowledge about each other. In this section we classify existing

botnets and highlight generic disruption techniques against these threats. We divide existing botnets in following 3 classes:

4.2.1. Centralized C&C

These botnets use a centralized botmaster. The bots maintain permanent or periodic TCP connections with the botmaster in order to receive new commands. Two examples of such botnets are Mybot and Gobot [54]. This design, while very efficient, makes it hard for the botmaster to hide its identity. If a single binary that carries the location of the C&C server(s) is captured then the C&C servers can be located and shutdown. Even if the binary does not possess the location of the C&C servers, a bot's traffic pattern may reveal the location of the C&C servers.

4.2.2. Hybrid C&C

These botnets use a mixture of centralized and decentralized design. The basic idea is to find the centralized C&C servers through a decentralized mechanism (e.g., P2P networks). Since this design includes a central component, all the measures used to defend against centralized botnets can be used to defend against hybrid botnets. Still this design is an improvement over the centralized botnet design. Such a mechanism is absent in the centralized botnet design. To the best of our knowledge, Storm [57] is the only botnet that employs a decentralized design. Storm uses a public DHT as a rendezvous mechanism for the bots to discover the location of the centralized botmaster. The rendezvous location is determined by the bots by using an algorithm. Using the same algorithm the defenders may find the location of the centralized server and destroy it.

4.2.3. Peer to Peer (P2P) C&C

Botmasters have recently started using P2P networks to avoid any central points of disruption. Bots implicitly maintain information about each other, which is a side-effect of network maintenance. Most P2P botnets set up private networks. This is not a clever approach because the bootstrap nodes for the private networks may create well-known disruption points. Examples of such a design are Nugache [55] and Phatbot [56]. Nugache uses a fixed set of 22 bootstrap nodes. If the bootstrap nodes can be captured and destroyed then no new peers can join the network. Phatbot uses Gnutella cache servers for bootstrapping. Capturing the pre-determined cache servers can seriously disrupt the network. Finally, Sinit [60] is the only malicious network that uses a public DHT. However, Sinit does not form a botnet. Instead, it is used for distributing a Trojan horse. Sinit can be disrupted by introducing a large number of sybils in the target DHT.

4.3. Design

As we discussed earlier, the goal of Tsunami is a C&C channel without points of failure, and it accomplishes this goal by being a parasite of the Kad system and leveraging the built-in communication capabilities of Kad. We describe how Tsunami can achieve these capabilities in this section.

4.3.1. Tsunami Objective Refined

Every Tsunami bot embeds itself in Kad as a peer node, and does not even necessarily know any other Tsunami bots. Nevertheless, a Tsunami botmaster must be able to send commands to Tsunami bots, and perhaps also receive responses from

the bots. Of course, Tsunami would prefer good performance such as reaching its bots quickly. Albeit not required, Tsunami bots can also try to stay unnoticeable.

A Tsunami botmaster does not know which peers in Kad are Tsunami bots. In order to reach them, the botmaster can send a message to *every* peer in Kad. However, as Kad uses a 128-bit ID space to represent peers, doing so implies 2^{128} messages. This is not only resource intensive, but will also make the botmaster appear anomalous and get captured.

This problem can be resolved by realizing that a 128-bit ID space can never be fully populated. In fact, Kad uses a large ID space to avoid collisions between the IDs of different peers. We know from our earlier measurement study on Kad [61] that in practice, only the top 22 bits of a peer's ID are used to locate peers. In other words, as long as the top 22 bits are common between a lookup ID and a peer ID, the desired peer will be found. Thus, instead of sending 2^{128} messages, the refined objective of Tsunami is for the botmaster to send 2^{22} messages instead to broadcast its command. Note that 2^{22} is the total number of messages that needs to be sent; individual bots only send a small fraction of these messages, as explained in Section 4.3.3.

Our measurement study on Kad [61] shows that the number of bits required to locate peers is directly proportional to peer population; as peer population increases, more bits are required to locate peers. Thus, Tsunami must adjust the number of messages as the peer population changes, and also must cope with reduced number of command-carrying bits. While we do not provide a technical mechanism to address this problem, we note that only when peer population doubles, one bit needs to be added to the lookup bits (i.e., instead of 2^{22} messages, 2^{23} messages need to be sent). The botmaster may periodically crawl the Kad DHT, and as the peer population doubles, it may increase the number of messages.

4.3.2. Tsunami Command

The Tsunami botmaster issues commands using Kad *lookup* messages. In particular, the target ID in such Kad lookup messages is divided into the following two parts (Figure 4.1.):

- Lookup Bits: The top 22 bits are used to find Tsunami bots. As we discussed in Section 4.3.1., using the top 22 bits is sufficient for finding bots.
- Command Bits: The lower 106 bits are used to carry a Tsunami command. For example, it can encode multiple 32-bit IP addresses as the victims of a DDoS attack. The reader may observe that Kad may counter this communication mechanism by always using zeros for lower order 106 bits. However, recall from Section 2.2.1.1. that in Kad each file and keyword is uniquely identified by a 128 bit ID. Decreasing the number of bits by using zeros for lower order bits will lead to collisions.

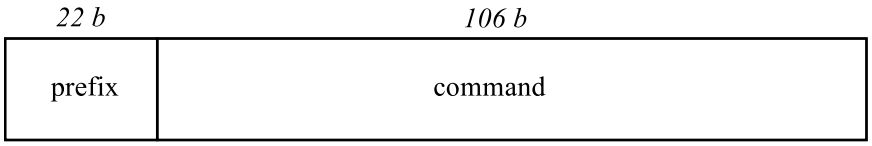


FIGURE 4.1. Tsunami command format.

Upon the receipt of a Kad lookup message that carries a Tsunami command, a benign peer will simply process it as a normal lookup message, while a Tsunami bot must be able to recognize it, obtain the command, and act upon the command. To meet this condition, the command bits can carry a command signature.

The botmaster can further encrypt the command bits, such as by using the private key of the botmaster, and a bot can decrypt the command and verify the

command is indeed from the botmaster. This will require every bot to have the decryption key, such as the public key of the botmaster, when distributing the binary code of the botnet.

4.3.3. Sending Commands to Tsunami Bots Embedded in Kad

We now describe how a Tsunami command can reach Tsunami bots embedded in Kad. In doing so, Tsunami wants to (1) reach as many bots as possible, (2) be fast in reaching them, and (3) stay relatively quiet in order not to appear suspicious to ISPs.

Tsunami uses a tree-based command propagation mechanism. First, the botmaster sends lookup messages toward a small number (e.g., 100) of target IDs. Every bot that happens to receive a lookup message will repeat the same process. If a bot discovers that a command is a duplicate (such as that caused by a loop), it will simply drop the command. Note that the tree is constructed on-the-fly, and does not require any coordination among the bots. Since the bots receive messages purely by chance, the larger the bot population the higher the reachability of the command (Section 4.5.).

As shown in Figure 4.2., this process will involve three different kinds of nodes:

- M represents the bot master.
- B represents a bot.
- K represent a regular Kad peer.

We emphasize that the botmaster of a Tsunami botnet can hide itself in two measures: First, the botmaster hides itself behind regular bots. If a defender captures a Tsunami bot binary and finds out the pattern of Tsunami commands, it could detect

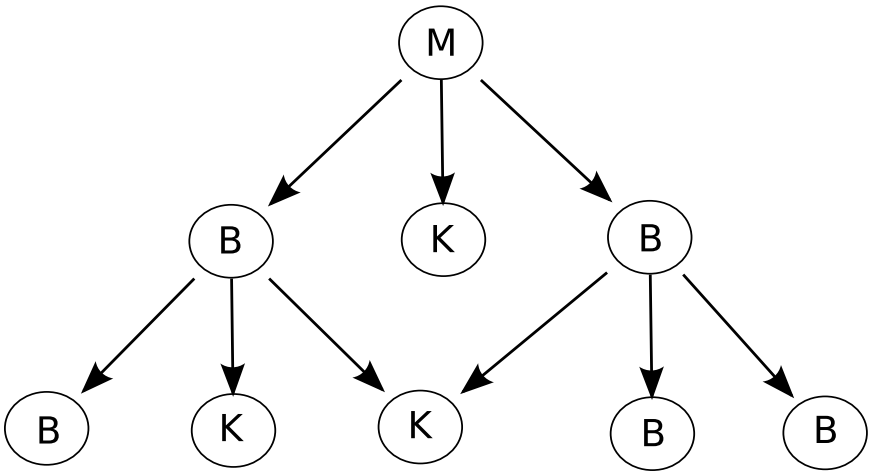


FIGURE 4.2. Sending commands to Tsunami bots embedded in Kad.

Tsunami commands and try to trace who sent the commands. However, there is no way to distinguish between a message sent by the botmaster and a message from one of thousands of bots. Furthermore, since lookup messages use UDP and their source IP address can be spoofed without affecting message delivery, the botmaster (and a regular bot as well) can fake its IP address at the last hop of the lookup message.

We introduce two forms of bot-bot communication:

– Whispering: For a bot, lookup messages simply provide a way to find other peers in the network. A bot hopes (without knowing) that some of the discovered peers are bots. As a result, the bot does not have to carry the command bits in its lookup message. It can simply find a peer closest to any random ID. Then it can send the actual command bits to the discovered peer using some other Kad protocol message (e.g., publish or search). This approach saves the bot from exposing itself. During the lookup, at intermediate hops, a bot cannot fake its IP address because it expects replies to progress in routing. As a result, if a lookup message is intercepted in the middle and if the lookup message is carrying command bits, the bot itself can be exposed. This cautious approach,

however, may reduce the efficiency of the botnet because after every lookup only one peer is sent the command bits.

– Shouting: Shouting is opposite of whispering. In this approach, the lookup messages issued by the bots carry the command bits. As a result, any bot along the way will learn about the command. This efficiency comes at the possible cost of exposure, as described above.

4.3.4. Receiving Response

A botmaster may wish to receive responses for certain commands, such as the machine information of bots and the success rate of spam. Since Tsunami bots do not know the identity of the botmaster, they cannot directly send responses to the botmaster.

To solve this problem, Tsunami bots use *logical overlay links* that are created during command dissemination. At each step of forwarding a Tsunami command, a bot A will discover at least one bot B: when B forwards a command to A, A immediately learns that B is a bot and is a parent of A. This information creates a logical uni-directional link from A to B. Bot A sends its response to bot B, which collects responses from all its children and propagates aggregated response to its parents. Same as Tsunami command, the response messages are also delivered via Kad protocol messages.

A bot's parent(s) may depart, disconnecting it from the botmaster. However, the botmaster can recreate the overlay by periodically rebroadcasting a command. Each command may carry a nonce to distinguish command re-broadcasts from original command broadcasts. Bots can respond at the first command-rebroadcast after the fulfillment of the original command.

Command responses can also help the botmaster to estimate the bot population. (As described in Section 4.5., the botmaster can then determine the number of copies that every bot needs to forward for a command.) Bots communicate this information by embedding it in the aggregated responses, where every aggregated response carries the number of responses merged. Note because the redundancy in the Kad network may artificially inflate the bot population, this approach only provides an *estimate* of bot population.

4.4. Tsunami Attack Examples

In this section we describe how Tsunami issues distributed denial of service (DDoS) and spam commands.

4.4.1. Distributed Denial of Service (DDoS)

A DDoS command can be delivered using either short commands or long commands.

Short Commands: A short DDoS command consists of two parts: a 10-bit unique command ID, and at least one target IP address. The command ID helps the botmaster identify the response to a command, as described in Section 4.3.4.. The botmaster requires the command ID to be unique across unanswered commands only. Thus, the botmaster can issue 1024 ($2^{10} = 1024$) distinct commands in parallel.

Tsunami uses lookup messages to deliver short DDoS commands. It embeds the command in the 106-bit suffix of the lookup ID, as described in Section 4.3.2.. It can also embed a Tsunami signature (e.g., a particular bit string) for bots to recognize it is a Tsunami command.

Long Commands: A long DDoS command consists of four components: a 10-bit unique command ID, IP address(es) or hostname(s) of the target, date and time of the attack, and bot population. As the Kad lookup message does not provide enough space to accommodate all components, Tsunami uses Kad's storage/retrieval-phase messages for these commands. It delivers long commands in two steps. First, it uses lookup messages to find peers for storing commands. Next, it sends PUBLISH messages (i.e., either PuK or PuF) to the discovered peers, with commands embedded in the value portion. Tsunami prefers PUBLISH messages over SEARCH messages, because SEARCH messages do not have a value portion, which limits the space for a command.

Tsunami uses different keys for a given pair of LOOKUP and PUBLISH messages. The keys share the prefix, but they differ in the suffix. The suffix of the lookup key is completely random because it does not carry a command. On the other hand, the suffix of the PUBLISH key must carry the command.

4.4.2. Spam

Because of the *bulky* nature of spam, issuing spam commands is more challenging than issuing DDoS commands. Modern spam campaigns [62] typically consist of list of email addresses, email template and a collection of values for macros in the text, called a dictionary. Tsunami solves this problem by storing the spam information as key-value pairs in Kad, and broadcasting only the keys to the bots.

The botmaster uses one PuK message for each component of spam information. First, the botmaster randomly chooses three 128-bit IDs K_1, K_2, and K_3 as the keyword hashes for three PuK messages M_1, M_2, and M_3, respectively. The botmaster uses the body of each PuK message to encode spam information. Next, it goes through

the lookup phase to find the closest possible peers to K_1, K_2, and K_3, as described in Section 2.2.1.1.. After that, it sends M_1, M_2 and M_3 to the discovered closest possible peers. Finally, the botmaster broadcasts K_1, K_2 and K_3 using the same method as DDoS.

As bots receive K_1, K_2 and K_3, they retrieve each component of spam information. First, each bot goes through the lookup phase to find the closest possible peers to K_1, K_2 and K_3. Next, each bot sends SeK messages to discovered peers and retrieves spam information. Finally, they construct spam email messages using email template and email dictionary, and start sending spam (e.g., every bot can randomly choose a pre-determined number of email addresses for sending spam).

4.5. Evaluation

We have developed a discrete event simulator to evaluate the performance of C&C in Tsunami when it runs on top of Kad. We now present the results for these evaluations.

4.5.1. Issuing Commands

We use "time taken to reach maximum possible bots" as the main performance metric, which is affected by the following two factors:

- Bot population
- Number of lookup messages issued by each bot

During our simulations, we vary these two factors to understand their impact on command dissemination time. We use a real world churn model [63] to simulate the arrival and departure of the bots.

We keep peer population constant at 4,000,000 [61] and lookup latency constant at 8 seconds [64]. Note that lookup latency takes into account the network latency as well. We vary bot population and number of messages issued by each bot. Bots are placed randomly in Kad, as they would be in a real-world scenario. During our simulations each bot picks up a random ID and simulates a lookup. We check if the lookup ID is registered as a bot. If it is then it is considered as a successful lookup. Upon receiving a command, the bot simulates more lookups and the process continues until there are no more pending messages in the botnet.

The results of our simulation are shown in Figure 4.3. and Figure 4.4.. Figure 4.3. shows the impact of varying the percentage of bot population on command distribution time. The x-axis shows bot population as the percentage of total population. The y-axis shows command distribution time in seconds for reaching 75% of bots. Each line in the figure represents different number of messages sent by a single bot. If a given line does not show a value for a given $j\%$ of bots, it means that with $j\%$ bot population and i messages, 75% of bots are not reachable. Figure 4.3. demonstrates that increasing the bot population decreases command distribution time. This observation follows the intuition that a larger bot population will distribute commands more efficiently. Figure 4.3. also shows that increasing the number of messages sent by each bot increases the command distribution time. For example, for a bot population of 7% or more, sending 50 messages from each bot is the optimal operating point. Sending 60 or 70 messages only increases command distribution time without much gain. Sending more messages from each bot increases the likelihood of discovering already contacted bots, thus wasting the message. Figure 4.3. also indicates that for 6% bot population, at least 80 messages must be sent from each

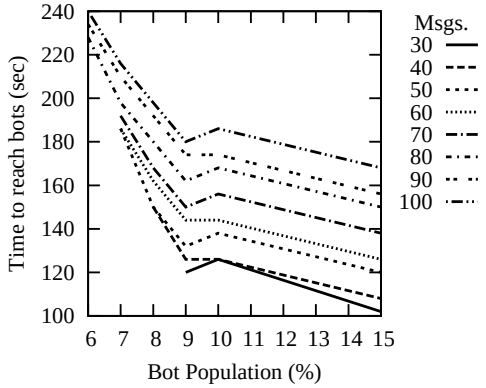


FIGURE 4.3. Impact of Bot Population on Command Distribution Time

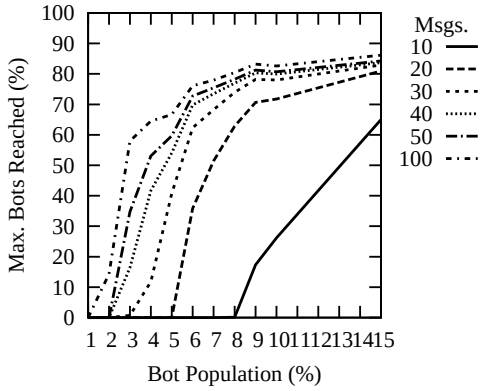


FIGURE 4.4. Maximum Number of Bots Reached

bot to reach 75% of bots. Figure 4.3. also shows if the bot population is less than 6% of total population, then 75% of bots cannot be reached.

Figure 4.4. shows maximum possible bots that can be reached under the current churn model, while varying the bot population and number of messages sent by each bot. The x-axis shows bot population as the percentage of total population. The y-axis shows the percentage of bots that can be reached. Each line in the figure represents different number of messages sent by each bot. Figure 4.4. shows increasing the bot population increases the likelihood of reaching more bots. Figure 4.4. also shows that increasing the number of messages sent from each bot increases command

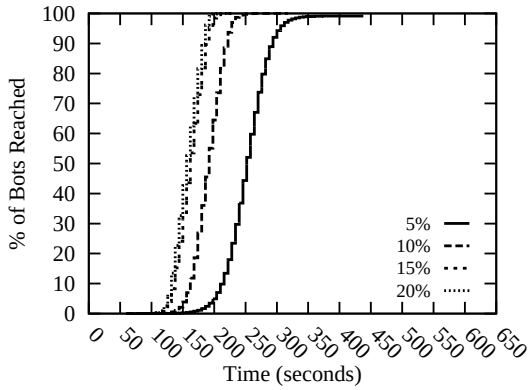


FIGURE 4.5. Simulation results without churn.

reachability but to a certain point only. For example, if the bot population is 7% or more of the total population, then sending more than 50 messages from each bot does not increase command reachability. This is because of the message overlap phenomenon described earlier. Finally, we note from figure 4.4. that under the current churn model, slightly more than 80% of bots can be reached at best. Thus, if maximum bots need to be reached with minimum number of messages, then at least 9% of total population must be bots and at least 50 messages must be sent from each bot.

In order to get an upper bound on bot performance we also conducted experiments with a constant bot population. The results of this experiment are shown in Figure 4.5.. This experiment was conducted with 100 messages per bot. We only varied the bot population. Figure 4.5. shows that as long as 5% bots are present in the system, all the bots can be easily reached. As bot population increases, the time to reach all the bots decreases.

4.5.2. Performance of Response Reachability

Figure 4.6. shows how many bots are able to respond to a Tsunami botmaster. The x-axis shows bot population as percentage of total population. The y-axis shows those bots that receive a command and have a response-route to the botmaster. Each line in the figure shows the number of messages sent by each bot. Figure 4.6. shows that for 30 or more messages, 99% of the bots have a response route to the botmaster, and can successfully send the response. In other words, the bot population has no impact on response route when each bot sends 30 or more messages. Figure 4.6. also shows if each bot sends less than 30 messages, then response routes only exist when the bot population is greater than or equal to 6% of total population. If the bot population is less than 6% and each bot sends less than 30 messages, then under the given churn model, no route exists from bots to the botmaster.

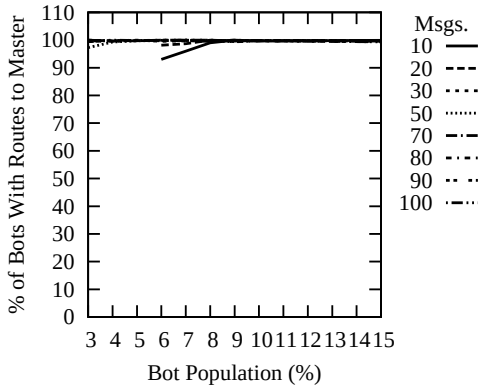


FIGURE 4.6. Successful routes from bots to botmaster.

4.6. Implementation of Tsunami

In addition to evaluating the performance of Tsunami via simulation (Section 4.5.), we demonstrate its feasibility via emulation over Kad.

We opted for emulation instead of a real-world implementation, because an implementation would disrupt the normal operation of Kad. An implementation would require deployment of modified Kad peers that can identify and propagate malicious commands. Such peers would artificially inflate the peer population, and their sudden departure at the end of the experiment would result in stale routing table entries.

In order to avoid above-mentioned problem, we developed an emulator that uses Planetlab [65] nodes as *bot emulators*, which inject malicious commands in the real-world DHT. Specifically, the emulator chooses random Kad peers as *artificial bots*, and assigns one bot emulator to each artificial bot. The botmaster injects a malicious command at a random point in the Kad DHT. As artificial bots receive malicious command traffic, bot emulators propagate commands in the DHT on their behalf. The use of Planetlab nodes helps us in avoiding the artificial inflation of Kad peer population, while still propagating malicious commands in the real-world Kad DHT.

The specific emulation steps are shown in Figure 4.7., and described below:

1. Crawler crawls a *zone* of the Kad DHT once every minute, and discovers all the peers in that zone.

   We define a zone as a collection of Kad IDs that share the same x bit prefix. For example, a 12 bit zone, 0xacf, contains all the lookup IDs with 0xacf as prefix. For the emulation, we crawl a 12 bit zone instead of the entire DHT, in order to minimize the resource usage.

   The crawler discovers all the peers in a given zone by issuing lookups for all the Kad IDs in that zone, and finding all the closest peers. Finally, it chooses $n\%$ of discovered peers at random as artificial bots. Similar to simulations, we vary n from 1 to 15.

2. The botmaster injects a command in the Tsunami botnet. In a real-world implementation the botmaster would simply issue lookups, as described in Section 4.3.. However, for emulation, the botmaster requests the *controller* to issue lookups on its behalf. As the name indicates, the controller is the central component of the emulator that coordinates its activity.

3. The controller requests identities of currently chosen bots from the crawler.

4. The crawler sends the Kad ID, IP address and port number of currently chosen bots to the controller.

5. The controller chooses a random Planetlab node to perform lookups on behalf of the botmaster.

6. The chosen Planetlab node performs lookups in the crawled 12 bit zone, and sends the Kad ID, IP address and port number of all discovered peers to the controller.

7. The controller compares the discovered peers from the previous step against the bot identities received from the crawler. If a match is found, the controller assumes that a bot received the command and it will propagate the command further. For each match, the controller picks a random Planetlab node to perform lookups in the crawled 12 bit zone. This process continues until all lookups are complete on all Planetlab nodes and no more unique bots are found.

4.7. Defense Against Tsunami Botnet

In general, defense against a botnet may use the following approaches: (i) patching the software or hardware vulnerabilities so that the botnet cannot recruit

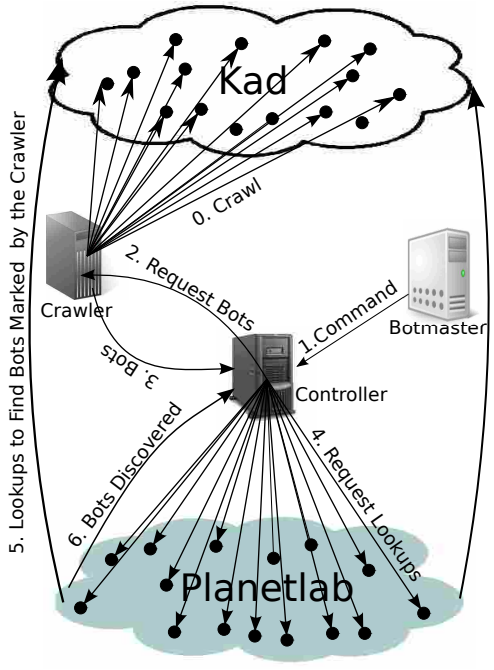


FIGURE 4.7. Tsunami emulator design.

new bots; (ii) protecting probable targets [66] from a botnet attack; and (iii) disrupting the C&C of the botnet once it becomes active. In this chapter, we focus on disrupting the C&C. Since unlike today's botnets, Tsunami does not employ bottleneck nodes for us to locate and shut down, we investigate how we may intercept C&C traffic of Tsunami in Kad.

In this section, we first briefly describe our generic technique for capturing DHT traffic. We then use this technique for defending Kad against Tsunami. After that, we evaluate the effectiveness of defense. Finally, we describe the weaknesses in the defense approach.

4.7.1. Capturing DHT Traffic

In order to capture DHT traffic, we extend *Montra*, described in Section 3.3.. Montra introduces a large number of passive peers—called *monitors*—to capture Kad

traffic. The monitors are passive because they do not issue any queries. A single monitor can capture the traffic received by a single peer. We modify these monitors to capture the traffic destined towards a single lookup ID.

Monitoring a lookup ID is simpler than monitoring a peer, as described in Section 3.3.1.. We simply set the ID of the monitor M same as the lookup ID T in question, thus making M the closest peer T. As a result, M receives all the traffic destined for T.

4.7.2. Protecting Kad Against Tsunami

In order to intercept Kad traffic that carry Tsunami commands, we must first *capture* it and then *redirect* it, as described in the next two subsections.

Capturing Malicious Traffic:

We capture malicious traffic by monitoring all command-carrying lookup IDs (i.e., use ID monitors), as described in Section 4.7.1.. In order to deploy ID monitors, we must find the command-carrying IDs in advance. From the analysis of the bot binary, we know that Tsunami bots issue lookups for 2^{22} lookup IDs, varying high-order 22 bits. Thus, we know all the 2^{22} prefixes. However, we do not know the 106 bit command suffix that the botmaster is currently using to issue the command. Without the command suffix, we cannot construct the lookup IDs and cannot deploy the monitors.

We find the command suffix by monitoring a large number of individual peers (i.e., use peer monitors), as described in Section 4.7.1.. Since bots send commands to random peers, the larger the number of monitored peers, the higher the probability of intercepting malicious traffic. As described in Section 3.4., we can use one desktop machine to monitor 32,000 Kad peers with 90% accuracy. Thus the percentage of

Tsunami's 2^{22} lookups that our monitors can receive is $\frac{Number of Monitors=32{,}000}{Total Population=2{,}000{,}000} = 0.016$, or 1.6%. In other words, we can receive $67,109$ lookup messages on average—more than enough to find the command-carrying suffix (we only need one lookup for this purpose). The monitors decrypt all the intercepted traffic, using Tsunami's public key. By examining the decrypted traffic for command signature, the monitors finally discover the command suffix.

Upon finding command suffix, we construct 2^{22} lookup IDs by combining 2^{22} prefixes and one command suffix, and deploy 2^{22} ID monitors. It may seem resource-intensive to deploy such a large number of monitors. However, [61] demonstrates that we can deploy 2^{22} monitors using 64 desktop machines. Compared to the recent botnet defense proposals [66], which have advocated the use of a multi-million-node, non-malicious botnet to defend against a malicious botnet, 64 machines are negligible.

Redirecting Malicious Traffic:

We redirect malicious traffic by sending it towards a *traffic sink*. The traffic sink consists of large number of modified Kad nodes, distributed across the world in order to prevent the identification of traffic sink nodes by IP prefix. These nodes pretend to be the closest peers to all the incoming traffic, preventing the traffic from going to any other peer.

The task of traffic redirection is best-suited for traffic monitors that capture malicious traffic. These monitors can mislead the traffic-sending bots and force them to send the traffic to the traffic sink. Specifically, they use false IDs for traffic sink nodes, such that those IDs appear to be closest to the lookup ID in the intercepted traffic. (The traffic monitors construct false IDs by using the same 30-bit prefix as the captured lookup ID, and choose the remaining 98 bits randomly. Usually a closest peer and a lookup ID share a 22-bit prefix [67]; using 30 bits is simply to be more

safe.) As a result, the bots regard traffic sink nodes as the closest and stop sending further traffic.

4.7.3. Evaluation of Tsunami Defense

In this section we evaluate the effectiveness of our defense mechanism. Our evaluation metric is the percentage of requests that we successfully intercept and redirect. For evaluation, we emulate defense and C&C over the real Kad network, as described below.

Emulating Defense:

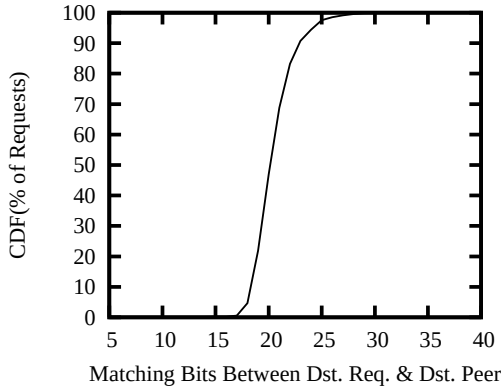
We emulate the defense by deploying ID monitors in an 8 bit zone of the Kad ID space. We deploy ID monitors by generating artificial command-carrying IDs. An 8-bit zone contains $\frac{1}{2^8} * 2^{22} = 2^{14}$ command-carrying lookup IDs. Thus, we generate 2^{14} ID prefixes. We assume that we know the 106-bit command-carrying suffix without monitoring a large number of Kad peers (Montra can capture destination traffic with 90% accuracy). By using 2^{14} prefixes and *one* constant 106-bit suffix, we generate 2^{14} lookup IDs and deploy 2^{14} ID monitors.

Emulating C&C:

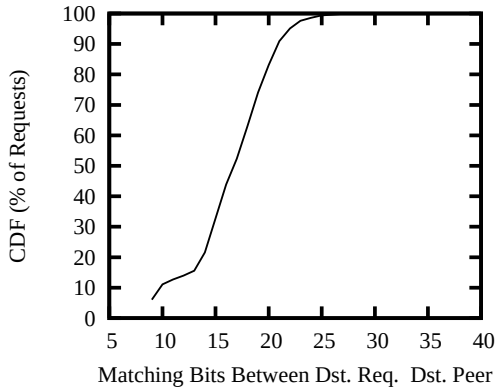
We emulate C&C over the real Kad network by issuing 2,000 command-carrying lookups towards the 8-bit monitored zone. We construct command-carrying lookup IDs as follows:

- We set the first 8 bits same as the prefix of the monitored zone, so that the lookups can enter that zone.
- We set the next 14 bits randomly, because the bots issue commands randomly.

– Finally, we use the same 106-bit command carrying suffix that we use for deploying ID monitors.



(a) Number of bits matching between requested ID and destination node for Regular Traffic



(b) Number of bits matching between requested ID and destination node for Traffic After Adding Montra Monitors

FIGURE 4.8. Evaluations for defense against Tsunami.

Results: The results of our evaluation are shown in Figure 4.8.. Figure 4.8.a shows the distribution of the number of common bits between requested ID and destination node under normal conditions. We borrowed this result from [67]. The two lines, min and max, show the variability of results obtained from different zones of the ID space; the results do not vary across different zones. Figure 4.8.a shows that for 20%

of requests, the closest possible node and the lookup ID has 19 bits in common; for the next 20% of the requests, the closest possible node and the lookup ID has 20 bits in common; and the remaining 60% of requests have 21 or more bits in common with the destination node.

Figure 4.8.b shows the distribution of the number of common bits between requested ID and the last closest node that is contacted before the Montra monitors intercept the request and mislead the request-issuing bot. The figure shows that 70% of the requests are captured at the 19^{th} bit. From this we can derive that 50% of requests are captured before they reach their destination, since previously only 20% of requests had destination at the 19^{th} bit. In addition only 10% of the requests are captured at 21 or more bits. Under normal conditions, 60% of the requests found their destination at 21 or more bits.

4.7.4. Limitations of Proposed Defense

Although our proposed defense intercepts and re-directs 50% of traffic, it has the following weaknesses:

- We must know the command signature a priori. Tsunami may further use polymorphic commands [68] to hide its traffic.
- We capture traffic before it reaches destination, but bots at intermediate hops may still receive commands.
- The proposed defense only works for short commands that are carried by first-phase lookup messages. We cannot capture long commands that are carried by second-phase storage/retrieval messages.

4.8. Recap

In this chapter we presented Tsunami, a parasitic botnet over Kad, in order to demonstrate the lethality of lack of membership management in P2P systems. If a strict membership management system as in place, bots could be held accountable for their actions, and their service from the network could be discontinued. Tsunami establishes a parasitic relationship with Kad, embeds its bots in Kad, and uses its communication channels for C&C traffic. Tsunami can disseminate a command to 75% of bots when the total bot population is only 6% of total Kad peer population.

CHAPTER V

COMPASS: A CONTENT DISCOVERY SERVICE ON THE INTERNET

5.1. Overview

Text from this chapter is based on work submitted to Infocom 2014. My advisor Prof. Jun Li, Dr. Matteo Varvello and I were the primary contributors to this work.

Although originally designed for connecting machines, the Internet has become the primary medium for producing and consuming content. Discovering the location of content on the Internet, however, is still challenging. The scale of content is only becoming larger as new content is continuously added. The location of content is often not at a fixed location as content is becoming more and more mobile. Yet, a client must still be able to locate content quickly and accurately, and should be directed to the nearest copy of content as content is also often replicated at multiple locations.

One key property from which an effective content discovery system can benefit is the support for *persistent* content names. Persistent content names are only bound to the content itself, and are independent of its location or ownership. With a persistent name, even if content moves from one owner to another (e.g., different organizations) or is replicated at a different location, it can always be referred by the same name, thus naturally supporting content mobility and content replication. Although one may consider using the Domain Name System (DNS) [16] for content discovery, DNS does not support persistent content names. If content is specified using a Unique Resource Identifier (URI) [53], DNS can map a content's domain name (which is

derived from its URI) into an IP address, but whenever content moves from one DNS domain to another, its name must be changed. Furthermore DNS is not location-aware. Name resolution with DNS does not consider the IP address of a requesting client, thus not guaranteeing that the client will be directed to the closest copy of a content.

Another salient property to have is the separation of content discovery and delivery, to enable content recipients (or their ISPs) to have more control over the quality of content discovery, such as choosing from whom and how their content should be delivered. Use content distribution networks (CDNs) as a counter example, where every CDN consists of a collection of geographically distributed content servers with replicated content. Although CDNs alleviate the location-awareness problem of DNS (typically by using a collection of DNS servers and measurement tools), it does not support persistent names either. Worse, every CDN tightly couples content discovery and delivery: Not only does it deliver content to end-hosts, it also decides which content server an end-host should contact to retrieve content needed, even if that may not be a good choice for the end-host or its ISP. Instead, if content discovery were separated from content delivery, users would be better served since they would be able to choose which CDN or which CDN server is the most preferred when content is replicated at multiple locations, and individual CDNs could also be relieved from discovering content.

Recently, the networking research community has started a new initiative called Information-Centric Networking (ICN) [69–71]. It uses content names as end-points of communication, instead of IP addresses, and clients directly ask for content using content names, with no need of IP addresses and DNS resolution. ICN, however, is a clean-slate design for the future Internet, and requires a complete revamp of today's

network infrastructure. Keeping this in view, another property that content discovery systems must possess is support for incremental deployment.

Peer-to-peer (P2P) networking technologies, especially their distributed hash table (DHT) design, appear to support all aforementioned properties - i.e., persistent content names, separation of content discovery from content delivery, and incremental deployment. Recent proposals (e.g., P4P [72] and DHT-based localization [73]) have also shown that P2P networks can integrate new mechanisms for location awareness. However, P2P networks have their own weaknesses, as described in Section 3.6., which hamper their use as an Internet-scale content discovery service.

In this chapter, we design and evaluate a P2P-based content discovery service for today's Internet, called *Compass*. Since Compass is based on P2P networks, it naturally supports persistent content names, separates content discovery from content delivery, is scalable and is incrementally deployable. Furthermore, Compass introduces new mechanisms to address the weaknesses of P2P systems (Section 3.6.). In Compass, content is uniquely identified by *content-ID*, a cryptographic hash of content name. Content names, and thus content-IDs, are persistent. Compass consists of content discovery agents (CDAs), which resolve content identifiers into their locations, i.e., IP addresses. CDAs are application level servers, similar to DNS resolvers, which do not require changes to the underlying Internet infrastructure and are incrementally deployable.

CDAs control their participation in Compass by announcing the set of content-ID prefixes for which they are willing to store resolution information. The length of the prefix determines the number of supported content IDs: the longer the prefix, the less the burden on their infrastructure. Prefix announcements propagate through the Internet using a BGP-like protocol. CDAs can also wthdraw an already-announced

prefix if the resource requirements become too high. This flexibility of Compass addresses the heterogenity weakness of P2P networks: each CDA announces prefixes according to their own storage and bandwidth capacity, instead of those requirements being dictated by the system. Furthermore, this property also makes Compass scalable.

Compass leverages replication of content ID prefixes in order to both ensure resilience and lower the resolution latency (i.e., stretch). Neighboring CDAs can partner with each other to resolve the entire content ID space, ensuring minimal resolution latency to their collective customers. We refer to such partnerships as *realms*. A CDA can join one or multiple realms, provided that it reaches an agreement with one of the realm's current members to be responsible for a content ID prefix. Realms address the stretch weakness of P2P systems.

Compass supports *publish* and *search* operations. In the publish operation, a content provider uses a CDA to inform all the relevant CDAs about the mapping from content ID to IP address. We have designed the publish operation such that it assumes transient failures of Compass nodes instead of churn. In the search operation, an end host uses a CDA to find a closest possible CDA that may store the mapping from a given content ID to an IP address.

Finally Compass addresses the membership management weakness of P2P systems by using public key cryptography to establish strong identities for CDAs, and by creating chains of trust. Specifically, a new CDA can only join Compass if an existing CDA trusts the new CDA and they establish a business relationship with each other.

We evaluate Compass via our simulation framework based on real Internet topologies. We show that Compass can conduct its two primary operations efficiently:

a random CDA can resolve a content name into an IP address in 5 or less hops, which in ballpark numbers is 55 ms, and significantly better than 382 ms seen by DNS resolution on average. Compass can publish the mapping between a content name and an IP address in 15 or less hops, which translates in approximately 10 seconds for publishing to the whole Internet. This is significantly lower than what CDNs expect today (100 seconds on average). The storage cost of Compass is also manageable. For example, a tier-1 CDA may need to store $48,500$ entries in its forwarding table, which is negligible.

The remainder of the chapter is organized as follows: Section 5.2. presents an extensive literature review of relevant content discovery solutions. Section 5.3. designs Compass, and Section 5.4. addresses security challenges faced by Compass.

5.2. Related Work

5.2.1. Content Distribution Networks

Content distribution networks unify the content dissemination and content discovery processes: content is discovered via a CDN's own content discovery mechanism, and content is disseminated via CDN's own dissemination mechanism. CDNs were primarily designed to decrease the latency of content delivery, in order to satisfy end hosts. In this chapter we review the various mechanisms that CDNs employ to distribute content, and then highlight their limitations with focus on content discovery.

Several CDNs exist today, and are used for downloading content (e.g., web pages, large software updates, multimedia files), or streaming content (e.g., on-demand movies, live television). Some of the CDNs use P2P technology to provide service.

As of 2012, Akamai [74] owns 48% of market share [75]., Level 3 [76] 25%, and Limelight [77] 18%. In this section, we use Akamai and Limelight as examples, because they are two of the top three CDNs, and are most widely studied.

CDNs consist of large number of geographically distributed machines that replicate content, called *content servers*. CDNs also run large number of DNS servers, either on the same machines as content servers or separately. This infrastructure is divided in to 3 logical components, as follows:

- Content Discovery: This component is responsible for discovering the best possible content server. It consists of CDN's DNS servers.
- Content Dissemination: This component is responsible for strategically replicating content across the world. It consists of content servers.
- Network Monitoring: This component is responsible for monitoring Internet paths and content servers to rank the content servers based on their availability and congestion along the paths.

We describe each of the above-mentioned components in the following sections.

5.2.1.1. Content Discovery

As mentioned above, instead of designing a new content discovery mechanism, CDNs rely on DNS for content discovery in order to maintain backward compatibility. CDNs cannot rely on the authoritative servers of content providers to provide the IP address(es) of a CDN server, because the IP address of a CDN server depends on the IP address of the DNS resolver that sent the content discovery request. Furthermore, even if the same DNS resolver repeatedly sends the same DNS query, the IP address of the content provider would vary, because CDNs continuously monitor the path and

load on their content servers, in order to choose the *best* possible content server at the moment. For these reasons, each CDN deploys its own content discovery component, which consists of a large network of DNS servers.

CDNs require a *hook*, through which a DNS query can enter their network of DNS servers. This hook is provided by the authoritative DNS server of the original content provider. When a DNS resolver contacts the authoritative server for a domain to retrieve its IP address on behalf of a content consumer, the authoritative server, instead of responding with the IP address for the domain name, responds with a canonical name (CNAME) for a CDN. For example, if a content resolver queries *www.cnn.com*, the authoritative server for *www.cnn.com* may respond with the canonical name *a11105.b.akamai.net* (Akamai), or *dns11.llnwd.net* (Limelight), depending on CNN's CDN.

The above-mentioned *hook*, allows a DNS query to enter a CDN's network of DNS servers. Upon receiving the canonical name for a domain from its authoritative server, a DNS resolver attempts to resolve the canonical name, as stated by the DNS protocol [16]. Since the canonical name belongs to a CDN (e.g., **.akami.net*, **.llnwd.net*), the DNS resolver contacts CDN's authoritative server. As the DNS query enters CDN's network of DNS servers, the CDN determines the current *best* server for a given DNS resolver, and in turn the content consumer.

CDNs organize their DNS servers into different topologies. Akamai uses a two-layer architecture [78], whereas Limelight uses a flat one-layer architecture [79]. With regard to Akamai, a DNS resolver first reaches the top layer of Akamai's DNS servers by resolving the CNAME (e.g., a11105.b.akamai.net), received from the authoritative name server of a domain. Specifically, when the content resolver sends a DNS query to the authoritative name server of .net to resolve .akamai.net, it receives the IP address

of a top layer Akamai nameserver. The top layer Akamai server resolves a query from a DNS resolver for .b.akamai.net by considering the IP address of the resolver, and responding with an IP address for .b.akamai.net that is closest to the DNS resolver. This IP address for .b.akamai.net belongs to a second layer DNS server. Finally, the second layer DNS resolver returns the IP address of a content server that is the current *best* for the DNS resolver. The above-mentioned process shows that Akamai uses a two-layer architecture for its DNS servers so that it can direct a DNS resolver to its closest possible DNS server. The IP addresses discovered through Akamai's two-level DNS infrastructure are cached for approximately 30 seconds [80], so that the *best* content servers can be repeatedly discovered. Unlike Akamai, however, Limelight uses a flat one-layer architecture, because it does not use its DNS servers to determine the closest possible DNS server for a DNS resolver (described next).

CDNs use different addressing schemes for their DNS servers. Akamai uses unicast addressing, whereas Limelight uses anycast addressing. IP anycast [81] allows the same address to be assigned to different hosts. When forwarding packets, intermediate routers forward packets to the closest possible host. Every DNS server in Akamai has its own IP address, whereas DNS servers in Limelight can have the same IP address. By using IP anycast, Limelight is relieved from the responsibility of finding the closest possible DNS server for a given DNS resolver. This approach saves Limelight from the overhead of discovering and maintaining the closest possible DNS servers for DNS resolvers, but Limelight loses fine-grained control over the choice of DNS servers for DNS resolvers. For example, at a certain point in time, some DNS servers and the associated content servers could be overloaded or on a congested path. Limelight cannot redirect content resolvers away from such DNS servers and content servers. Akamai's use of unicast addressing, on the other hand, gives it fine-grained

control over its selection of DNS servers, at the cost of discovering and maintaining the closest possible servers for a given DNS resolver.

5.2.1.2. Content Dissemination

Content dissemination is the process of strategically placing content replicas around the world, so that content can reach maximum end hosts with minimum possible replicas. There are two prevailing approaches in this regard:

Deep into ISP:

In this approach, a CDN deploys content servers inside an ISP's Point of Presence (PoP). The key idea is to get as close to the end hosts as possible, in order to decrease the content retrieval latency. While this approach decreases latency for end hosts, it requires that CDNs develop sophisticated techniques to manage a globally distributed system of content servers. Furthermore, content between these servers is transported via the public Internet, which requires sophisticated algorithms. The deep into ISP approach is utilized by Akamai. Akamai owns $60,000$ servers in $1,400$ data-centers on 900 networks across the world [82]. Geographically, Akamai covers 650 cities in 76 countries.

Bring ISP Home:

In this approach, a CDN builds small number of large content distribution centers close to PoPs of multiple ISPs[1]. Furthermore, content between content distribution centers is transported via private high-speed connections. This architecture is easier to manage than "deep into ISP" approach, at the expense of slightly higher end host latency. Limelight uses this approach for content distribution. Limelight controls 18 content distribution centers across the world [79].

[1]PoPs of multiple ISPs may be co-located in the same building [83]

5.2.1.3. Network Monitoring

Few details of CDNs' monitoring infrastructures are publicly available. However from the little existing public information [84], it is clear that CDNs use servers from their discovery and dissemination infrastructure to probe each other, in order to discover uncongested network paths and lightly-loaded content servers. Using the information from probing, CDNs construct a map of the Internet, which is used when responding to DNS queries from DNS resolvers. Specifically, depending on the IP address of the DNS resolver, CDN's DNS server maps the content consumer to a content server that is is most available and lies on an uncongested network path.

5.2.1.4. Weaknesses

In this section we discuss the weaknesses of CDNs, which make them unsuitable for content discovery.

The first and foremost problem of CDNs is the lack of support for persistent names. Different CDNs may host the same content, but it cannot be discovered because the content names vary with CDNs. For example, Netflix [85] and Hulu [86] both employ 3 CDNs [87, 88], in order to ensure availability. However, both Netflix and Hulu have devised sophisticated architectures so that end hosts can switch between CDNs. When a video player (i.e., client) on an end host connects with Netflix or Hulu server to stream a video, it receives a manifest file that lists the CDNs that can be contacted along with other information. The client retrieves video from a single CDN at a given time, and continuously measures the observed performance. When a CDN's performance drops below 100 Kbps, the client switches to another CDN, as mentioned in the manifest file. If CDNs supported persistent content names, then Netflix and Hulu would only need to provide a content name. The end hosts could

then resolve content names, and download from *any* content server that possesses the content.

Furthermore, at present, no single CDN covers the entire world well. For example, 61% of Akamai servers, and 68% of Limelight servers are located in the United States [79]. This unevenness in coverage leads to several different issues. First, in countries where a CDN's presence is sparse, the end hosts will not be able to retrieve content with low latency [80]. Unfortunately, the decision to use a particular CDN is made by the content provider, and the end hosts have to suffer the consequences. Furthermore, CDNs use their presence in different ISPs around the world to perform network monitoring, as discussed above. The uneven coverage implies that CDNs can obtain rich measurements in the US, but outside the US, the measurements are fairly sparse. Finally, unless CDNs have a presence in every AS around the world, they cannot determine last-mile performance. Because of the above-mentioned reasons, CDNs rarely select the best possible content server [89].

Finally, CDNs dictate to ISPs the content server that its end hosts must use. However, the choice made by a CDN for an ISP maybe costly, if the ISP has to repeatedly send the traffic through its provider. The recent explosion in video traffic [90] is only going to get worse [91], thus making it more challenging for ISPs to co-exist with CDNs. Unlike CDNs, peer-to-peer networks do not dictate which peer to use for downloading content. Instead P2P networks give end hosts (and thus ISPs) choice among multiple hosts. Because of these options, a cooperation between ISPs and P2P networks is beginning to emerge [72, 92].

5.2.2. Information Centric Networks

Information-centric networking is a new networking paradigm that proposes a natural solution to content discovery: *content is requested by its name instead of by its host*. This novel networking paradigm requires a groundbreaking change in the way content is delivered. Content is retrieved via *name-based routing*, where the final destination of packet is "content name" instead of "content host". Information-centric networking offers multiple benefits: (1) the network provides full DNS functionality, in favor of fault tolerance, speed and scalability, (2) content can be naturally cached along the path providing better network utilization, (3) content itself can be secured, instead of the paths where it flows.

In this section, we discuss each ICN architecture in detail and highlight its limitations.

5.2.2.1. TRIAD

TRIAD [93–95] was the very first ICN design, introduced in 2001. TRIAD uses DNS-style hierarchical URLs to discover content. It is built on top of the IP network, which means it resolves content names into IP addresses. TRIAD suffers from the lack of persistent names, and lack of scalability. In the following subsections, we first describe the architectural elements of TRIAD and then its weaknesses.

Architectural Elements:

- ***Naming:*** TRIAD replaces IP addresses with DNS style domain names - i.e., end hosts have DNS style names. Intermediate routers maintain server names and their next hop mappings. Just like IP addresses, TRIAD routers also aggregate those names that have common elements and common next hop information into a single unit. However, IP addresses are grouped using

common prefixes, whereas TRIAD names are grouped using common suffixes. For example, *cs.uoregon.edu* and *math.uoregon.edu* will be aggregated into *uoregon.edu*.

- ***Content Router:*** TRIAD replaces IP routers with content routers. Content routers also possess a forwarding information base (FIB) and routing information base (RIB), similar to IP routers. In IP routers, the FIB table contains IP prefix x and the router interface y on which an incoming packet with destination prefix x must be forwarded. The RIB table, on the other hand, contains complete AS-level paths to different prefixes. The RIB table is used to construct the FIB, by finding the best possible path from RIB for each prefix. Instead of storing the next hop and AS path for prefixes in FIB and RIB respectively, TRIAD stores domain name prefixes.

- ***Routing and Forwarding:*** TRIAD introduces two new protocols for routing and forwarding:

  * *Internet Name Resolution Protocol (INRP)*: This protocol is a replacement for the Internet Protocol (IP) [96]. As the name suggests, this protocol is responsible for resolving names at the network level. However, as mentioned earlier, TRIAD only has DNS style names. Thus, INRP resolves each name into next hop, just like IP resolves IP addresses into next hops. Similar to IP, INRP also performs longest name matching. However, INRP performs longest suffix matching instead of longest prefix matching.

    Name-based forwarding is different from IP forwarding in at least one respect. For IP addresses, a given prefix can be announced from only

one location [2]. DNS style suffixes, on the other hand, can be announced from multiple locations. For example, a collection of uoregon.edu servers might be hosted at the Oregon State University (OSU). Thus, intermediate routers must possess a notion of *best server*, when choosing among the common suffixes with differing routing. TRIAD routers use number of intermediate hops to determine the best server. In addition, intermediate routers may also use server load and other performance metrics for making the decisions.

INRP responses either carry the data or the name of a specific host that possesses the data. The responses may carry the data if the content is small. If the content is large, then the responses carry the name of a specific host. For example, when accessing www.cnn.com, clients may receive host1.cnn.com as response. The client can establish a TCP connection with that host to exchange data.

INRP responses are carried along the same path through which the request came in. Each INRP request accumulates router information along the path, which is used to send the responses back. Use of the same path for request and response helps intermediate routers in the following ways:

- Intermediate routers may detect that the *best* content server has failed if they do not receive a response.
- Intermediate routers can deduce the performance of a content server based on the response rate.

* *Name-Based Routing Protocol (NBRP)*: This is an application level protocol, which provides a replacement for BGP. It operates just like BGP,

[2]Anycast addresses can be announced from multiple locations, but that is a special case

i.e., it is a distance vector routing algorithm and it carries the path of content routers towards a content server.

As mentioned earlier, TRIAD uses DNS style names, instead of IP addresses. As a result, NBRP also advertises DNS style names. Specifically, each AS advertises the name prefix that is reachable through it.

Weaknesses: TRIAD's focus has been on supporting DNS style names at the network level so that the best possible content replica can be retrieved. However, TRIAD binds content to domain names, and resolves domain names into IP addresses. As a result, TRIAD does not support persistent content names, which implies if a replica of the same content exists but is bound to a different domain name, then TRIAD will not be able to discover it.

TRIAD also suffers from lack of scalability. TRIAD supports human-readable names at the network layer. Unlike IP addresses, human readable names belong to an almost infinite namespace, and thus may overwhelm the memory in the content router.

5.2.2.2. Named Data Networking (NDN)

Named Data Networking (NDN) [71] is the most recent ICN design. NDN uses hierarchical, human-readable names to address content, and name-based anycast routing over a completely new network infrastructure. However, NDN suffers from the problem of scalability, because NDN routers must handle forwarding tables with hundreds of millions of extremely long content prefixes [97, 98]. In the following subsections, we first discuss architectural elements of NDN, and then its weaknesses in detail.

Architectural Elements: NDN introduces the following architectural components:

- ***Content Names:*** NDN uses human-readable content names at the network layer, instead of IP addresses. The content names are hierarchical, which allows logical grouping of the content. Specifically, NDN requires that human-readable strings be separated by '/' in order to construct NDN name. For example, */uoregon/edu/cs/ghulam/index.html* is an acceptable NDN name. NDN network expects the content names to fall under various prefixes (e.g., */uoregon.edu/cs*) by convention. However, this requirement is not imposed by the architecture.

  NDN names also support segmentation of content. NDN data is divided into segments, similar to packets in today's IP networks. Information about the segments is also carried in the content names. For example, a content name of the form /uoregon.edu/cs/ghulam/index.html/1, indicates the first segment of the index.html page. The segment numbering is not dictated by NDN. It is left entirely upto the applications. For example, a client may send a request for /uoregon/edu/cs/ghulam/index.html and the server may send the data for /uoregon.edu/cs/ghulam/index.html/1. Looking at that name, the client may infer that the segments start from number 1, and based on a previous agreement by an application level protocol, increases monotonically. Thus, the client will start sending requests for /uoregon/edu/cs/ghulam/index.html/2 and so on.

- ***Packets:*** NDN introduces the following two types of packets:

  * *Interest:* These are the request packets issued by a client to indicate that it is interested in receiving data. In NDN, one interest packet corresponds to one data segment. The communication in NDN is entirely driven by

the data recipient. Thus, if a data packet is lost, the client times out and issues another interest packet.

  * *Data:* The data packet carries actual data in response to an Interest packet. In addition, each data packet in NDN also carries the name of the requested data segment. The data packet also carries a signature that binds the data name to the data segment. Because of these properties, a data packet is an independent entity, i.e., it can be independently transported and verified.

- ***NDN Router:*** NDN requires that today's traditional IP-based routers be replaced by NDN routers. An NDN router consists of three components, described below:

  * *Forwarding Information Base (FIB):* Similar to today's IP routers, the FIB in an NDN router stores prefixes and the next hops through which those prefixes are reachable. Of course, in the case of NDN routers, the prefixes are for content, instead of IP addresses. For example, a single FIB entry maybe of the form: $< /uoregon/edu, IF1 >$, where /uoregon/edu is the prefix and IF1 is the interface through which the router should send the packet to the next hop.

  * *Content Store:* NDN routers use the content store for caching the data segments. Upon receiving an interest packet, an NDN router checks whether the requested segment exists in its content store. If the segment exists in the content store, then it is served right away. If not, then the interest packet is forwarded.

  * *Pending Interest Table (PIT):* A PIT holds the interest packets that are pending to be served. Whenever an NDN router receives an interest packet,

it logs the data name and the incoming interface in the PIT. If an entry with the exact same name already exists, then the router simply appends the new incoming interface to the entry. However, if the entry does not exist, then it means that the router received this interest for the first time, and forwards it according to the forwarding strategy.

A PIT automatically builds the multicast capability into the architecture. When an NDN router receives a data packet, it consults its PIT to find the interfaces on which the interest packets were received. Then it forwards the data packets on all those interfaces, thus mimicking the multicast behavior.

The size of the PIT allows the router to indicate the load a router can tolerate: the larger the PIT, the larger the number of packets that the router can handle. It also allows the router to prevent a DoS attack in the form of an interest flood. If the router receives more Interest packets than it can handle, it simply drops those packets.

PIT also prevents the router from an unsolicited data packet flood, which is another form of DoS. If the router receives a data packet for which there is no interest, the router simply drops that packet without propagating it further.

– ***Routing and Forwarding:*** In NDN, routing and forwarding mostly pertains to Interest packets, because data segments follow the same path as the Interest packets, but in reverse direction. Routing and forwarding in NDN is similar to today's IP routing and forwarding. For building the routing tables, different ASes announce the content prefixes that they can serve, instead of the IP prefixes.

For forwarding Interest packets, the routers find a longest prefix match in their FIB, and forward the packet to the next hop. When the data segment comes back from the content provider, at each hop the content router consults its PIT and sends the data segment on the interface on which the Interest packet was received, thus *consuming* the PIT entry. This process of symmetric routing is referred to as *bread-crumb* routing in NDN.

- ***Security*** As content is the focus of this architecture, security is focused on the content as well, i.e., *data-centric security*. By embedding security in the content, the architecture saves the end hosts from trusting the end points or the network paths. NDN employs the following two mechanisms for data-centric security:

  * *Data Signatures:* The NDN architecture's main security contribution is that it securely binds the content names with the content. This is a direct benefit of using the content names for routing. As a result of this approach, each packet can be independently verified whether it carries the data for the name that it purports.

    Each data packet carries a data signature that is computed over the data name and the content. The signature is computed using a standard signature generation algorithm. The packet also carries the information about the key that is used to generate the signature. That information may carry the name of the key or the location from where the key can be obtained.

NDN treats the signature keys just as any other data. The name of the key is the name of the key provider, and the content is the key itself. These packets are signed by a well known source of trust (e.g., Verisign).

At the lower layers, NDN only checks the binding and the integrity of the data via the signatures. The trust in the signature (i.e., whether the signatory can be trusted) is managed at the higher layers, where it depends on the context of the trust. For example, for a banking website, it is important that the signatory is verified to be the bank itself. However, for a blog, only verifying the integrity of the data might be enough. This verification is performed by NDN routers.

This form of data-centric security helps in preserving the integrity of the data and in verifying the origin of the data. Thus, data-centric security satisfies the requirement of security being built into the architecture.

* *Data Encryption:* Data encryption is used as an access control mechanism in NDN. Only the end hosts that possess the decryption key can access the data. However, the protocol does not impose any restrictions on the decryption keys. In addition, the protocol does not specify any means of obtaining the decryption keys.

Weakness: The biggest weakness of NDN is its use of human-readable content names at the network layer, which results in lack of scalability. The human readable names, unlike IP addresses, come from an almost infinite namespace, which implies that the number of prefixes stored in the FIB of tier-1 AS maybe infinite. Even by conservative estimates [97], a backbone NDN router would consume extra 3,302 W of power, and additional $130,000 to build.

Another problem that NDN faces is the possible lack of aggregation in names. ISPs are allowed to announce completely arbitrary names, and in the worst possible case there may not be any aggregation of names at all. Furthermore, the hosts are allowed to move to other ASes and simply announce their prefixes as they move. This provision for mobility implies that even if aggregation among prefixes is possible, it may not be applied because different prefixes maybe served from different interfaces.

5.2.2.3. Data-Oriented Network Architecture

DONA [69] is a more recent ICN design that leverages flat, self-certifying names and name-based anycast routing built on top of IP. DONA suffers from the problem of lack of scalability and lack of persistent names, as per the requirements mentioned in Section 1.3.1.. In the following subsections, we first describe architectural elements of DONA and then its weaknesses.

Architectural Elements: DONA introduces the following architectural elements:

- ***Content Names:*** Content names in DONA are of the form *P:L*, where P is the hash of public key of the content provider, and L is a human-readable label. Content consumers use P to verify the integrity and authenticity of content. Whenever a content consumer retrieves content, it receives the triplet $< data, publickey, signature >$. The content consumer verifies the integrity by using the signature and the public key. The consumer verifies the authenticity of the content by hashing the public key and comparing it against P. If the two values match, then it shows the content came from the original provider.

  Embedding the public key in content names makes the names user-unfriendly. For example, URLs can be passed verbally because they are very user-friendly. But, DONA names cannot be passed verbally. DONA expects

that content consumers will learn DONA names via search engines, personal recommendations through digital means (e.g., email), and social networks (e.g., Facebook [99], Twitter [100]).

- ***Packets:*** DONA uses the following two types of packets:
  - REGISTER: The content provider uses the REGISTER packet to inform all the relevant RHs about the availability of its content. The register packet either carries the name *P:L*, which specifies a specific content item, or P:*, which indicates that all of P's content is available at this content provider. REGISTER packets are DONA's control plane packets, similar to BGP, because they set up the state for later content discovery.
  - FIND: The content consumer uses the FIND packet to reach the closest possible replica for content name *P:L*. The FIND packets constitute the data plane traffic in DONA. These packets are carried within the IP packets, by creating a shim layer between IP and transport headers.
- ***Resolution Handlers:*** DONA introduces a new class of network entities, called resolution handlers (RHs). Each autonomous system (AS) in today's Internet must run an RH to be part of DONA. Each RH maintains *registration table*, which stores name to next RH hop mapping. Names can be of the form *P:L* or P:*, as described above. The registration table also maintains distance to the copy of the content in terms of RH hops. In addition to maintaining registration tables, RHs are also responsible for forwarding REGISTER and FIND packets, as described below. In order to forward traffic, RHs maintain the same relationships with other RHs, as their respective ASes.

– ***Routing and Forwarding:*** Each RH forwards a packet depending on packet's type. Whenever an RH receives a REGISTER packet from its customer or peer, it forwards the packet to its provider(s) and peer(s), only if either no such record exists or the REGISTER message carries the name of a closer replica. Whenever an RH receives a FIND packet, it first performs a *longest prefix match* on its own registration table. DONA's longest prefix match mechanism is different from the longest prefix match in IP: instead of comparing bits, DONA compares components. Specifically, the queried name, *P:L*, carried in the FIND packet, may match just P or *P:L* in the registration table. If the packet matches *P:L*, then that next hop is chosen, otherwise the next hop that matches P is chosen. At each next hop, the respective RH forwards the packet further, until it reaches the content provider. When a FIND packet reaches a content provider, it establishes a transport connection with the sender of the FIND packet, after which the data is transmitted using the IP network.

  On the other hand, if no match is found in the registration table, then the RH forwards the FIND packet to its provider. If the FIND packet reaches a tier-1 RH, and does not find a match, then the name is considered to be absent, and the tier-1 RH returns an error to the source of the packet.

Weaknesses: In this subsection, we discuss the weaknesses of DONA, as per the requirements mentioned in Section 1.3.1.. The key problem that DONA faces is that of scalability. Specifically, each tier-1 RH is required to store the next hop for all content items that exist. According to Google's measurements, approximately 1 trillion web pages are in existence [2]. Furthermore, everyday approximately $150,000$ new domains are announced [101], and a single domain consists of $5,000$ pages on average [102]. If content providers REGISTER individual web pages, then the tier-1

RH has to pay significant storage and bandwidth cost. In addition, given the flat nature of content names, they cannot be aggregated, unlike today's IP addresses.

DONA also lacks persistent content names. Specifically, the human readable L in each content name is bound to content provider's public key P. If the provider of the content changes, then the name of the content must change is well. Furthermore, if the same content is provided by two different providers, it cannot be retrieved without making a decision about the content provider first. In other words, the presence of public key in content name is equivalent of today's domain names in content names.

5.2.2.4. Publish-Subscribe Internet Routing Paradigm (PSIRP)

PSIRP [70] is an ICN approach centered around the publish-subscribe model: publishers publish the location of content in the network, and consumers subscribe to content names. While PSIRP builds an extensive architecture for content distribution, its *rendezvous layer* is responsible for resolving content names into addresses, and thus is the most relevant component to our research. Consisting of an *overlay network* and several *rendezvous networks*, it mainly uses distributed hash table (DHT) for resolving content names.

Architectural Elements: PSIRP introduces the following architectural components:

- ***Naming:*** PSIRP uses three types of names for different levels of the architecture:
  - *Application IDs:* As the name indicates, these identifiers are specific to a given application. Such identifiers exist in today's Internet as well. Examples include URLs for the world wide web, and file names for the file sharing peer-to-peer networks. These identifiers hold meaning inside the application only, and are not understood by the network.

* *Rendezvous IDs:* Publishers and subscribers use rendezvous IDs to inform the network that they are exchanging information of mutual interest. While application IDs are understood by the applications, rendezvous IDs are understood by the network.

* *Forwarding IDs:* As the name indicates, forwarding IDs are used for forwarding packets. Each link in PSIRP is identified by a forwarding ID.

As mentioned earlier, all the above-mentioned IDs are used at different levels of the architecture. Application IDs are resolved into rendezvous IDs and rendezvous IDs are resolved into forwarding IDs.

– ***Rendezvous Nodes and Rendezvous Networks:*** Rendezvous nodes provide a meeting point for publishers and subscribers. Each autonomous system (AS) in the PISRP architecture runs at least one rendezvous node. Rendezvous nodes create an overlay similar to the BGP border routers. Such an overlay is called *rendezvous network*.

Multiple rendezvous networks exist in a PSIRP Internet. Each rendezvous network is rooted at a provider ISP. PSIRP faces the challenge of connecting multiple rendezvous networks. PSIRP creates a distributed hash table (DHT) [26, 103] using the root of each rendezvous network. The rendezvous IDs are the keys in the DHT, and the forwarding IDs are the values.

– ***Routing and Forwarding:*** In order to publish a rendezvous ID, a publisher informs its upstream router about the ID. Information about the rendezvous ID propagates until it reaches the root of the rendezvous network. The root, then publishes the rendezvous ID in the top-level DHT. When a subscriber wishes to subscribe to a rendezvous ID, it informs its upstream router. The subscribe

requests keeps propagating upstream until either a provider for rendezvous ID is found or the the request reaches the top-level DHT. In the top-level DHT, the search is performed using the well-known DHT mechanisms.

As described earlier, publish and subscribe mechanisms at the rendezvous layer match publishers and subscribers with each other. In order to allow the publishers and subscribers to communicate with each other, each AS also runs a *topology manager*. The high level goal of topology construction is to construct logical multicast tress for efficient content delivery. Whenever a rendezvous node matches a publisher and subscriber, it requests the topology manager to either add the node to an existing multicast tree or construct a new tree.

Rendezvous and topology formation constitute PSIRP's control plane, whereas forwarding (described next) makes PSIRP's data plane. The end result of rendezvous and topology construction is a set of forwarding IDs.

– ***Security:*** PSIRP uses public key cryptography to identify each host i.e., each host possesses a public/private key pair. These keys are different from self-certifying names, because they cannot be used for routing. In order to use public key cryptography, PSIRP proposes the use of Packet Level Authentication (PLA) [104]. PLA includes a certificate in each packet, which binds the identity of a host with its public key. In addition, the packet also carries a signature, which is a hash of the packet signed by the sender's private key. Presence of this certificate means that the host is legitimate, because it must have acquired the certificate from a trusted third party. In addition, presence of the signature means the sender cannot deny that it sent the packet. Combination of the certificate and the signature creates accountability, which can deter end hosts from malicious acts (e.g., DoS attacks).

Currently, PLA cannot be deployed because today's hardware cannot verify signatures and certificates at line speed. The designers of PSIRP believe that during the next few decades, such hardware will be available.

Weaknesses: PSIRP's rendezvous networks and the DHT that connects the rendezvous networks are the closest components to content discovery. We believe that the DHT can cause scalability problems. Specifically, the root of each rendezvous network is required to participate in the DHT, without regard to its capacity and infrastructure. A DHT equally distributes the resolution state across all the participating nodes. As a result, each AS is required to store the same resolution state, whether it is a tier-1 AS or a tier-n AS. In other words, the ASes do not have autonomy over the resolution state they must store.

5.3. Design

Compass is a content discovery service for the Internet. An end-host on the Internet, or a **content consumer**, can query Compass and learn the locations of content. For example, a consumer can first use a search engine and input some keywords to obtain a content name, and then use Compass to locate the content based on its name. Content is originally generated by a **content provider**, but can be replicated and hosted at multiple **content servers**. For simplicity, in this section every content location is an IP address. But, in Section 6.1. we provide concrete details of how different entities in Compass communicate with each other. Below we explain the design details of Compass and how it meets the key properties outlined in Sec. 5.1..

5.3.1. Content Name and ID

Compass distinguishes between *content name* and *content identifier* or *content-ID*. Used by higher-level applications such as search engines, a content name is a human readable name assigned to a content item by its content provider. (We do not dictate how these names are chosen and how these names are kept unique, in order to avoid tussle [105].) A content-ID is a cryptographic hash of a content name (e.g., SHA-1 [37]). The conversion from content name to content-ID happens at the consumer. Compass works on the content-IDs only.

Compass uses content-IDs rather than content names for publication and resolution operations, in order to put a bound on the content name space and improve storage efficiency. While the name space for human-readable names is almost infinite, content-IDs are of a fixed length (128 bits), and the space with content-IDs in Compass is 2^{128} content items. Based on cryptographic hash functions, content-IDs also has a low collision probability. Even for 1 trillion content items [2], the collision probability of a hash function is $p \leq \frac{n(n-1)}{2^{(b+1)}}$, where n is the total number of items that are hashed, and b is length of each content-ID in bits. Assuming $n = 10^{12}$ and $b = 128$, we then have $p \sim 10^{-16}$.

As shown above, there is a vanishingly small probability that collisions may occur. In order to avoid collisions altogether, content providers include the human-readable content name in content metadata during the content-ID publication process, and the resolution request also carries the human-readable content name along with the content-ID.

5.3.2. Compass Operation Overview

The main player in Compass is **Content Discovery Agents (CDAs)**. When a network administrator would like her users to be able to resolve content names quickly, she can set up a CDA for her users. Similar to today's DNS resolvers, this CDA is also called the **local CDA** for her users.

At its own discretion, every CDA can further choose to store location information for certain content, usually for all content sharing the same content-ID prefixes. We call such CDA a **resolving CDA**. Specifically, the CDA maintains a **resolution table (RT)** to store a list of `<content-ID, IP address>` tuples for those content-IDs that the CDA maintains. For simplicity, we assume every resolving CDA is responsible for one content prefix, and our design can easily be extended to allow more than one content prefix per resolving CDA.

Compass consists of multiple CDAs, and these CDAs can form two overlay networks: One is an unstructured overlay formed by *all CDAs*. It is designed for resolving the locations of content, and we call it the **resolution overlay**. The other is a structured overlay formed by *all resolving CDAs*. It is designed for publishing content locations at resolving CDAs, and we call it the **publication overlay**. In other words, every CDA joins Compass by becoming a node on the resolution overlay, and if it is a resolving CDA, it will also need to join the publication overlay. We describe the management of these two overlays in Sec. 5.3.3. and 5.3.4., respectively.

With the two overlay networks, Compass supports three fundamental operations: **announcement**, **resolution**, and **publication**. Below we first provide a high-level overview of these operations, and then illustrate them in detail in Sec. 5.3.5.,5.3.6., and 5.3.7., respectively.

- *Announcement:* A resolving CDA uses the announcement operation to advertise its willingness to help resolve a particular portion of the content-ID space, as indicated by a content-ID prefix. It generates a content prefix announcement to propagate through the resolution overlay, in the same way as today's BGP (Border Gateway Protocol) announcements propagate through today's ASes (autonomous systems). As prefix announcements propagate through the resolution overlay, every CDA en route learns its path for reaching a resolving CDA in charge of the content prefix in question.

- *Resolution:* A content consumer uses the resolution operation to find the "best" CDA and learn all the locations of a content item, where "best" may mean the closest CDA or the one least costly. When resolving a content-ID, a content consumer contacts its local CDA with a resolution request. The local CDA finds the next hop where it should forward the resolution request. The next hop repeats the same process and forwards the resolution request further. In this way, the resolution request eventually reaches a CDA that can resolve the given content-ID.

- *Publication:* When a content provider creates new content, or moves content to a new location, the provider uses the publication operation to inform all resolving CDAs that need to maintain the mapping between a given content-ID and its locations. Like the resolution operation, the content provider also first contacts a local CDA, which then further routes its publication request on top of the publication overlay to all the resolving CDAs responsible for the content in question.

5.3.3. Resolution Overlay

We now describe how the resolution overlay is formed. For every new CDA that wants to join Compass, it chooses one or more existing CDAs to establish connections with. In doing so, it must first seek authorization from such existing CDAs in the form of out-of-band agreements. Similar to today's AS (autonomous system) relationships on the Internet [106], these agreements can be of a peer-to-peer or customer-provider nature. An existing CDA may agree to be a new CDA's provider in return for payment, or it may agree to be a new CDA's peer and they may exchange similar amounts of Compass traffic between themselves.

After a CDA joins the resolution overlay, if it is further a resolving CDA responsible for a particular sub-space of the entire content-ID space, neighboring resolving CDAs can form partnerships with each other to share the content resolution workload, thus being able to resolve the entire content ID space collectively. We call such a set of neighboring CDAs a **content resolution realm**. Figure 5.1. shows an example with 11 CDAs and 3 realms.

A CDA can initiate a realm with itself as the sole member, and then have more CDAs join the realm. The CDA that initiates a realm is called **realm root**, or **root**. The root of a realm is initially responsible for storing the resolution information for the entire space. As more CDAs join the realm, the resolution state is distributed from the root to all member CDAs.

We believe that a CDA will initiate or join a realm only if it has a strong incentive. While real-world Compass deployment will reveal real-world incentives for CDAs, in this chapter we can only speculate. We believe a CDA would start a new realm to reduce resolution latency for its customers. On the other hand, a CDA may join an

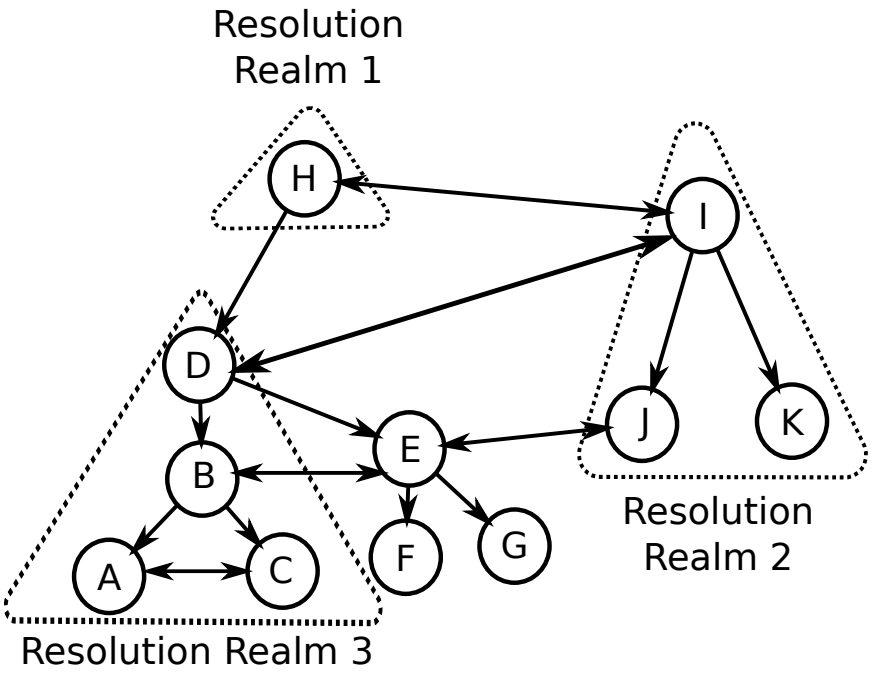


FIGURE 5.1. A Compass resolution overlay. CDAs with a double-arrowed link between them form a peer-to-peer relationship, but otherwise a customer-provider relationship. It has three content resolution realms, with each realm storing the resolution information for the entire content-ID space, while four CDAs are not part of any realm.

existing realm for monetary gain. For example, a CDA may join the realm of its transit CDA and share the resolution load in exchange for reduction in transit cost.

CDA	Realm	Prefix
A	3	0x0/2
B	3	0x1/2
C	3	0x2/2
D	3	0x3/2
H	1	0x0/0
I	2	0x0/2
J	2	0x1/1
K	2	0x1/2

TABLE 5.1. Content Prefixes Announced by Different CDAs

Table 5.1. shows possible prefixes for CDAs shown in Figure 5.1.. The prefixes are shown in CIDR notation [107]. For example, Table 5.1. shows that CDA K belongs to resolution realm 2, and announces 2 bit prefix $0x1$.

5.3.4. Publication Overlay

The goal of the publication process is to disseminate the publication information to a subset of relevant CDAs. This problem can be reduced to application-level multicast. There has been plenty of work in the area of application-level multicast, especially after the inception of Distributed Hash Tables (DHTs). Following this line of work, we also use structured network for publication.

Application-level multicast roughly falls into two categories: create a multicast group and then broadcast over that group [108], or construct a spanning tree, and then multicast messages within the tree [109]. Both the approaches incur significant group-creation overhead, but it is bearable because multicast sessions last for several minutes/hours.

Compass faces a challenging problem because the multicast sessions consist of disseminating a particular publication announcement, and must conclude as soon as possible. In this regard, if the multicast members repeatedly join different groups, then the overhead may become unbearable. Thus, we use a slightly different approach for multicast than regular group formation and message dissemination.

All resolving CDAs in Compass form a publication overlay to facilitate the publication operation. The overlay is a tree, and every CDA is positioned on the tree according to its responsibility. If a CDA is responsible for a larger content-ID space, it will be at a higher level of the tree. For example, if a CDA X announces a 4-bit long prefix 0xc and CDA Y announces an 8-bit long prefix 0xc8, X will be

at a higher level than Y, thus an ancestor of Y. When two CDAs announce the same prefix, however, they are peers, and we treat them collectively as one logical node on the tree. For all CDAs that announce content prefix 0x0/0—i.e., each is a resolving-CDA for *all* content-IDs, they will collectively form the root of the tree as a single logical node.

Every resolving CDA maintains a **Publish Neighbor Table (PNT)** to store information about its ancestors, descendants and peers in the tree. Every entry in a PNT is a `<content-ID prefix, IP address of CDA, type of neighbor>` tuple, where the type of neighbor can be parent, child, or peer. This way, the CDA can know what content-ID prefixes its parents, children, or peers have announced, and their IP addresses. Note peering CDAs will have identical PNTs.

A resolving CDA Z that is responsible for a content prefix P joins the publication overlay by finding its parents, peers and children in the overlay. To do so, Z resolves a random content-ID contained within P (e.g., if P=0xde/8, Z may resolve content-ID 0xdeff..ff) and finds a resolving CDA J for the random content-ID. Depending on $J's$ announced prefix, J will be either an ancestor, a peer, or a descendant of Z. Compass can then use J as an anchor to walk through the tree and retrieve $Z's$ parents, peers and children. The insertion process is completed once Z informs its parents, peers and children about its presence in the tree, along with the prefix P it is serving.

Using the above-above mentioned methodology, we arrange the prefixes mentioned in Table 5.1. in a binary tree, which is shown in Figure 5.2.. Figure 5.2. shows one node for each announced prefix. However, there could be multiple CDAs that announce the same prefix. For example, as shown in Table 5.1., CDA B and CDA K announce the same prefix (i.e., 0x1/2). In Figure 5.2., both the CDAs are represented by the same node 0x1/2, and are thus considered as peers. The labels

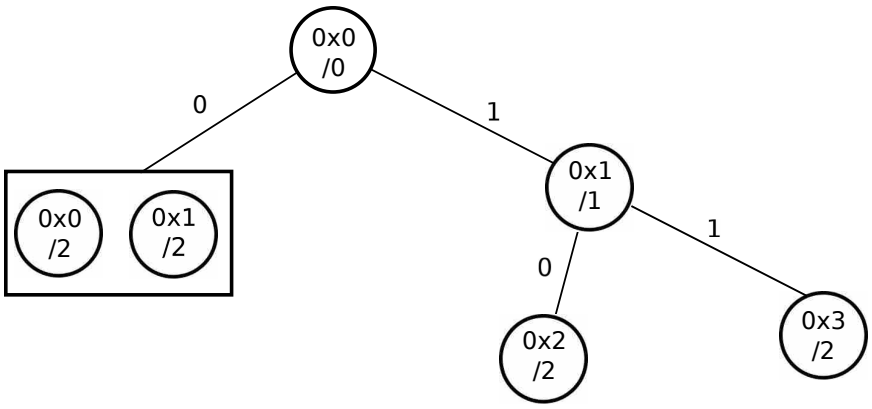


FIGURE 5.2. Publication Overlay

on edges indicate the bit value of a particular prefix and the level of the tree indicate the bit index in a particular prefix. Both the values combined (i.e., bit value and bit index) together determine a prefix's position in the tree. Figure 5.2. also shows that prefix 0x0/0 have two children along the left branch (i.e., branch 0). This is possible because the tree is not a balanced binary tree. Specifically, none of the CDAs announced the prefix 0x0/1, which is the direct parent of 0x0/2 and 0x1/2. As a result, both the prefixes are stored as children by the grandparent 0x0/0.

The publication overlay also needs to be maintained in presence of the failures of CDAs. If a CDA notices a neighboring CDA has departed from the publication overlay (e.g., its communication with that CDA continues to fail), it will then mark that CDA as "departed" in its PNT, and further inform its neighbors about the departure, until the parents, peers, and children of the departed CDA are all informed.

5.3.5. Announcement Operation

When a resolving CDA announces the content prefix it is responsible for, the prefix announcement message propagates through the resolution overlay in a way similar to BGP routing announcement on today's Internet [110]. The announcement

message includes the content prefix in question, as well as a CDA path that is an ordered list of all the CDAs to go through in order to reach the resolving CDA.

If a CDA receives a content prefix announcement from a neighboring CDA, it will record the content prefix and the CDA path contained in the announcement. Moreover, it constructs or updates a **resolution forwarding table (RFT)** to maintain the next-hop CDA for the announced prefix. Every entry in the RFT is a tuple in the form of `<content-ID prefix, next-hop CDA>`. If the CDA decides this newly learned path is the best path for resolving the content prefix in question, it will prepend itself to the path, and further notify its other neighbors about the new path.

Multiple CDAs can announce the same content prefix, thus multiple CDAs can store the same resolution information for the same content prefix. The direct benefit in doing so is that content consumers can discover the location of content through the nearest resolving CDA for the content, thus a latency reduction, and the whole system will be more fault-tolerant.

A CDA may also withdraw an existing prefix (or its path to the prefix) by announcing the withdrawal of the prefix. The withdrawal announcement propagates the same way through the CDA network as an announcement for a new prefix. When a CDA receives a withdrawal announcement, it checks whether its path for the withdrawn prefix needs to be updated. If the CDA appears to have no path to the prefix any more, it will forward the withdrawal announcement to its neighbors. If it appears to have to use a new path, it will update its RFT and send an announcement of new path to its neighbors.

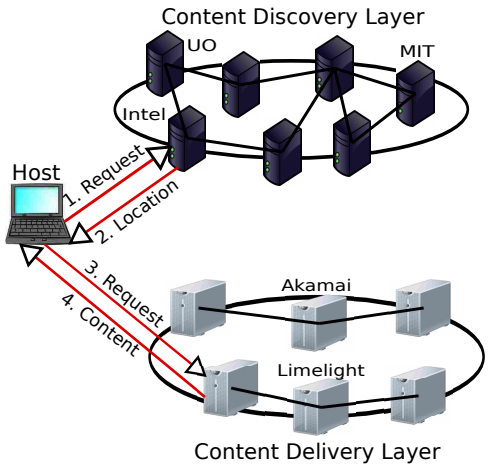


FIGURE 5.3. Search Flow

5.3.6. Resolution Operation

The high-level process of searching content name to location mapping is shown in Figure 5.3.. In the first step, a content-consuming host sends a content name to the content discovery layer so that it can be resolved into location. Each end host is configured with the IP address of a first-hop content discovery agent, similar to a first hop DNS server. End hosts contact the content discovery layer through first hop content discovery agents. In the second step Content discovery layer searches for the given content name, and returns the IP address of the *best possible* content server. In the third step, the end hosts establishes a TCP connection with the content server given by content discovery layer. In the fourth step, the end host starts retrieving content.

When a CDA receives a resolution request from a content consumer for a content-ID, if the CDA is a resolver for the content-ID, it consults its resolution table (RT) and resolves the content-ID to an IP address. The CDA then sends a resolution response towards the content consumer. Otherwise, it performs a longest prefix match on its

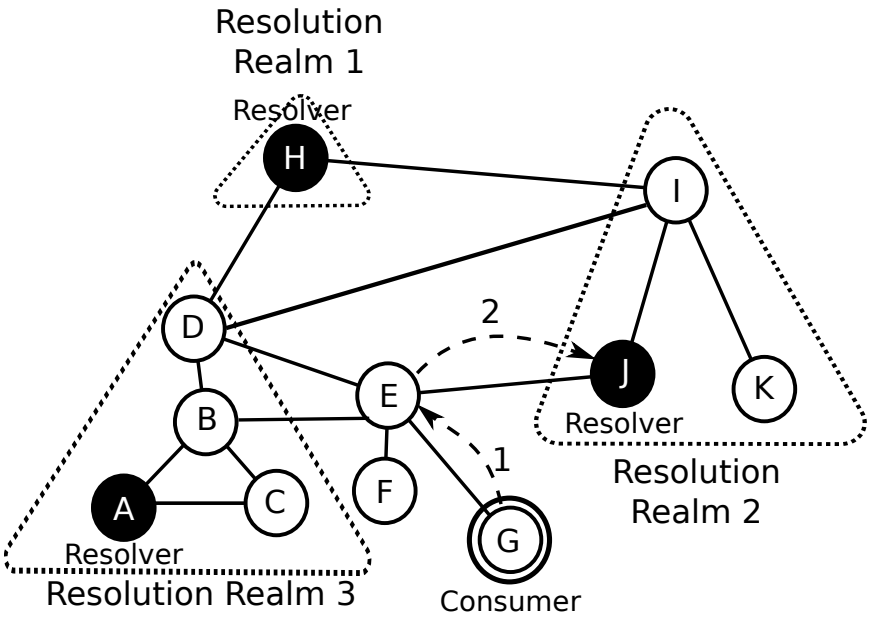


FIGURE 5.4. Content-ID resolution operation in Compass. A resolution request originates from CDA G and is forwarded via CDA E toward CDA J, which is the closest resolver to the consumer.

resolution forwarding table (RFT) to determine the next-hop CDA for the resolution request.

The resolution request can be either iterative or recursive. If iterative, the CDA sends the next-hop CDA information to the content consumer, who then further contacts the next-hop CDA. If recursive, the CDA will directly forward the request to the next-hop CDA. Either way, the process continues until the content consumer receives a resolution response. Once the content consumer receives the resolution response, it then can retrieve the content. Figure 5.4. shows an example of recursive resolution operation.

In Figure 5.4., we show the resolution process for a request for content ID 1110 which originates in CDA G. The RFTs for CDAs G and E are shown in Table 5.2. and Table 5.3., respectively. When the resolution request is received at CDA G, it performs longest prefix match (LPM) on its RFT. $G's$ RFT contains only one entry -

Prefix	Next Hop
0x0/0	E

TABLE 5.2. RFT for G

Prefix	Next Hop
0x0/1	B
0x1/1	J

TABLE 5.3. RFT for E

i.e., prefix 0x0/0, which is reachable through E. This is because G is not part of any realm, does not announce any prefix and has only one provider. As a result, G always forwards all the resolution requests to its provider, E. When E receives the resolution request from G, it also performs LPM on its RFT. $E's$ RFT contains only two entries because it aggregates the prefix announcements that come to it. For example, in this case, prefix announcements 0x0/2 and 0x1/2, from CDAs A and B respectively, are aggregated into 0x0/1, which is reachable through B. After performing LPM on its RT, E determines that the next hop for the request is J. Since J announces 0x1/1 (see Table 5.1.), it is the best possible CDA for content ID 1110, given that the content ID exists in $J's$ resolution table.

During resolution, if a CDA finds that multiple IP addresses exist for a given content-ID, then instead of making a choice, the CDA sends all the IP addresses to the content consumer. Then, it is upto the application running at the consumer's end how it uses these IP addresses. For example, the application may send initial hand-shake packets to all the IPs, and the one that responds first is the host with lowest latency, and thus most desirable. An application may also communicate with all the IPs in parallel, and receive different blocks of data from different hosts. In doing so, Compass pushes the decision to choose proper host content retrieval to the end host.

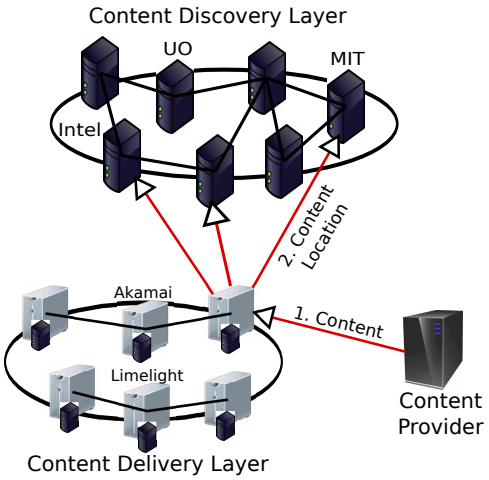


FIGURE 5.5. Publish Flow

5.3.7. Publication Operation

The high-level process of publishing content name to location mapping is shown in Figure 5.5.. In the first step, the content provider hires a CDN (e.g., Akamai), which replicates content on its servers. In the second step, the content delivery layer distributes content name to location mapping in the content discovery layer.

We now explain the publication operation that a content provider uses in order to publish a content with content-ID "C" to *all* resolving CDAs for C. The content provider sends a publication message for C to its local CDA, e.g., CDA X. X looks up in its resolution forwarding table (RFT) the longest content prefix that matches C, and identifies the next-hop CDA toward which it forwards the publication message. This procedure continues until it finds a CDA Y that is responsible for a prefix that contains C, i.e., Y should store the location of content-ID C.

Y then further forwards the publication message to all its peer CDAs in the publication overlay, and one randomly chosen parent. The parent also forwards the publication message to its own peers and one randomly chosen parent. This process continues until the root of the tree is reached. Furthermore, Y also forwards the

publication messages to a randomly chosen child which is also a resolving CDA for C (i.e., its content prefix matches C). This process continues until a leaf in the tree is found or there are no more children that are a resolving CDA of C.

Compass uses the same procedure above to update an existing content-ID-to-location mapping or delete an existing content-ID-to-location mapping. An update message carries a content-ID, an old location, and a new location. A delete message carries a content-ID and a location.

5.4. Security

In this section we outline and address the fundamental security issues that Compass faces. Before outlining security challenges, we first clarify the terminology used in the rest of the section:

- *Content Publisher:* A host that is the original contributor of the content.
- *Content Replicator:* A host that replicates the content contributed by the content publisher (e.g., CDNs).
- *Content Name Resolver:* A CDA that resolves content names into their locations.

Compass faces two major security challenges, and they are described in the next two subsections.

5.4.1. Protecting the Binding

Compass must protect the binding between content names and their locations. Specifically, a content name resolver may change the binding from what the publisher publishes - i.e., it may change the name of the content or the location of the content.

We solve this problem by using public key cryptography, certificates and a certificate authority. Each publisher generates its own public/private key pair, and obtains a certificate from a certificate authority that carries its public key. The publisher uses the private key to sign the content name to location binding, as described below, and the certificate vouches for the public key of the publisher.

So far, the Compass design only requires the publisher to publish the location of the content, and content metadata, which includes content name along with other information (e.g., content type, content encoding). However, in order to solve the binding problem, mentioned above, content publisher must also publish and content name resolver must store the following additional information:

– Signature

– Certificate

In addition, Compass also requires that content publishers must include the IP address or IP prefix of the content in the content metadata.

In order to generate the signature, the content publisher computes a cryptographic hash (e.g, SHA-1, MD5) of the content metadata, and encrypts the hash with its own private key The inclusion of the IP address or IP prefix and content name in content metadata essentially means that the publisher itself signs the binding between content name and its location.

5.4.2. Protecting Prefix Announcements

Compass faces another security challenge that an intermediate CDA A may tamper with a prefix announcement in the following ways:

– Modify the prefix that was originally announced.

– Modify the path that the announcement has taken to reach A.

In order to solve the above-mentioned problem, Compass requires that each CDA must do the following:

– Generate a public/private key pair.

– Acquire a certificate from a certification authority that vouches for the CDA's IP address, port number and public key.

A certificate authority may employ several different mechanisms to confirm that the IP address, port number and public key belong to the claimed CDA before issuing a certificate. One simple way to vouch for this information is to send a challenge (e.g., what is the sum of 2 and 2), which is encrypted with the claimed public key of the CDA, to the claimed IP address and port number. If the CDA is able to receive the message and able to correctly answer the challenge, then that means the claimed information is correct, and the CA can vouch for the CDA.

When a CDA announces a prefix, it signs the prefix and the path attribute of the packet with its private key. It then includes the signature and its certificate in the packet. Each CDA along the path repeats the same process: adds its own IP address and port number to the path attribute, signs the prefix and the path attribute with its own private key, and includes the signature and its own certification. In order for this approach to work, each CDA must carry the public keys of all existing certification authorities, so that the integrity of the certificate can be verified. This is a requirement because CDAs can acquire their certificates from any certification authority.

Since each CDA along the path signs the prefix announcement, and the signatures from all CDAs are included in the prefix announcement packet, the integrity of the packet can be verified at any CDA. In order to verify the integrity of the packet,

a CDA Z would go through all the signatures and certificates in the packet one by one, starting from the first signature. In order to begin the verification process, Z first removes all the CDAs from the path attribute, except for the first CDA A that announced the prefix. Then, Z computes a cryptographic hash over the announced prefix and the path attribute. Next, it decrypts the signature of the CDA A by using the public key from A's certificate, and retrieves the cryptographic hash computed by A. If the two hashes match, then it means that prefix that Z received is valid along with the path attribute until the second CDA. In addition, if the IP address and port number in the path attribute match the IP address and port number in A's certificate, then it also means that A did not falsify its own identity. CDA Z repeats the same process from second CDA, and continues until the integrity of the prefix announcement is verified for all preceding CDAs.

Our proposed approach is similar to BGPSEC [111], with difference in the level of security. Compass requires that the certificates must include the identity of a CDA (i.e., IP address and port number), and the public key is mapped to the identity of a CDA. BGPSEC, on the other hand, requires that the certificates must map IP prefixes to public keys. Keeping this in mind, our proposed mechanism enables *accountability*, whereas BGPSEC enables *authorization*. More specifically, in Compass, if a CDA is involved in a malicious activity regarding prefix announcements (e.g., tampering with announcement packets), then it can be held accountable for that activity. BGPSEC provides the same level of accountability, but it also enables authorization, because no AS can announce a prefix unless it is authorized by a central authority (e.g., RPKI [112]). This authorization comes in the form of the certificate: an AS cannot posses a certificate that maps its prefix to a public key, unless it is authorized.

The level of security provided by Compass is weaker than BGPSEC, and for good reason. BGPSEC assumes that if an AS announces a prefix, then no other AS can announce the same prefix. However, Compass allows multiple CDAs to announce the same prefix. Thus, Compass does not require authorization; it only requires accountability. Moreover, a weaker security model allows Compass to rely only on certificate authorities, rather a centralized root of trust, which has been the major hurdle in the deployment of BGPSEC.

5.4.3. Reputation

Compass faces the problem of determining the reputation of a content provider (i.e., publisher). In today's domain-oriented naming system, end hosts might prefer one domain over another to retrieve content. For example, content from CNN might be considered more reliable and trustworthy than content from FOX.

As mentioned in Section 5.3.1., such a naming scheme tangles the name of content provider with actual content, which affects the name persistence of content - i.e., as content moves to another domain it needs to be renamed. As a result, Compass advocates a clean separation between content provider and content name, using only content name to discover its location. However, in doing so, Compass loses the built-in trust model that today's domain-oriented naming provides.

We solve the problem of reputation by including the name of publisher in the certificate, which is issued to the publisher by a certification authority. Compass does not dictate any structure on the name; the name can be anything that the publisher wants, as long as it is unique. In order to ensure uniqueness of names, a global registry can be employed, similar to today's DNS registry. With this addition of publisher

name, the certification authority vouches for a publisher's name as well as its public key.

The simple approach of including publisher name in the certificate solves the reputation problem, because when during the resolution process, end hosts receive multiple certificates for various publishers offering the same content, it can choose based on its own preference.

5.4.3.1. Content Distribution Networks

The simple approach of including the name of a publisher in the certificate fails when the content publisher uses a content distribution network to replicate its content. Our approach of protecting content name to location binding, as described in Section 5.4. does not distinguish between an original content publisher and a content replicator. Specifically, if a CDN replicates the content of a content provider, the CDN would use its own certificate to sign the content name to address mapping. While this approach works to protect the binding between content name and location, it does not give any clue to the end host about who originally created the content. Thus, the end host may have trouble trusting the content.

To solve this problem, we create an authorization mechanism through which the original content publisher authorizes content replicators to replicate content. In order to seek the authorization, the content replicator sends all the IP prefixes, which may replicate content, to the content publisher out-of-band. The content publisher signs the binding between content name and each IP prefix, and sends all the signatures to content replicator out-of-band. These signatures constitute the authorization from the original content publisher to the content replicator. From this point onwards, the

content replicator can publish the content by using the signatures provided by the original content provider, and the certificate of the original content provider.

As a result of this authorization mechanism, when an end host retrieves content name to location mapping from a content name resolver, it gets the certificate of original content provider along with other information. As discussed above, this certificate may help the user in deciding which content to trust more.

5.5. Recap

In this chapter we presented Compass, a new content discovery service for the Internet. Compass supports persistent content names, which is the key requirement for any content discovery service. Specifically, Compass resolves content identifiers (cryptographic hashes of human readable content names) into locations (i.e., IP addresses). Furthermore, Compass is scalable because it allows CDAs to decide the extent of their participation in storing content ID resolution information. Specifically, a CDA that stores content ID resolution information announces a content ID prefix, which indicates the range and amount of stored resolution information. In addition, Compass maintains tight control over system membership, by forcing CDAs to form agreements before joining the system. Finally, Compass is deployable over today's Internet without requiring any changes in the underlying infrastructure.

CHAPTER VI

COMPASS: IMPLEMENTATION, EVALUATION AND ADDITIONAL ISSUES

Text from this chapter is based on work submitted to Infocom 2014. My advisor Prof. Jun Li, Dr. Matteo Varvello and I were the primary contributors to this work.

In this chapter we present the implementation of Compass, evaluate Compass using our simulation framework, and present additional issues related to the deployment of Compass.

6.1. Implementation

In this section we discuss the implementation of Compass, including the communication protocols used and the format of messages exchanged. Our goal in this section is to explore the feasibility of Compass operations.

We have implemented a single CDA in python in 900 lines of code. The program takes local host information (i.e., IP address, TCP port number and UDP port number), prefixes announced by the CDA, and the host information for CDA neighbors as parameters. CDA neighbor information also includes the type of relationship, which can be either peer-to-peer, provider-consumer or consumer-provider. The code is single-threaded and uses asynchronous network communication.

In this section, we describe the implementation and operation of Compass protocol from the perspective of a newly joining CDA $< A >$. We $< A >$ first starts, it binds the local IP address, UDP port and TCP port, which are provided as parameters. Next, it establishes TCP connections with its neighbors. Once the connections are successfully established, $< A >$ sends HELLO messages to all the

FIGURE 6.1. Hello Request

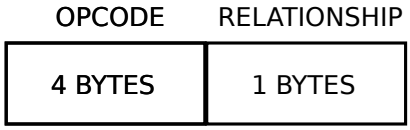


FIGURE 6.2. Hello Response

neighbors. The format of the HELLO message is shown in Figure 6.1.. Figure 6.1. shows that a HELLO message is 4 bytes long, and it only carries a 4 byte code that identifies a HELLO message. The purpose of a HELLO message is to establish that the CDA and its neighbor use the same protocol and that the TCP connection was not accidental.

Each neighbor of $< A >$ responds with a HELLO_RES message. The format of the HELLO_RES message is shown in Figure 6.2.. Similar to a HELLO message, a HELLO_RES message is also 4 bytes long, and carries a 4 byte code that identifies a HELLO_RES message.

Each CDA maintains three neighbor tables, one for each type of neighbor - i.e., peer, consumer, provider. Once HELLO and HELLO_RES messages are successfully exchanged between $< A >$ and its neighbors, $< A >$ adds neighbors in appropriate neighbor tables, and neighbors add $< A >$ in proper neighbor table.

CDAs also use HELLO and HELLO_RES messages as ping messages to check the liveliness of its neighbors. Every 60 seconds, a CDA sends HELLO messages to all its neighbors. If a neighbor fails to respond to 5 consecutive HELLO messages with

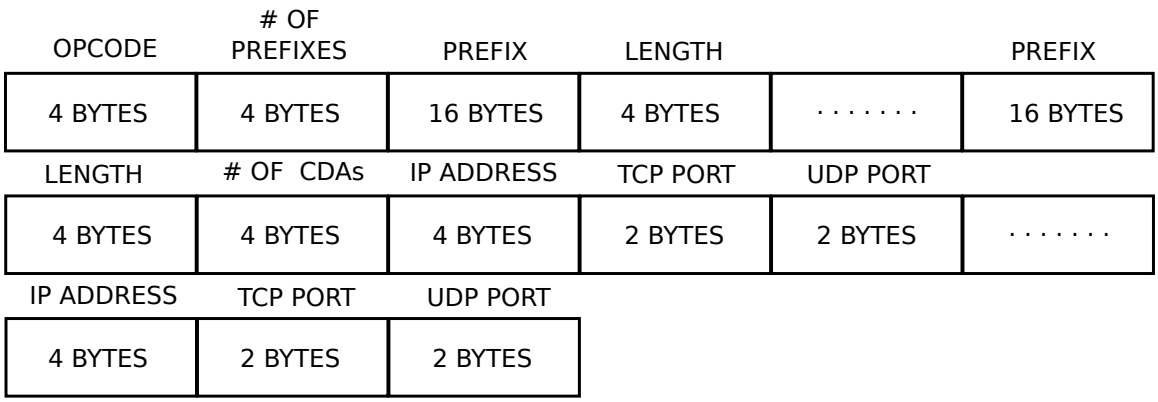


FIGURE 6.3. Announcement

a HELLO_RES message, then the CDA considers the neighbor dead, and discards it from the neighbor table.

Once a neighbor adds $< A >$ in its neighbor table, it sends an announcement message to the CDA. An announcement message from a neighbor to $< A >$ communicates all the prefixes that are reachable through the neighbor. Figure 6.3. shows the structure of the announcement message. The announcement message starts with a 4 byte number that identifies the announcement message. The next 4 byte number indicates the number of prefixes n that the message carries. After that there is a list of n prefix/length pairs; each prefix is 16 bytes long and each length is 4 bytes long. After the list of prefix/length pairs comes a 4 byte number that indicates the number of cda hops m through which the announcement message originally passed to reach $< A >$'s neighbor. Finally, the message contains a list of m IP/udp port/tcp port tuples. Each IP address is 4 bytes long and udp and tcp ports are 2 bytes long each.

When $< A >$ receives an announcement message from its neighbor, it extracts the prefix/length pairs and inserts the pairs in its global forwarding table. In our implementation, each CDA organizes its global forwarding table as a binary tree for

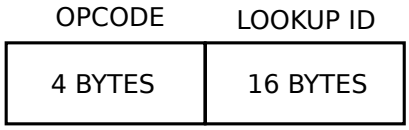


FIGURE 6.4. Lookup Request

high-speed longest prefix match. The CDA also extracts the CDA hops from the announcement message and stores the hops with the prefix in the global forwarding table.

Once a new CDA successfully establishes connections with its neighbors and receives prefix announcements from its neighbors, it inserts itself in the publication tree, as described in Section 5.3.4.. The first step in the insertion process is to perform a lookup for a random ID that contains the prefix that $< A >$ will later announce.

Our implementation supports UDP as well as TCP protocols for lookup. UDP-based lookup is fast, but unreliable. TCP-based lookup, on the other hand, is relatively slower but reliable. We believe that end hosts may use UDP for lookup, whereas CDAs may use TCP for lookup.

The structure of the lookup message is shown in Figure 6.4.. Similar to other messages, the lookup message also starts with a 4 byte number that identifies that lookup message. After that, the lookup message carries the 16 byte ID that the sender of the lookup message is querying.

We implement the lookup operation as an iterative process, so that the originator of the lookup request (either end host or a CDA has complete control over how the request proceeds further). In order to insert itself in the publish tree, $< A >$ starts the lookup process on a random ID that contains $< A >$'s prefix by performing a longest prefix match on its global forwarding table. The result of the longest prefix match on the global forwarding table is a prefix, an IP address, UDP port number,

and TCP port number of the next hop. Before sending a lookup message to its neighbor, $< A >$ checks whether it has already found an ancestor or a descendant in the publish tree. We refer to this test as the ancestor/descendant test. If the n bit long prefix of the neighboring CDA matches the n high order bits of the randomly chosen ID, then the neighboring CDA passes the ancestor/descendant (i.e., the neighboring CDA is either an ancestor or a descendant of $< A >$ in the publish tree), and the CDA does not perform any lookup. On the other hand, if the neighboring CDA fails the ancestor/descendant test, then the CDA starts the lookup process, by sending a lookup message to the neighboring CDA, using the existing TCP connection. The next hop neighbor performs a longest prefix match on its global forwarding table, and sends the next hop information to the CDA.

The format of the lookup response message is shown in Figure 6.5.. The first 4 bytes of the lookup response message contains a 4 byte number that identifies the lookup response message. The next 16 bytes contain the queried ID. Following the ID, the message contains a 4 byte length field, which indicates the length of the prefix of the next hop in bits. Following the length, the lookup response message reserves 16 bytes for the prefix. Finally, the lookup response message contains 8 bytes of host information, which includes 4 bytes of IP address, 2 bytes of TCP port number, and 2 bytes of UDP port number.

Once the CDA receives a lookup response message from its neighbor (or other hops that are farther away), it performs the ancestor/descendant test on the next hop. If the next hop CDA fails the test, then the new CDA establishes a TCP connection with the next hop, and sends the lookup message. This process continues, until $< A >$ finds a CDA $< Z >$ that passes the ancestor/descendant test. Once $< A >$ finds an ancestor or a descendant, it starts the insertion process.

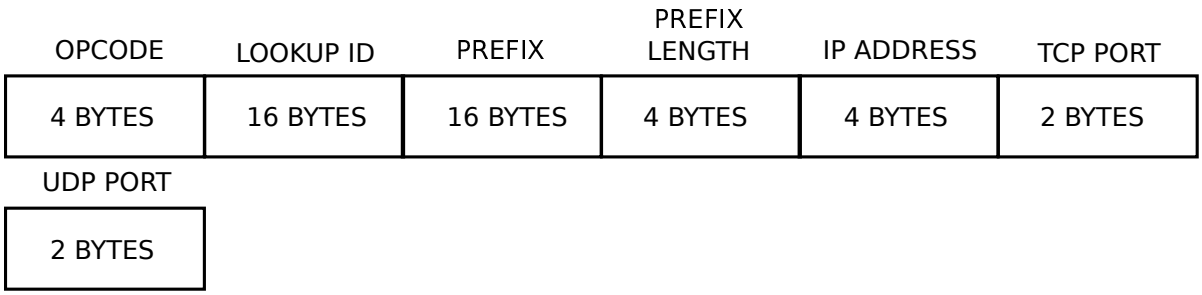


FIGURE 6.5. Lookup Response

As described in Section 5.3.4., $< A >$ further classifies the CDA $< Z >$ that passes the ancestor/descendant test into an ancestor, peer or descendant, depending on the following criteria:

- If $< Z >$'s announced prefix is smaller in length than new CDA's prefix, then $< Z >$ is new CDA's ancestor.
- If $< Z >$'s announced prefix is equal in length to new CDA's prefix, then $< Z >$ is new CDA's peer.
- If $< Z >$'s announced prefix is larger in length than new CDA's prefix then $< Z >$ is new CDA's descendant.

After classifying the CDA $< Z >$, $< A >$ establishes a TCP connection with $< Z >$, and sends a request for CDAs(RC) message. The structure of RC message is shown in Figure 6.6.. First four bytes of the RC message identify the RC message. Next one byte is a bitmask in which the lower order three bits are used and high order five bits are unused. If bit 0 is set, then the request is for $< Z >$'s parents, if bit 1 is set, then the request is for $< Z >$'s peers, and if bit 2 is set, then the request is for $< Z >$'s children. If multiple bits are set then the request is for multiple types

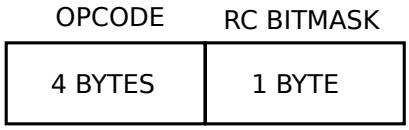


FIGURE 6.6. RC

of CDAs. For example, if bits 0 and 2 are set, then the request is for $< Z >$'s parents and children.

Depending on $< Z >$'s classification (i.e., ancestor, peer or descendant), $< A >$ sends an RC message with following bits set in the bitmask:

- If $< Z >$ is a descendant, then bit 0 is set in RC message, requesting $< Z >$'s immediate parents.
- If $< Z >$ is an ancestor, then bit 2 is set in RC message, requesting $< Z >$'s immediate children.
- If $< Z >$ is a peer, then all 3 bits are set, requesting $< Z >$'s children, peers and parents, because in case of a peer, $< A >$'s parent, peer and children tables will be same as $< Z >$'s tables.

Upon receiving RC, CDA $< Z >$ responds with CDAs from its publish neighbor table. The structure of the response message is shown in Figure 6.7.. The response message starts with a 4 byte code that identifies the response message. Next, the response message carries a 4 byte number that specifies the number of CDAs that are embedded in the response message. For each CDA, the response message carries a 4 byte number that indicates the length of the CDA prefix in bits, the CDA prefix, CDA IP address, TCP and UDP port numbers for the CDA, and 1 byte bitmask, which indicates $< A >$'s relationship with the CDA $< Z >$. As before, 5 high order

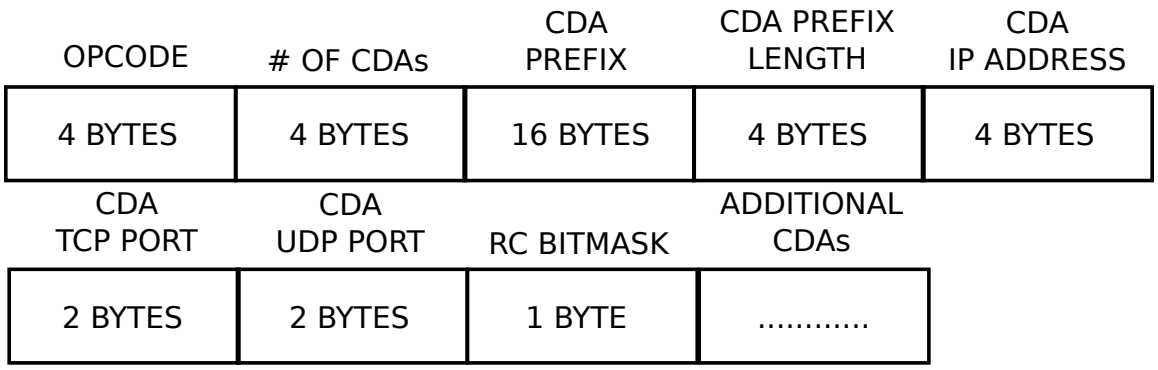


FIGURE 6.7. RC Response

bits in bitmask are unused, and lower order 3 bits indicate either parent (bit 0), peer (bit 1) and child (bit 2).

When the CDA $< A >$ receives an RC Response from a CDA $< Z >$, and if Z is either a descendant or an ancestor than $< A >$ randomly chooses one CDA and sends an RC message. The CDA $< A >$ continues this process to find its immediate parent. However, if $< Z >$ is a peer of $< A >$, then $< A >$ has already found its proper position in the tree, and does not need to search for its immediate parents, because $< Z >$'s parents are $< A >$'s parents. The CDA $< A >$ continues its search for immediate parent until it finds a CDA $< K >$ that satisfies the following two properties:

- $< K >$'s prefix length is smaller than A's.
- The difference between prefix lengths of A and K is the smallest across all the CDAs that A contacts.

At the end of this search, either $< A >$ finds its peer $< Z >$, or it finds its immediate parent $< K >$. In both the cases, $< A >$ sends an RC message to either $< Z >$ or $< K >$ with all 3 bits set, thus requesting their entire neighbor tables. However, it uses the responses different as follows:

– If $< A >$ contacts its peer $< Z >$, then $< Z >$'s neighbor table becomes $< A >$'s neighbor table - i.e., $< Z >$'s ancestors are $< A >$'s ancestors, $< Z >$'s peers are $< A >$'s peers, and $< Z >$'s descendants are $< A >$'s descendants.

– If $< A >$ contacts its immediate parent $< K >$, then $< K >$'s ancestors become $< A >$'s ancestors, $< K >$'s peers become $< A >$'s parents, and a subset of $< K >$'s descendants become $< A >$'s peers or descendants. All those descendants of $< K >$ that have the exact same prefix and prefix length as $< A >$, become $< A >$'s peers. All those descendants of $< K >$ whose prefix length is larger than $< A >$'s and whose high order n bits of the prefix match the n bit prefix of $< A >$, become $< A >$'s descendants.

It is certainly possible that A may fail to find a CDA K which satisfies the above two properties. If that is the case, then it means that A is the new root of the tree.

Once $< A >$ completes the process of learning about its immediate parents, ancestors, peers, immediate children, and descendants, it establishes a TCP connection with all of them and sends TREE_HELLO messages. The structure of the TREE_HELLO message is shown in Figure 6.10.. First 4 bytes of the TREE_HELLO message identifies the message. The next 4 bytes contain the length of the prefix announced by the new CDA in bits. The next 16 bytes contain the new CDA's prefix, and the following 8 bytes contain new CDAs IP address (4 bytes), UDP port number (2 bytes), and TCP port number (2 bytes). As CDAs in the publish tree receive TREE_HELLO message, they add the new CDA as an ancestor if the CDA's prefix length is smaller, as a peer if the CDA's prefix length is the same, and as a descendant if CDA's prefix length is larger.

Once $< A >$ has successfully inserted itself in the publish tree, it retrieves its share of resolution information from one of the following:

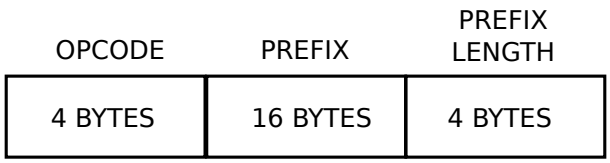


FIGURE 6.8. RR Message

– If $< A >$'s peers exist, then it can retrieve all the resolution information from one of its peers, because the peers carry the same resolution information as $< A >$.

– If no peer of $< A >$ exists, then $< A >$ retrieves the resolution information from its immediate parents and its immediate children.

In order to retrieve the resolution information from either its parents and children or its peers, $< A >$ establishes a TCP connection and sends an RR message. The structure of an RR message is shown in Figure 6.8.. First 4 bytes of the RR message identify the message. Next 16 bytes contain $< A >$'s announced prefix. Finally 4 bytes contain length of prefix in bits.

When CDA $< I >$ receives an RR message from $< A >$ it performs one of the following actions:

– If $< I >$ is $< A >$'s peer or child, then I sends its entire resolution table to $< A >$ in an RS message.

– If $< I >$ is $< A >$'s parent, then $< I >$ responds with only those resolution entries whose n high order bits match $< A >$'s n bit prefix. Our CDA implementation uses a hashtable to implement the resolution table. In order to find relevant resolution entries for $< A >$, $< I >$ iterates over its entire

resolution table and finds those entries whose n high order bits match $< A >$'s n bit prefix.

Implementing the resolution table as a hashtable is a design choice to support resolution in $O(1)$ time. Of course this design choice implies that the construction of an RS message requires $O(m)$ time, where m is the number of resolution table entries, because $< I >$ must iterate over the entire resolution table.

Another way to implement the resolution table is as a binary tree, in which case both the operations of resolution and RS message construction will take $log(m)$ time, where m is the number of entries in the resolution table. However, we believe that the resolution operation will be far more frequent than the construction of an RS message. Thus, it is important to perform resolution in constant time.

The structure of an RS message is shown in Figure 6.9.. The first 4 bytes identify the RS message. Next 4 bytes indicate the number n of resolution entries contained in the message. After that, the message contains n resolution entries. Each resolution entry consists of a 16 byte content ID, a 4 byte number m that indicates the number of content publishers that carry the content for the content ID, and m IP address (4 bytes), TCP port number (2 bytes) tuples.

After being completely inserted in the publish tree, the new CDA announces its prefix through its neighbors, using the announcement message shown in Figure 6.3..

When an end host conducts a lookup, it sends a lookup message (shown in Figure 6.4.) to its first hop CDA. The first hop CDA responds with a lookup response message (shown in Figure 6.5.), which contains information about the next hop. The end host then sends a lookup message to the next hop. This process continues until

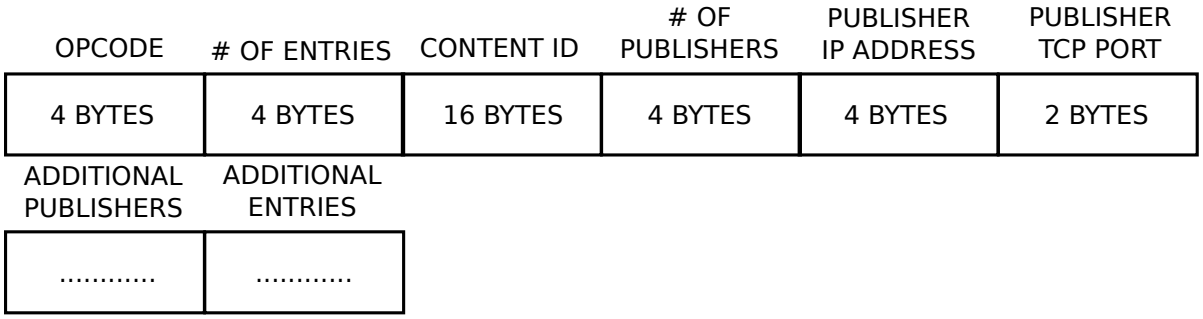


FIGURE 6.9. RS Message

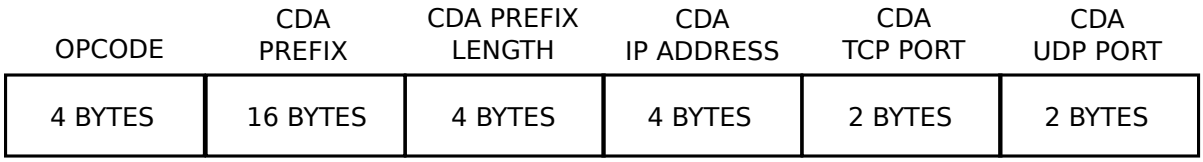


FIGURE 6.10. Tree_HELLO

the request reaches a CDA whose n bit prefix matches the n high order bits in the queried ID. If the last CDA possesses information about the queried ID, then it sends a LOOKUP_MAPPING message, which is shown in Figure 6.11.. Otherwise, it sends a LOOKUP_NULL message, which is shown in Figure 6.12..

As shown in Figure 6.11., LOOKUP_MAPPING message starts with 4 byte number that identifies the LOOKUP_MAPPING message. Next, it contains the 16 byte ID that the end host queried. After that, the message contains a 4 byte number, which indicates the number of publishers carried in the message. Finally, for each

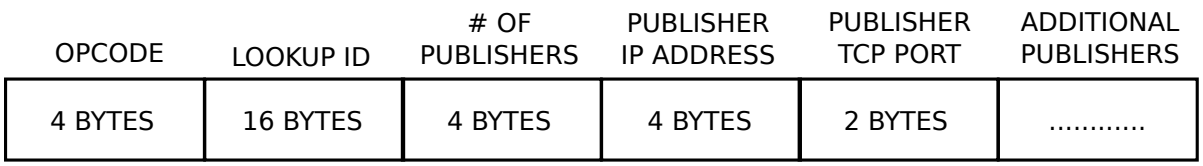


FIGURE 6.11. Lookup Mapping

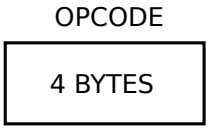


FIGURE 6.12. Lookup NULL

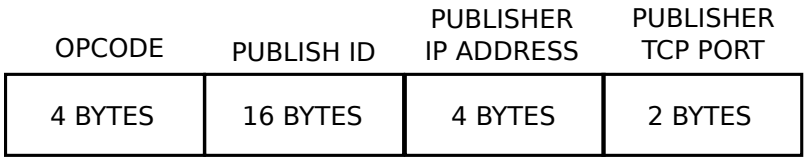


FIGURE 6.13. Publish

publisher, the message carries 6 bytes of information; 4 bytes for IP address and 2 bytes for TCP port number.

As the name indicates, LOOKUP_NULL message is an empty message that only carries a 4 byte code, which identifies the message.

When an end host wishes to publish the name to location mapping, it starts the publish process by performing a lookup. At the end of the lookup process, it first finds the first CDA that should store the name to location mapping. The next step in publish process is to send a publish message to the discovered CDA. The structure of the publish message is shown in Figure 6.13.

As shown in Figure 6.13., the publish message starts with a 4 byte number that identifies the publish message. Next, it contains the 16 byte ID, which is the hash of the content name. Finally, the message contains 6 byte information for the publisher, which includes 4 byte IP address and 2 byte port number.

The end host establishes a TCP connection with the discovered CDA, and sends the publish message. The discovered CDA propagates the publish message along the publish tree, as described in Section 5.3.7..

6.2. Evaluation Methodology

6.2.1. Metrics

We evaluate Compass in terms of *resolution latency*, *publication latency*, *storage cost*, and *networking cost*. We measure the latency as the number of CDA hops required to complete an operation. We focus on the latency of publication and resolution operations; we do not evaluate the latency of the announcement operation as this is similar to BGP, which is well-studied in the existing literature [113]. We measure the storage cost as the number of entries to be stored in the Resolution Forwarding Table (RFT), Resolution Table (RT), and Publish Neighbor Table (PNT), respectively. Finally, we measure the networking cost as the bandwidth required for publication operations.

6.2.2. Simulator, CDA Topology, and Realm Construction

We evaluate Compass using our own discrete event simulator. Our simulator implements Compass's operations (cf. Section 5.3.) and uses the real-world Internet topology from CAIDA [114] as the topology for the CDAs. We assume that each AS (autonomous system) in the CAIDA topology runs a CDA.

By considering how much a CDA participates in the resolution process, we investigate two scenarios for realm construction: *selfish* and *altruistic*. In the selfish scenario, a realm only consists of the root CDA that stores the entire RT. This scenario represents the worst case for Compass, where CDAs are not willing to share the resolution load. In the altruistic scenario, a CDA joins all the realms of its "provider CDA." The altruistic scenario represents a more realistic adoption of Compass in the current Internet topology.

We further need to select realm roots and assign content prefixes to CDAs in every realm. A realm root eventually stores the resolution information for all the content items in existence. Thus, CDAs with a large infrastructure should be more likely to become realm roots. We select as realm roots the CDAs with the highest ranking in the CAIDA topology (CAIDA ranks ASes by the size of the *customer cone* of every AS, where an AS's customer cone consist of its direct and indirect customers [115].) In our simulations, we specify the number of realms roots (and thus realms) as a fraction r of the total number of CDAs, where r varies from 0.001 to 0.01. Finally, we assign prefixes to CDAs by evenly dividing the content-ID space among realm members. We assume that 1 trillion content items have to be indexed and resolved, as this is Google's estimated size of the Web [2].

6.2.3. Measuring Resolution and Publication Latency

To evaluate resolution latency and publication latency, we select 10 CDAs as vantage points, i.e., local CDAs for resolution requests or publication requests. Since content consumers and providers are located at the edge of the Internet, we expect the majority of resolution and publication requests come from stub ASes, and we therefore select these 10 CDAs from stub ASes as well.

Every resolution simulation starts by building an RFT for one of the 10 local CDAs. Then for a random content-ID, the simulator can use the RFT to find the RFT tuple `<content prefix` P`, next hop CDA>` that is the longest prefix match for the content-ID. The simulator can then retrieve the CDA path corresponding to P, and use the number of hops in the path as the resolution latency for the content-ID in question.

Every publication simulation first organizes all the CDAs into the publication overlay based on their announced prefixes, and then propagates a publication message about a random content-ID from a local CDA. The message will reach all those CDAs whose prefix matches the random content-ID that is being published, as described in Sec. 5.3.7..

6.2.4. Measuring Networking Cost

The publication simulations maintain per-link counters to measure *link stress*: the number of times individual network links are used for publication operations. This metric allows us to quantify the per-link bandwidth requirements for publication operations. In order to compute link stress, the simulator uses the AS numbers of the ASes in which CDAs are situated: when CDA A forwards a publication message to a CDA B, the simulator uses C-BGP [116] to compute valley-free route from $A's$ AS to $B's$ AS. Each successive AS pair of the form (X,Y) in the above-mentioned valley-free route represents an AS-level link from X to Y.

We convert link stress for each link into bandwidth requirements by computing the following two numbers: 1) the maximum number of publication messages per link per publication simulation across all publication simulations for a given number of realms, and 2) the number of publication simulations for a given number of realms in which a given link was used, normalized by the total number of publication simulations for the given number of realms. We consider the first number as the worst case usage for a link, and the second number as the probability that a single link will be used during a publication session. Everyday approximately $150,000$ new domains are announced [101], and a single domain consists of $5,000$ pages on average [102], which results in 750 million publication announcement sessions per day. We use link

participation probability to calculate the fraction of publication sessions that will pass through a given link for a given number of realms. Next we multiply the total number of publication sessions that pass through a link by the worst case usage for a single publication session for a given number of realms, which gives the total number of publication packets that pass through a link. Finally, we multiply the total number of packets by publication packet size (i.e., 90 bytes, including payload and packet headers) to calculate the total traffic that passes through a link in 1 day.

6.2.5. Measuring Storage Cost

We focus on the three main data structures when measuring the storage cost of Compass:

RFT size—In order to evaluate RFT size, we construct RFTs of 10 CDAs whose corresponding ASes have the largest customer cones [114]. Similar to today's Internet, these CDAs have the largest RFTs.

RT size—In order to evaluate this metric, we consider the RT size of all CDAs. We compute the RT size for CDA A by dividing the total number of content items by the total number of CDAs in $A's$ realm.

PNT size—We evaluate PNT size by considering PNTs of all the CDAs. First, we use the announced prefixes to construct the publication overlay as described in Section 5.3.4.. Then, we compute the size of each CDA's PNT by summing up the number of its peers, descendants and ancestors.

6.3. Evaluation Results

This section describes the results from our simulations. Before we discuss results for our latency, storage and networking cost simulations, we first discuss two

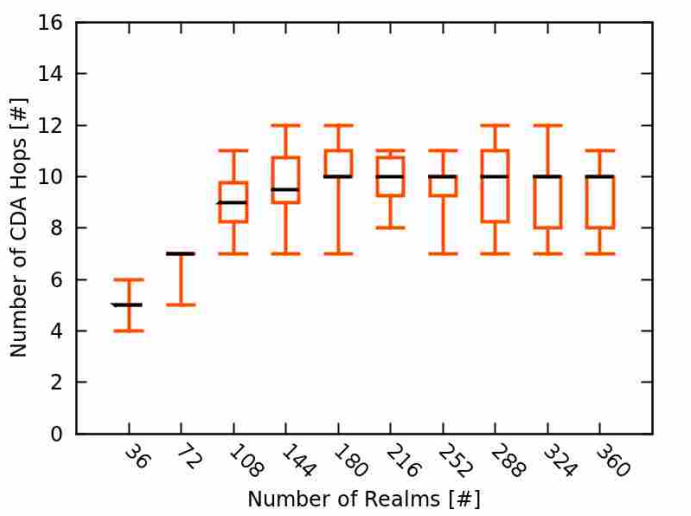
properties that emerge as a side-effect of our evaluation methodology. First, as the number of realms increases, the average size (i.e., number of CDAs per realms) of a realm decreases. For example, with 36 realms ($r = 0.001$), the average realm size is $21,000$ CDAs, and with 360 realms ($r = 0.01$), the average realm size is $1,200$ CDAs. This phenomenon is observed because when a CDA becomes a realm root, it cuts all ties with all other realms. Since the CDA does not belong to any other realm, its customers do not belong to other realms either, unless either a customer becomes a root itself, or the customer is multi-homed. Second, as the number of realms increases, the average prefix length of announced prefixes decreases. For example, with 36 realms, the average announced prefix length is 14 bits, and with 360 realms, the average announced prefix length is 11 bits. This phenomenon is observed because as the size of realms decreases, CDAs in the realms have increased responsibility for prefix announcements.

Next, we discuss the results for latency, storage and networking cost metrics.

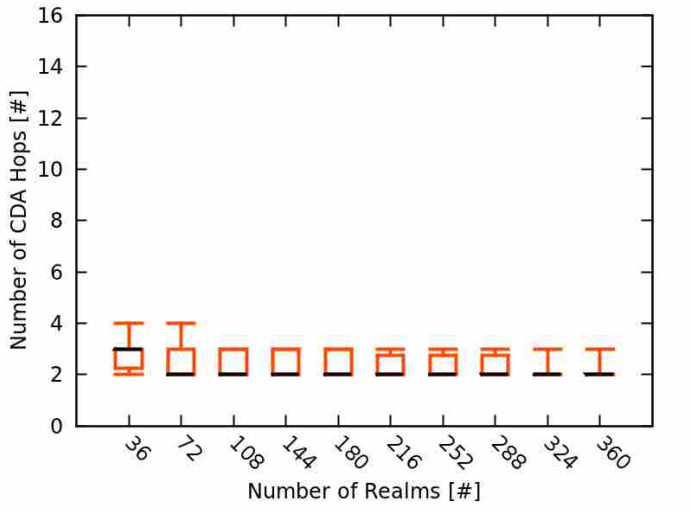
6.3.1. Latency

Figure 6.14. and Figure 6.15. show results for publication and resolution latency respectively for both altruistic and selfish scenario. Each subfigure in Figure 6.14. and Figure 6.15. shows the number of realms on the x-axis and number of CDA hops on the y-axis. Each subfigure also shows min, max, 25^{th}, 50^{th} and 75^{th} percentiles as box and whiskers plot, which represents variance in the number of CDA hops for a given number of realms.

Figure 6.14.a shows that in the altruistic scenario, as the number of realms increases, publication latency increases as well. This is because, the larger the number of realms, the larger the number of resolving-CDAs, which means the publication

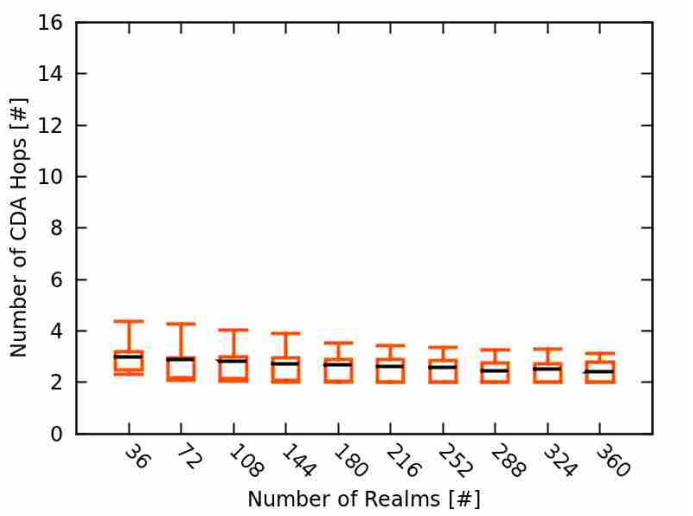


(a) Altruistic

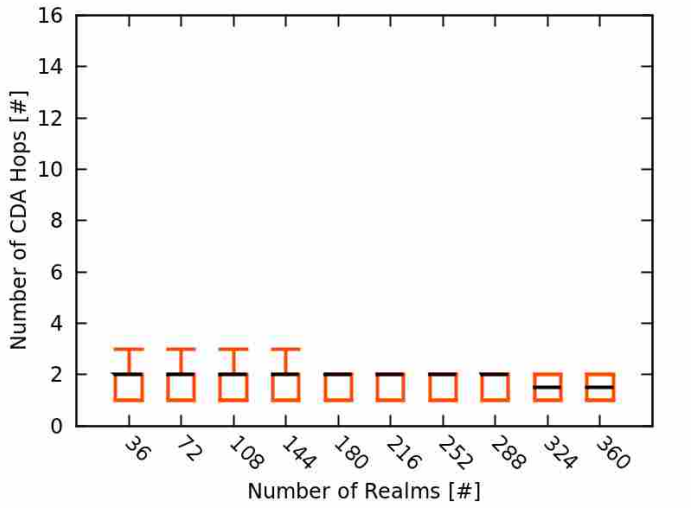


(b) Selfish

FIGURE 6.14. Publication Latency



(a) Altruistic



(b) Selfish

FIGURE 6.15. Resolution Latency

packets have to travel for more hops. Figure 6.14.b shows that in the selfish scenario the increase in number of realms does not have a significant impact on publication latency. This is because in selfish scenario, all the CDAs are peers. Once any CDA in the publication overlay is found, the other CDAs are contacted in parallel, which counts as one hop. Thus, the only variable is the number of hops required to find a CDA in the publication overlay, which is slightly higher for smaller numbers of realms but becomes constant as the number of realms increases.

We convert the CDA hop latency, shown in Figure 6.14.a, into time latency by using Padhye et al.'s [117] TCP throughput model to provide approximate numbers. Padhye et al.'s model takes three parameters as input: round trip time (RTT), probability of packet loss and maximum TCP window size. Since content providers and CDAs send only one packet to the next hop, we keep maximum window size constant and compute it as follows: $\frac{TCP\_DEFAULT\_WINDOW\_SIZE}{TCP\_MAXIMUM\_SEGMENT\_SIZE}$. On Linux distributions, $TCP\_DEFAULT\_WINDOW\_SIZE = 56KB$ and $TCP\_MAXIMUM\_SEGMENT\_SIZE = 1460B$. We vary the RTT value from 10 ms to 1,000 ms, based on recent measurements [118, 119]. Finally, we use three different values for loss probability, 0.001, 0.01 and 0.1, as used by Padhye et al. [117]. Using the TCP throughput model and the above-mentioned values, we generate several values of TCP throughput in packets/second. For example, in the best case (RTT=10 ms, packet loss probability=0.001), TCP throughput is $1,635$ packets/second or 0.61 milliseconds/packet. Figure 6.14.a shows that largest publication latency is 13 hops, which converts to 8.54 milliseconds. In the worst case (RTT = $1,000$ ms, packet loss probability=0.1), TCP throughput is 1.45 packets/second or 0.7 seconds/packet, which converts to 9.8 seconds for publication latency.

We compare the above-mentioned publication time latency numbers against CDNs' requirements to assess the feasibility of Compass. CDNs are the most frequent publishers because they repeatedly change the location of their content servers. Recent measurements [120] have shown that CDNs change location of their content servers every 100 seconds on average. Our numbers are well below what CDNs require.

Figure 6.15.a and Figure 6.15.b show that under altruistic and selfish scenarios respectively, the resolution latency decreases as the number of realms increases. This is because the increase in number of realms increases the number of CDAs that can resolve a given content-ID to IP address, which increases the probability of finding a CDA that is close-by.

Similar to publication latency, we convert resolution latency from hops to time. However, instead of varying RTT between 10 ms and $1,000$ ms, we fix it at 40 ms. For resolution operation, CDAs use their geographical neighbors. Recall that for our simulations, we use the CAIDA AS-level topology and replace each AS with a CDA. According to [108], worst case one way inter-AS latency is 20 ms. We use twice this value to convert into RTT. Using 40 ms as the RTT and 0.001, 0.01, 0.1 as values for loss probability, we get the following three values for TCP throughput in packets/second: 585.79(0.001), 80.4(0.01) and 2.94(0.1). When we multiply these values by the largest resolution latency in hops (i.e., 4.5), we get the following resolution latency values in milliseconds 7.68(0.001), 55(0.01) and $1,530$(0.1). In order to assess the feasibility of Compass, we compare the above-mentioned approximate numbers against DNS resolution latency, which is 382 ms on average [121]. These numbers show that Compass performs significantly better than DNS, as long as less than 10% of packets are lost.

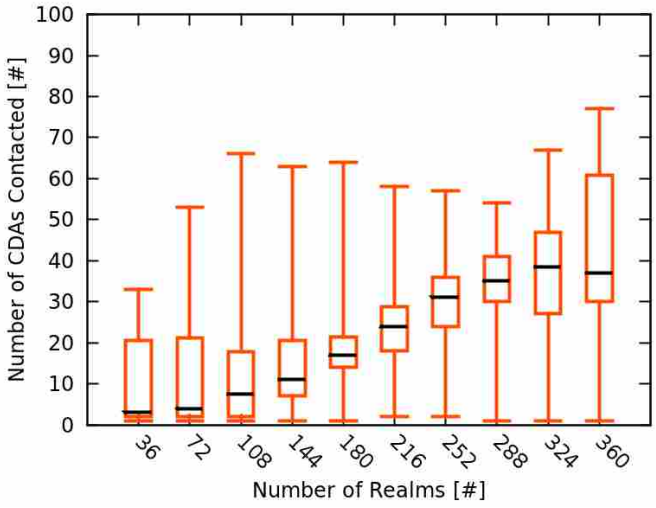
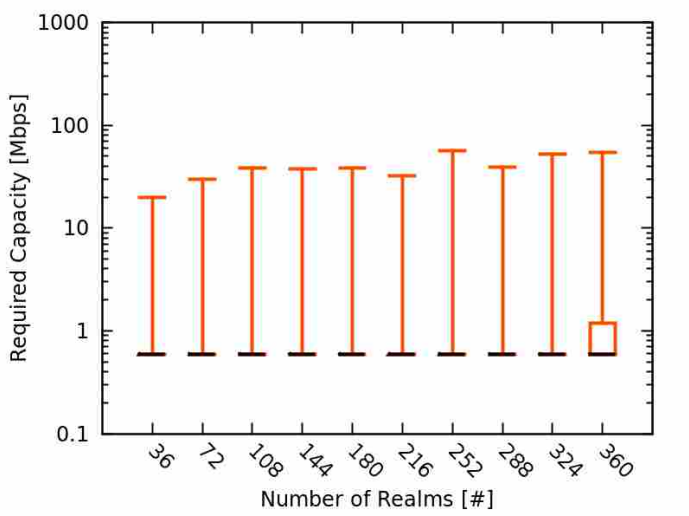


FIGURE 6.16. CDAs Contacted

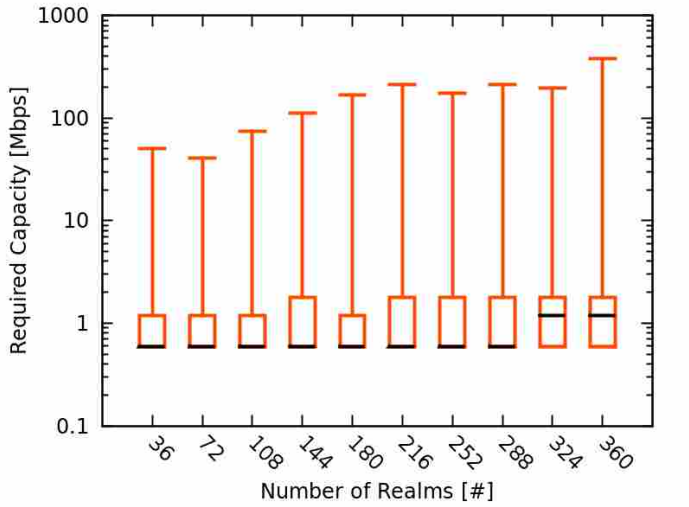
6.3.2. Networking Cost

In this subsection we evaluate networking cost associated with the publication operation.

We first evaluate whether today's outgoing link bandwidths per CDA can support the publication operation. Figure 6.16. shows the minimum, maximum, 25^{th}, 50^{th} and 75^{th} percentiles as box and whiskers plot, which represents variance in the number of CDAs that a single CDA contacts for a given number of realms. The x-axis in the figure shows the number of realms, and the y-axis shows the number of CDAs contacted by a single CDA. The figure shows that as the number of realms increases, the number of CDAs contacted increases as well. Recall that when propagating a publication message, a single CDA contacts only one of its parents and only one of its children, but contacts *all* of its peers. The variation in the number of CDAs contacted with increasing number of realms is because of the variation in the number of peers per CDA. As the number of realms increases, average size of realms and average

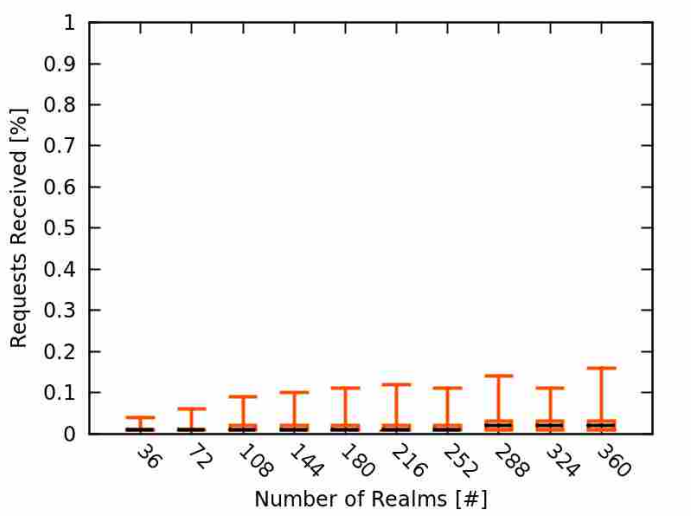


(a) Altruistic

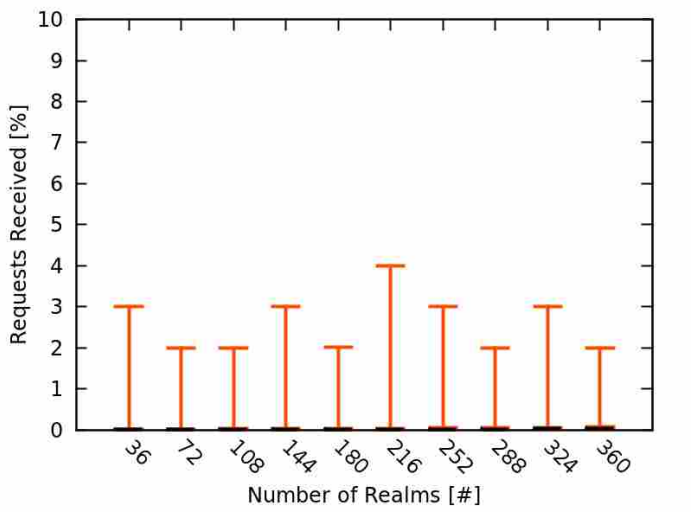


(b) Selfish

FIGURE 6.17. Bandwidth Consumption per Link



(a) Altruistic



(b) Selfish

FIGURE 6.18. Resolution Request Cost

length of announced prefixes decreases, which increases the similarity in prefixes and thus the number of CDA peers in the publication tree.

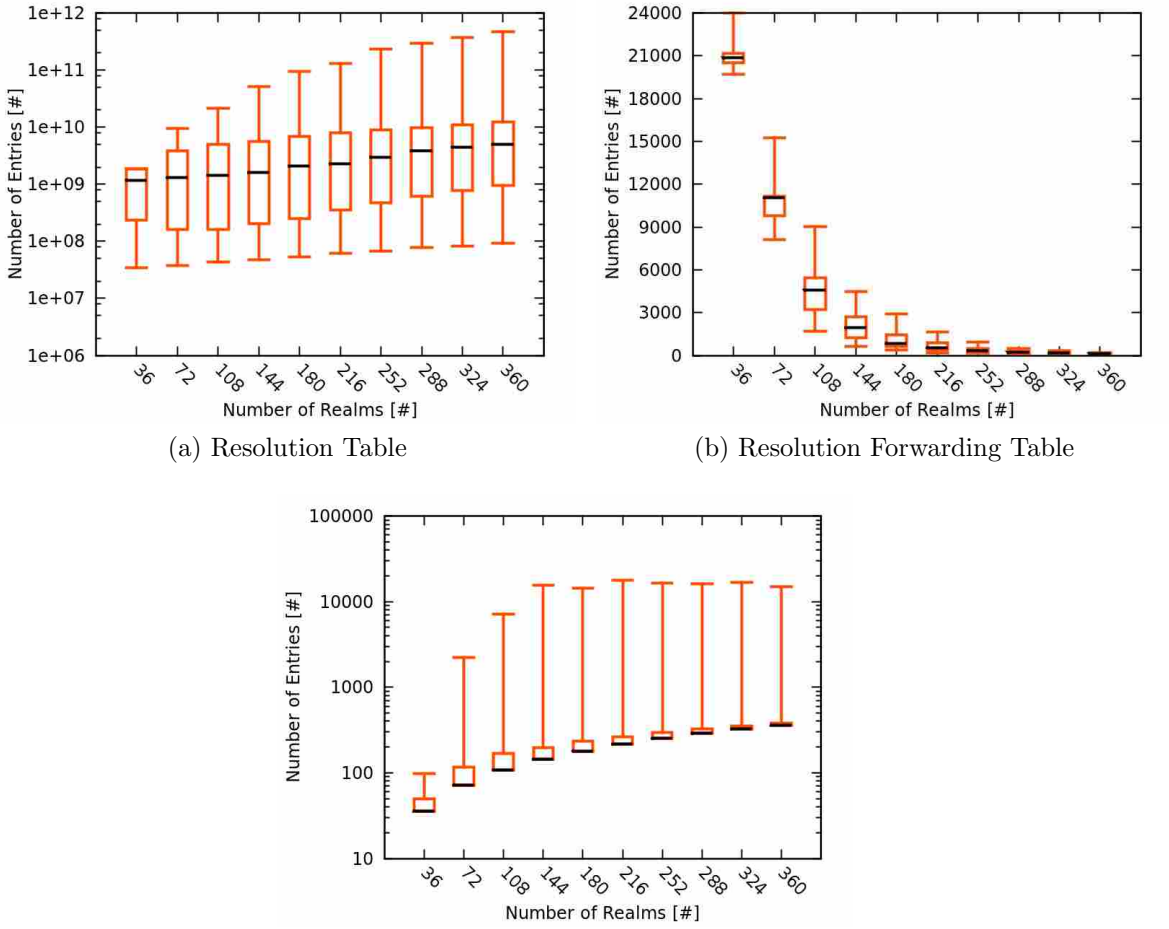
Figure 6.16. shows that at max, a CDA may contact 76 other CDAs. Each publication message is 90 bytes long (16 bytes for content-ID, 4 bytes for IP address, 2 bytes for TCP port number, 6 bytes for opcode that identifies publication message, and 62 bytes for packet overhead). In order to contact 76 CDAs in parallel, a single CDA's outgoing link must be able to transmit at $54Kbps$. Given that today's outgoing links can easily handle transmission rates on the order of several Mb/s, $54Kbps$ is a negligible requirement.

We assess whether today's network links possess enough bandwidth to carry publication traffic. The results are shown in Figure 6.17.a (altruistic scenario) and Figure 6.17.b (selfish scenario). The x-axis of the figure shows number of realms and the y-axis shows bandwidth requirement in Mbps using log-scale. Both the figures also show min, max, 25^{th}, 50^{th} and 75^{th} percentiles as as box and whiskers plot, which represents variance in bandwidth requirements for a given number of realms. Both the figures show little variance in bandwidth requirements, which implies that most links are used infrequently. Both the figures also show that for each realm setting (i.e., different number of realms), there are outlier links that observe large bandwidth requirement, and the bandwidth requirements increase with the number of realms - for the altruistic scenario the maximum capacity requirement is 54 Mbps, and for selfish scenario the maximum capacity requirement is 378 Mbps. As the number of realms increases, the number of resolving-CDAs increases as well, which increases the probability that a single link will be used multiple times, resulting in larger capacity requirements. While the capacity requirements for the altruistic scenario are manageable, the requirements are significantly high for the selfish scenario. In the

selfish scenario, the first CDA in the publication overlay that receives the publication message is responsible for contacting all other CDAs, because all the CDAs are peers. This process excessively uses the first CDA's outgoing links. Thus, the CDAs that announce 0x0/0 as their prefix must be aware that the bandwidth requirements are significantly high.

Finally, we measure the number of resolution requests that individual CDAs serve, which represents the cost incurred by each CDA. The results are shown in Figure 6.18.a (Altruistic scenario) and Figure 6.18.b (Selfish scenario). Both the figures show the number of realms on the x-axis and the number of resolution requests received by an individual CDA as a percentage of total resolution requests on the y-axis. Both the figures also show min, max, 25^{th}, 50^{th} and 75^{th} percentiles as as box and whiskers plot, which represent variance in percentage of received resolution requests for a given number of realms. Figure 6.18.a shows that as the number of realms increases, the percentage of received resolution requests increases as well. As the number of realms increases, average realm size and average announced prefix length decreases, and, as a result, individual CDAs receive more resolution traffic. Figure 6.18.a also shows that at max, an individual CDA receives 0.16% of total resolution requests, which is negligible. Figure 6.18.b does not show a particular pattern as the number of realms increases. This is because in the selfish scenario all the CDAs announce exactly the same prefix, and the resolution traffic received by a CDA depends on the origin of the request and the CDA's position in the topology. Figure 6.18.b shows at max a CDA receives 4% of total resolve requests, which is also a negligible number. To the best of our knowledge, no measurement study quantifies the total number of resolution requests in today's Internet, and, as a result, we cannot convert the above-mentioned percentages into exact resource requirements.

6.3.3. Storage



(a) Resolution Table

(b) Resolution Forwarding Table

(c) Publish Neighbor Table

FIGURE 6.19. Storage

Figure 6.19. shows results for resolution table (RT), resolution forwarding table (RFT) and publish neighbor table (PNT) in the altruistic scenario. We do not show results for selfish scenario, because in the selfish scenario each RT contains the entire content-ID space, each RFT has only one entry because only one prefix is announced, and each PNT contains all the other CDAs because all CDAs announce the same prefix. Figures 6.19.a, 6.19.b and 6.19.c show the number of realms on the x-axis and number of entries in respective tables on the y-axis. Each subfigure also shows

min, max, 25^{th}, 50^{th} and 75^{th} percentiles as box and whiskers plot, which represents variance in the number of table entries for a given number of realms.

Figure 6.19.a shows that as the number of realms increases, the size of the resolution table increases as well. The increase in the size of resolution table entries is because of the decrease in realm size - in smaller realms each CDA stores more resolution table entries. Figure 6.19.a further shows that at minimum a single CDA stores less than 100 million resolution table entries, and at maximum a single CDA stores approximately 500 billion resolution table entries. Each resolution table entry is 22 bytes long at minimum (16 bytes for content-ID, 4 bytes for IP address and 2 bytes for TCP port number). Assuming a machine with 32 GB of RAM, a CDA would require a cluster of approximately 344 machines to store 500 billion resolution table entries[1]. On the other hand, the CDA that stores less than 100 million entries requires only 2 GB RAM, which can easily be stored in one machine.

Figure 6.19.b shows that the number of RFT entries decreases as the number of realms increases. The decrease in number of RFT entries is because of the decrease in average realm size: CDAs announce lesser number of prefixes as average realm size decreases, which results in smaller RFTs.

Figure 6.19.b further shows that at max a CDA may store 24,000 entries in its RFT. Below we estimate the memory cost for storing the largest possible RFT in our simulations. Since we use the RFT for longest prefix matches, it is implemented as a binary tree. Assuming all the $24,000$ entries lie at the leaves of the binary tree, a 15 level binary tree would be enough to accommodate all $24,000$ entries. Each leaf of the RFT binary tree is 8 bytes long (4 bytes for IP address, 2 bytes for TCP port

[1]We realize that using a cluster of machines to store the resolution table in a distributed fashion is not straightforward. However, here our goal is to provide an estimate on infrastructure requirements to assess the feasibility of Compass.

number and 2 bytes for UDP port number). Each intermediate node of the RFT is also 8 bytes long, because it contains 2 pointers, one for each branch, and each pointer is 4 bytes long. The total number of intermediate entries that lie at levels 1-14 is 32766. Thus, the total number of nodes in the RFT is $65,534$, which require 512 KB of RAM. Given today's modern machines, 512 KB is negligible.

Figure 6.19.c shows that as the number of realms increases, the number of PNT entries increases as well. The increase in the number of PNT entries is because of the decrease in average prefix length: as CDAs announce shorter prefixes, they are placed higher in the publication overlay and have to store larger number of descendants. The figure shows that there is a fairly small variance, and majority of the nodes store less than 400 entries across all simulations. The figure further shows that, in our simulations, the largest number of PNT entries stored is $16,738$. Each PNT entry consists of the following fields: prefix announced (16 bytes), length of the announced prefix (4 bytes), IP address (4 bytes), TCP port number (2 bytes) and UDP port number (2 bytes), which totals at 28 bytes per entry. In the worst possible case of $16,738$ entries, the total table size in bytes is 458 KB. Given today's machines, 458 KB is a negligible memory requirement.

6.4. Results With Security

In this section, we revisit some of the metrics, which we evaluated in Section 6.3., and measure the overhead of our proposed security mechanisms. Specifically, we evaluate Compass in terms of number of entries stored in the Resolution Table (RT), and the bandwidth required for publish operations. We have chosen these metrics because these are the ones most affected by the addition of Compass's security

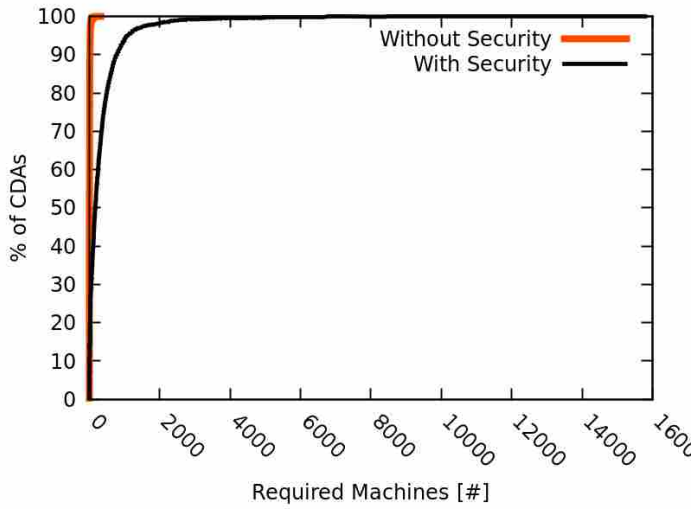


FIGURE 6.20. Resolution Table

mechanisms. In this section, we present results for $r = 0.01$ (i.e., number of realms) because it represents the worst possible scenario of security overhead.

Figure 6.20. shows results for resolution table (RT) for altruistic scenario. Figure 6.20. shows the number of machines required to store resolution table on x-axis, and the CDF on y-axis. The figure also shows two lines - one for resolution table storage cost with security mechanisms added and the other without security mechanisms added. As before, we calculate the number of machines assuming 32 GB of memory per machine.The figure shows that in the worst case, without security, a cluster of 344 machines is required to store the entire resolution table, and with security, in the worst case, $15,000$ (assuming 1 KB per table entry, adding extra length for the certificate) machines are required to store the resolution table. These numbers seem fairly high. However, these resources are required when CDAs announce 1 bit prefix and store half of content-ID space. If a CDA announces such a small prefix, then it must be prepared to contribute large amount of resources. Furthermore, we have reported these numbers assuming that CDAs will keep entire resolution table

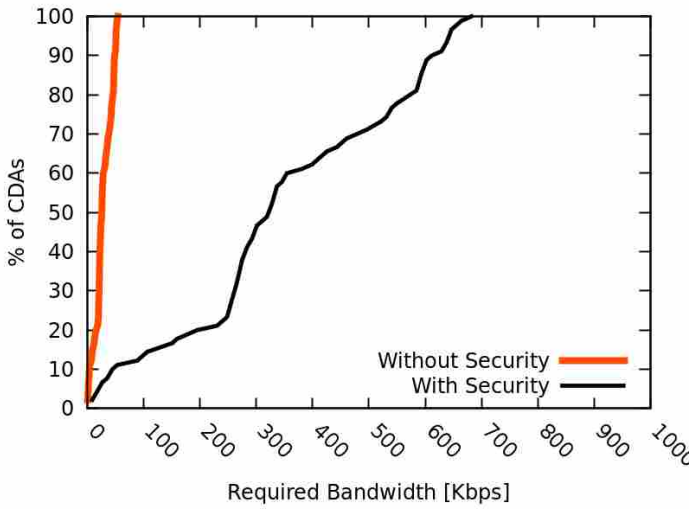
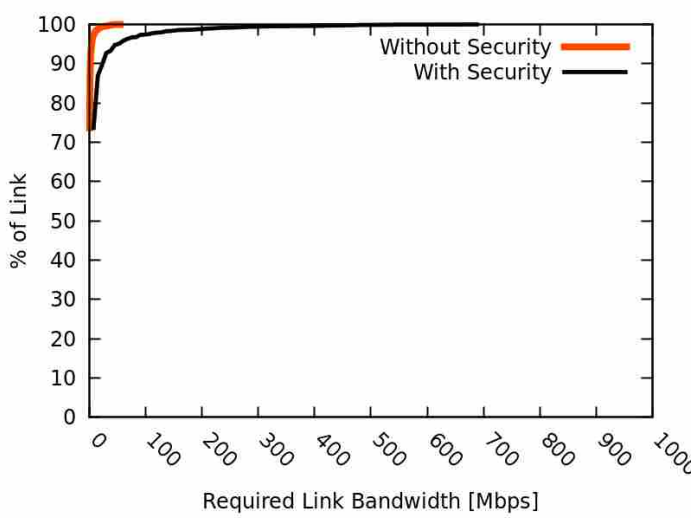


FIGURE 6.21. CDAs Contacted

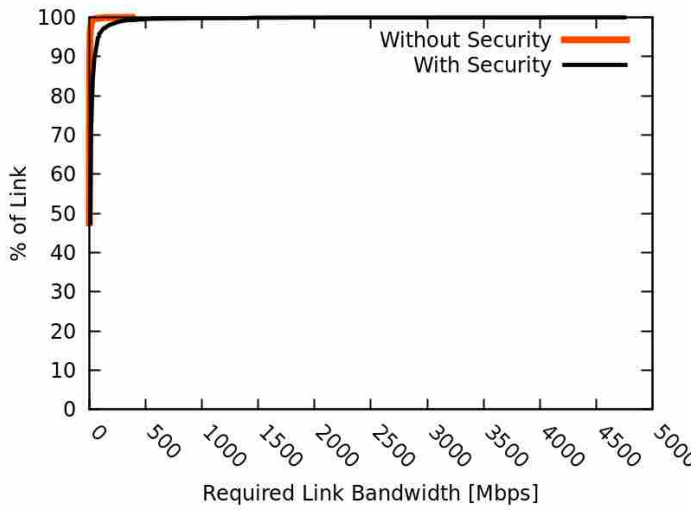
in the main memory. However, a CDA may choose to keep only popular resolution table entries in the main memory, and the remainder on the disk.

Next, we evaluate whether today's outgoing link bandwidths per CDA can support the publish operation. Figure 6.21. shows CDFs for publish operations with and without security mechanisms. The x-axis shows the required bandwidth and the y-axis shows the CDF. The figure shows that in the worst case, without security, a CDA only requires $54Kbps$ of outgoing bandwidth, which is easily manageable, given today's link bandwidths. The figure also shows that in the worst case, with security, a CDA requires approximately $682Kbps$ of outgoing bandwidth, which is also manageable, given today's multi Mbps links.

Finally, we assess whether today's network links possess enough bandwidth to carry publication traffic. The results are shown in Figure 6.22.a (altruistic scenario) and Figure 6.22.b (selfish scenario). The x-axis of the figure shows required link bandwidth and the y-axis shows the CDF for links. Both the figures show that for large number of links the bandwidth requirements are extremely low. However,



(a) Altruistic



(b) Selfish

FIGURE 6.22. Bandwith Consumption Per Link

some links do observe significantly high bandwidth requirements. For the altruistic scenario, small number of links require 54 Mbps without security, and 690 Mbps with security. Closer investigation shows that these links are between tier 2 ASes. Recent work on IXPs [122, 123] show that tier 2 ASes already exchange 10s of Gbps of traffic, and 690 Mbps is well within the range of these numbers. Furthermore, for the selfish scenario, small number of links require 378 Mbps without security, and 4.7 Gbps with security. As discussed in Section 6.3.2., these are the outgoing links of the source CDA. While 4.7 Gbps sounds very high, but the technology for 10 Gbps ethernet is available as a commodity, and can be easily employed. As before, the observation from this number is that if a CDA announces a prefix 0x0/0, which means it stores resolution information for all the content in existence, then it must be prepared to employ technology for high bandwidth.

6.5. Deployment of Compass

The content distribution ecosystem consists of the following members:

- Content Providers
- Content Distribution Networks (CDNs)
- Internet Service Providers (ISPs)
- Content Consumers

While the ecosystem itself is not changing, but the definition of content is continuously changing. Not so long ago, content distribution meant replicating static and dynamic web pages on CDN servers so that they were readily available to content consumers. However, recently that definition has shifted, and now majority of content

distribution consists of replicating and distributing video content. In 2010, Limelight showed that 51% of total Internet traffic in the U.S. consisted of video [124]. According to Sandvine's Global Internet Phenomenon report for 2013 [90], 68% of downstream peak traffic for North America consists of Real-Time entertainment traffic, and only 12.2% is web traffic. According the same report, Netflix [85] is the leader in generating video traffic with 32.3% share of downstream peak traffic, and Youtube [125] is the far second with 17.1% share of downstream peak traffic in North America. Finally, according to Cisco's Visual Networking Index (VNI) forecast [91], by the end of 2016, 86% of global consumer traffic will consist of video.

The above-mentioned numbers portray a pleasant picture for CDNs, who, despite decreasing their prices, are profitable [126]. These numbers also represent consumers' interest in video, who are free from scheduled programming, and have everything available on-demand. However, the numbers are troubling for content providers who have to foot the CDN bill, which increases with the amount of content that CDNs serve on content providers' behalf. Finally, the numbers are particularly worrisome for ISPs who have to upgrade their infrastructure to serve the increasing bandwidth demands, and also pay transit costs to bring the content to their consumers. We believe that content providers and ISPs can be mutually beneficial to each other in alleviating their video traffic problems, and Compass can play a key role in this process.

Content providers (e.g., Netflix [85], Hulu [86]) pay large amount of money to CDNs to deliver their content. For example, Netflix pays 6 cents per Gigabyte [127]. Furthermore, both Netflix and Hulu employ 3 CDNs [87, 88] instead of 1, so that if the performance of one CDN decreases other can be used immediately. Both Netflix and Hulu use fairly sophisticated architecture to use multiple CDNs [87, 88]. Netflix

is finally building its own CDN [128]. But, that is only because Netflix's traffic volume is higher than anyone else. Other content providers refrain from building their own CDNs because the entry barrier for deploying a global distributed system is very high, in terms of cost and engineering challenges. Even Netflix is still few years away from having a fully functional CDN [128]. Other content providers continue to pay large sums of money to CDNs, and they would certainly like to avoid large bills without building their own CDNs.

In order to keep up with video traffic, ISPs must add more bandwidth to their infrastructure, and bear the cost of transit traffic. To alleviate this problem, ISPs prefer that the content resides within their own network. In order to achieve this, ISPs may adopt two different approaches:

- Launch their own CDN (e.g., AT&T CDN [129])
- Use transparency caches [130]

In order to launch its own CDN, an ISP deploys content replication servers in its own network, and partner with content providers so that their data is available through ISPs content replication servers. However, for this strategy to work, an ISP must partner with all the major content providers, and the business dynamics may not always work out. The other option is to use transparent caching. As the name indicates, with transparent caching, an ISP maintains a content cache, and depending on caching policy, the content items are cached. Then the ISP examines every outgoing TCP packet, and in the case of HTTP, compares it against content items that exist in its cache. If the ISP finds a match, then it serves the content from its cache, else use the usual content retrieval process. However, examination of every outgoing TCP packet may add delay to the content retrieval process. Furthermore, if

the HTTP traffic is encrypted, then even if the content exists in ISPs cache, it cannot be served.

We believe that ISPs can be the early adapters of Compass, in order to enhance their approach towards transparent caches. With Compass, ISPs can check for a cache hit or a miss at the time of name resolution, instead of the time of the actual HTTP request. With DNS, it is not possible to discover a cache hit or a miss at the time of name resolution, because DNS packets do no carry full content names; they only carry domain names. More specifically, an ISP may run a CDA and become part of Compass. Next, it may publish the names and locations of cached content items in Compass. When a customer of Compass sends a name resolution request, using Compass the ISP can determine whether the queried content exists in its own cache, or the cache of its neighbor(s) or at the original content provider.

Once ISPs become part of Compass, then content providers also have an incentive to join Compass. Compass essentially allows ISPs to replicate and discover content in each others caches on content providers' behalf. This is beneficial for the ISPs because they will use the transit links only as last resort to retrieve content. This approach is also beneficial for the content providers because they will have to serve lesser and lesser content traffic. However, to enable this approach, content providers must publish content with persistent names in Compass, because without persistent names, the above-mentioned approach will not work.

Once content providers and ISPs adopt Compass, end users may adopt it as well. As mentioned above, video traffic is only going to increase as years go by. If end users can retrieve high quality video faster, which will be possible because of transparent caching, then it will be in their interest to adopt Compass.

Finally, the above-mentioned approach will drastically affect CDNs. As described in Chapter 5.2.1., CDNs consist of the following three components:

- Content Discovery
- Content Replication
- Monitoring

After adopting the above-mentioned approach, CDN's own content discovery service will be replaced by the Compass global content discovery service. Content will be replicated on-demand in ISPs' transparent caches. Finally, monitoring will no longer be necessary, because most of the time the ISP will serve the content from its own cache within the network, and ISPs already possess extensive knowledge of their networks because of traffic engineering [131]. Thus, adoption of Compass along with caching inside ISPs may emerge as a competitive threat for CDN businesses.

6.5.1. Coexistence With DNS and CDNs

In order to be deployed, Compass does not require elimination of DNS. However, in order to coexist with DNS, there must be a distinction between Compass names and DNS names. To establish that distinction, we expect that content providers will add the prefix "compass://" to the content names to distinguish them from DNS names.

6.5.2. Path to Adoption of Compass

In order for Compass to be adopted, several entities will have to play their part, as described below:

6.5.2.1. Name Discovery

As described in Section 5.3.1., Compass allows arbitrary human-readable names for content discovery. Similar to today's domain names, Compass content names need a means to be discovered. Compass content names can exchanged via digital communication (e.g., email) or word of mouth.

Compass content names can also be discovered via search engines. Similar to DNS names, content documents in Compass can also be connected via hyper-references. Thus, search engines can use their existing crawl approaches to discover and index content. Furthermore, use of persistent names may allow search engines to group similar content, which is not possible with today's domain-based names. For example, www.youtube.com/cricket_worldcup and www.dailymotion.com/cricket_worldcup maybe names of exactly same content. But, these names cannot be grouped together and presented as one because of presence of domain names.

6.5.2.2. Content Discovery Agents (CDAs)

As discussed in Section 5.3., all participants of Compass must run CDAs. Thus, ISPs, organizations and content providers must run CDAs. Furthermore, all the participants must form business relationships and realms to agree upon policy.

6.5.2.3. End Hosts

The operating system at the end host must be patched to add a new system call *gethostbyname_compass*, which will implement the Compass protocol at the end host. Furthermore, any application (e.g., a web browser) that wishes to use Compass

must be patched so it can use gethostbyname_compass system call, and retrieve IP addresses via Compass.

6.6. Open Issues

In Chapter V and Chapter VI, we have designed, implemented and evaluated Compass. Just like any other work, there are still open issues, which if addressed, can further improve the Compass design.

The first and foremost issue is the grouping of content-IDs. Compass uses flat, non-hierarchical, fixed-length hashes of names as content-IDs. While this approach puts a bound on the content namespace (Section 5.3.1.), it eliminates the possibility of grouping content names together. For example, there is no way for a CDA to announce a prefix such that it can only store the resolution information for a particular content provider (e.g., google), or for a particular content type (e.g., video) e.t.c. DONA [69] allows flat content names to be bound to flat content provider names. But, this approach violates the requirement for persistent names, as described in Section 5.2.2.3.. NDN [71] allows a natural grouping of names by announcing prefixes of human-readable names. However, the use of human-readable names violates the scalability requirement. Thus, the issue of grouping flat content-IDs remains open.

Another open issue is that currently, the security mechanism for prefix announcement in Compass relies on accountability - i.e., if a CDA does not resolve content-IDs for its announced prefix, then it will be held accountable. As described in Section 5.4.2., this approach is different BGPSEC, which relies on authorization - i.e., an AS cannot announce a prefix, unless it is authorized to do so. While authorization is a stronger property than accountability, BGPSEC requires the deployment of a centrally-controlled public-key infrastructure, which has been the major hurdle in

the adoption of BGPSEC. Compass, on the other hand, only require autonomous certificate authorities, which already exist today. However, it is still an open issue that whether the flexibility of Compass (i.e., use of certificate authorities) justifies it weaker security guarantee.

Compass also requires more work in understanding the CDA economy and their incentives. Specifically, to facilitate the deployment of Compass, it must be understood what are the incentives for a CDA to announce a prefix, to start a new realm, to join an existing realm and to not announce a prefix at all. Furthermore, we also must understand the nature of agreements between CDAs.

Compass evaluation results show that the deployment of Compass is feasible, given today's technology. However, the results can still be improved. For example, if a CDA announces 1 bit prefix, then it requires 344 machines without security overhead, and $15,000$ machines with security mechanisms enabled. However, this resource requirement can be decreased if the CDAs do not store everything in the main memory of the machine. Instead, if CDAs store only popular content-IDs in the main memory (i.e., cache), and unpopular content-IDs on disk, then this resource usage can be decreased significantly. Furthermore, Compass also imposes excessive bandwidth overhead, which is within today's bandwidth usage limits. The primary reason for excessive bandwidth usage is that the certificates are carried inside the publication packets. Thus, it remains an open issue whether an out-of-band certificate distribution mechanism or self-certifying names [132] can solve this problem.

6.7. Recap

In this chapter we evaluated Compass using simulations over a real-world Internet topology. We explored the impact of Compass deployment on CDAs in terms

of storage, latency and networking cost, and found that Compass is feasible for deployment. In addition, we also discussed how Compass can be deployed in real-world, and open issues that should be addressed.

CHAPTER VII

CONCLUSION

Content discovery is a challenging problem faced by today's Internet. The only globally deployed content discovery service is domain name system (DNS), and it only discovers domain names, instead of content names. Content names are persistent, and does not change when content moves from one location to another. Today's domain names, however, change as content moves. After the invention of DNS, content distribution networks (CDNs) came into being to augment DNS with location awareness. However, CDNs do not address content names either, and continue addressing domain names. During the last few years, the research community has proposed several infrastructure level solutions to address the problem of content discovery. However, these new solutions do not respect the autonomy of participants of the system, and cannot be incrementally deployed.

In this dissertation, we demonstrated that a peer-to-peer (P2P) network is an ideal candidate for building an Internet-scale content discovery service, provided certain weaknesses are addressed. Specifically, we addressed lack of heterogeneity, stretch, membership management, and management of churn in P2P networks. Furthermore, use of P2P networks gave us the advantages of scalability and incremental deployment as well.

In the remainder of this chapter, we first summarize our research, then discuss additional thoughts, and finally describe future work that can be done in this direction.

7.1. Summary

In this dissertation, we first presented Montra, a technique for capturing DHT traffic, in order to understand the characteristics and weaknesses of structured P2P systems. Montra captures DHT traffic without disrupting the network. The cornerstone of Montra is Minimally Visible Monitors (MVMs). MVMs resolve the following dilemma: a small number of monitors cannot capture a comprehensive view of the traffic and a large number of monitors may cause disruption. MVMs, because of their limited visibility, can be deployed in large numbers without changing or disrupting the system.

We validated Montra using measurements on Kad and Azureus networks. By comparing Montra's observations with those of instrumented source and instrumented destination peers, we established that Montra can accurately capture 90% of query traffic. Using Montra, we present several example observations of destination traffic in Kad and Azureus, based on simultaneously monitoring approximately 32,000 peers. The data includes an examination of file and keyword popularity, how popularity changes over time, file size, and geographic distribution.

To further demonstrate the weaknesses of P2P systems, we identified and created a new type of botnet, called *Tsunami*. Tsunami bots establish a parasitic relationship with a widely deployed DHT. Tsunami can successfully issue commands to its bots, launch various attacks, including distributed denial of service (DDoS) and spam, at ease, as well as receive responses from the bots, all without relying on any kind of central server or bootstrap nodes as current botnets do. The primary cause behind Tsunami's successful operation is the lack of membership management in widely deployed P2P systems.

Tsunami is not only theoretically feasible, but also operational in the real-world. In a Kad network with four million nodes and 6% of them being embedded Tsunami bots, it can reach 75% of its bots in about 240 seconds. Meanwhile, defending a system such as Kad against Tsunami necessitate a systematic approach. A solid defense must be able to effectively intercept Tsunami traffic with low cost, and do so without being intrusive to the system in question.

Finally, in this dissertation, we address the weaknesses of P2P systems, and design a new P2P content discovery service for the Internet, called Compass, which can fulfill all the requirements of a content discovery system.

Compass resolves content identifiers (cryptographic hashes of human readable content names) into locations (i.e., IP addresses). Compass does not impose any restrictions on the structure of human-readable content names, which implies that the names are not bound to domains and remain persistent. This approach solves the problem of existing globally deployed content discovery systems - i.e., DNS and CDNs.

Compass is secure because it maintains tight control over who can join the system. Specifically, a new content discovery agent (CDA) cannot join the system until an existing CDA allows the entry. Furthermore, existing CDAs are accountable to the CDAs that allowed their entry in the system. This chain of responsibility constitutes the entry mechanism for Compass. This approach is different from P2P's content discovery, in which any new host can join the network and can fatally destroy it.

Compass allows CDAs to decide the extent of their participation in storing content-ID resolution information. CDAs can store resolution information depending upon the capacity of their infrastructure. A CDA that stores content-ID resolution

information announces a content-ID prefix, which indicates the range and amount of stored resolution information. Compass introduces the concept of a *realm*, which is a collection of neighboring CDAs that can collectively resolve the entire content ID space. Realms allow a collection of CDAs to decrease the resolution latency for their customers, while keeping resolution state feasible. These approaches make Compass scalable, which is different from existing information-centric network designs. ICN designs either do not respect the autonomy of ASes (e.g., DONA, PSIRP), or use human-readable names inside routers (e.g., NDN, TRIAD). Both the approaches severely affect the scalability of ICNs.

7.2. Additional Thoughts

In addition to above-mentioned contributions, I also learned several lessons. Some of the most important lessons are highlighted in this section.

The first and foremost lesson I learned was that when dealing with a real-world system, it is extremely important to understand the actual implementation of a protocol. I learned this lesson during our work on Montra. While working on Montra, I read through large pieces of eMule [38], aMule [133] and Vuze [134] source codes. I found that even though all the three P2P clients implement the Kad protocol, but they make their own variations to suit the needs of their own application. Thus, reading the paper only gives a high-level view of the protocol. It is extremely important to be able to understand others' source code, and connect it with the high-level description of the protocol from the research paper. It is because of the understanding of how the source code worked, I was able to conceive the idea of minimally visible monitors.

I also learned the importance of applying knowledge across fields. At the time when I read the CCNx paper [71], I had accumulated significant knowledge about

P2P networks. I saw the CCNx work as an example of unstructured P2P network, in which content discovery requests were flooded. Since the next step after unstructured networks in the evolution of P2P networks was structured P2P networks, we applied the same thinking to CCNx. This way of thinking was the basis for our Compass work.

While working on Compass, I learned that naming is one of the most critical components in the design of an Internet architecture, and it is extremely hard to get it right. For example, designers of DNS did their best to design an appropriate naming scheme. However, we have seen that the DNS way of naming content is a deterrent to persistent names, and can lead to tussle [105]. Thus, one must be extremely careful when designing the naming scheme for an Internet architecture.

Finally, while working on Compass, I learned the importance of a properly decoupled architecture that assigns proper responsibilities. For example, all the ICN approaches require that autonomous systems (AS) should be involved in the content discovery process. We believe that it is an additional responsibility that does not belong there. An AS's responsibility is to carry traffic, and it should not be *required* to participate in the content discovery process. Therefore, for our Compass work, we created content distribution agents (CDAs), which can be owned by any entity (e.g., ISP, organization) that is interested in content discovery.

7.3. Future Work

For future work, I would like to make efforts to deploy Compass in real-world. In doing so, I would serve college campuses as examples of running Compass to commercial ISPs and content providers. If Compass proves to be beneficial to college campuses, then slowly commercial ISPs and content providers may adopt it as well.

For example, the University of Oregon (UO), Oregon State University (OSU) and Portland State University (PSU) may collectively form one Compass realm. In the beginning Compass may only be used to serve video traffic, as discussed in Chapter VI. In order to serve video traffic cheaply (i.e., without using transit links), UO, OSU and PSU may deploy transparency caches, as described in Chapter VI. All the three campuses must also deploy Compass CDAs. In addition to implementing the Compass protocol, the Compass CDAs must also implement the DNS protocol. Furthermore, all the three campuses may modify their campus machines as follows:

- Patch the operating systems so that they support gethostbyname_compass system calls.
- Patch the browsers so that for a pre-defined list of video websites (e.g., Netflix, Youtube, Hulu), the browsers use Compass instead of DNS.

When a user requests a video from a pre-defined list, the web browser hashes the entire name, and sends a resolution request to its first-hop Compass CDA, using gethostbyname_compass system call. The hashes include the domain name as well, which voids the benefit of persistent names. However, until Compass attracts the cooperation of content providers, this is the best that the initial deployment can do. The Compass CDA uses the Compass protocol to resolve name of video into location. If a match is found, then it means the video exists in one of transparency caches, and can be served from there. Otherwise, the Compass CDA uses DNS to find the location of video. As the video is downloaded, it maybe cached according to the caching policy. If any of the campuses cache the video, then it publishes the location of the video using Compass.

Using, the above-mentioned approach, if the three campuses can demonstrate a decreases in their transit bandwidth usage cost, then other commercial ISPs may becomes interested because they would like to decrease their transit bandwidth cost as well. Furthermore, the above-mentioned approach may attract content providers as well, because they have to pay large sums of money to content distribution networks (CDNs), based on the bandwidth used. If an ISP can serve a content provider's content locally, then it can be a win-win situation for content providers and ISPs.

REFERENCES CITED

[1] G. Memon, "Characterizing traffic in widely-deployed dht," Tech. Rep. CIS-TR-2009-01, 2008.

[2] "We knew the web was big..." Blog, 2008, http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html.

[3] D. Stutzbach, R. Rejaie, and S. Sen, "Characterizing unstructured overlay topologies in modern p2p file-sharing systems," *Networking, IEEE/ACM Transactions on*, vol. 16, no. 2, pp. 267–280, 2008.

[4] D. Stutzbach and R. Rejaie, "Capturing accurate snapshots of the gnutella network," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 4. IEEE, 2005, pp. 2825–2830.

[5] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger, "On unbiased sampling for unstructured peer-to-peer networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 2, pp. 377–390, 2009.

[6] D. Stutzbach and R. Rejaie, "Characterizing the two-tier gnutella topology," *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1, pp. 402–403, 2005.

[7] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications.* New York, NY, USA: ACM, 2001, pp. 149–160.

[8] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems.* London, UK: Springer-Verlag, 2002, pp. 53–65.

[9] J. Li, J. Stribling, R. Morris, M. Kaashoek, and T. Gil, "A performance vs. cost framework for evaluating dht design tradeoffs under churn," *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 1, pp. 225–236 vol. 1, March 2005.

[10] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The impact of dht routing geometry on resilience and proximity," in *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2003, pp. 381–394.

[11] M. Steiner, T. En-Najjary, and E. W. Biersack, "A global view of kad," in *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2007, pp. 117–122.

[12] M. Rabinovich and O. Spatscheck, "Web caching and replication," *Sigmod Record*, vol. 32, no. 4, p. 107, 2003.

[13] S. Sivasubramanian, M. Szymaniak, G. Pierre, and M. v. Steen, "Replication for web hosting systems," *ACM Computing Surveys (CSUR)*, vol. 36, no. 3, pp. 291–334, 2004.

[14] X. Zhang, W. Wang, X. Tan, and Y. Zhu, "Data replication at web proxies in content distribution network," in *Web Technologies and Applications*. Springer, 2003, pp. 560–569.

[15] M. Sayal, Y. Breitbart, P. Scheuermann, and R. Vingralek, "Selection algorithms for replicated web servers," *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 3, pp. 44–50, 1998.

[16] "Domain names - implementation and specification," RFC, 1987, http://www.faqs.org/rfcs/rfc1035.html.

[17] G. Memon, R. Rejaie, Y. Guo, and D. Stutzbach, "Large-scale monitoring of dht traffic," in *International Workshop on Peer-to-Peer Systems (IPTPS), Boston, MA*, 2009.

[18] Memon, Rejaie, Guo, and Stutzbach, "Montra: A large-scale dht traffic monitor," *Computer Networks*, vol. 56, no. 3, pp. 1080–1091, 2012.

[19] G. Memon, J. Li, and R. Rejaie, "Tsunami: A parasitic, indestructible botnet on kad," *Peer-to-Peer Networking and Applications*, 2013.

[20] G. Memon and J. Li, "Survey of future internet architectures," Tech. Rep. CIS-TR-2011-03, 2011.

[21] G. Memon, J. Li, and M. Varvello, "Compass: A content discovery service on the internet," Tech. Rep. CIS-TR-2013-14, 2013.

[22] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2001, pp. 161–172.

[23] P. Druschel and A. Rowstron, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," *Submission to ACM SIGCOMM*, 2001.

[24] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *Selected Areas in Communications, IEEE Journal on*, vol. 22, no. 1, pp. 41–53, 2004.

[25] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2001, pp. 149–160.

[26] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," *Peer-to-Peer Systems*, pp. 53–65, 2002.

[27] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*. IEEE, 2001, pp. 99–100.

[28] Y. Qiao and F. E. Bustamante, "Structured and unstructured overlays under the microscope," *USENIX Annual Technical Confrence*, 2006.

[29] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*. London, UK: Springer-Verlag, 2001, pp. 329–350.

[30] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and," Berkeley, CA, USA, Tech. Rep., 2001.

[31] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2001, pp. 161–172.

[32] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The impact of dht routing geometry on resilience and proximity," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, 2003, pp. 381–394.

[33] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "Opendht: a public dht service and its uses," in *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2005, pp. 73–84.

[34] J. Li, J. Stribling, R. Morris, and M. F. Kaashoek, "Bandwidth-efficient management of dht routing tables," in *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2005, pp. 99–114.

[35] "Emulab," Website, http://www.emulab.net/.

[36] "Nat," Website, http://en.wikipedia.org/wiki/Network_address_translation.

[37] "Sha-1," Website, May 2012, http://en.wikipedia.org/wiki/SHA-1.

[38] "emule," Website, http://www.emule-project.net.

[39] "Azureus," Website, Nov. 2011, http://azureus.sourceforge.net.

[40] J. Falkner, M. Piatek, J. P. John, A. Krishnamurthy, and T. Anderson, "Profiling a million user dht," in *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2007, pp. 129–134.

[41] Y. Qiao and F. E. Bustamante, "Structured and unstructured overlays under the microscope: a measurement-based view of two p2p systems that people use," in *ATEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2006, pp. 31–31.

[42] M. Steiner, W. Effelsberg, T. En Najjary, and E. W. Biersack, "Load reduction in the KAD peer-to-peer system," in *DBISP2P 2007, 5th International Workshop on Databases, Information Systems and Peer-to-Peer Computing, September, 24, 2007, Vienna, Austria*, Sep 2007.

[43] D. Stutzbach and R. Rejaie, "Improving lookup performance over a widely-deployed dht," *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pp. 1–12, April 2006.

[44] ——, "Capturing Accurate Snapshots of the Gnutella Network," in *Global Internet Symposium*, Miami, FL, Mar. 2005, pp. 127–132. [Online]. Available: http://www.barsoom.org/~agthorr/papers/gi05.pdf

[45] A. Klemm, C. Lindemann, M. K. Vernon, and O. P. Waldhorst, "Characterizing the query behavior in peer-to-peer file sharing systems," in *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2004, pp. 55–67.

[46] B. Krishnamurthy, J. Wang, and Y. Xie, "Early measurements of a cluster-based architecture for p2p systems," in *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*. New York, NY, USA: ACM, 2001, pp. 105–109.

[47] M. Ripeanu, A. Iamnitchi, and I. Foster, "Mapping the gnutella network," *IEEE Internet Computing*, vol. 6, no. 1, pp. 50–57, 2002.

[48] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2003, pp. 314–329.

[49] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," *IEEE/ACM Trans. Netw.*, vol. 12, no. 2, pp. 219–232, 2004.

[50] T. Karagiannis, A. Broido, N. Brownlee, K. C. Claffy, and M. Faloutsos, "Is p2p dying or just hiding?" in *Proceedings of the GLOBECOM 2004 Conference*. Dallas, Texas: IEEE Computer Society Press, November 2004.

[51] "Mojito," Website, Nov. 2011, http://www.ohloh.net/p/limewires_mojito.

[52] D. Stutzbach, S. Zhao, and R. Rejaie, "Characterizing Files in the Modern Gnutella Network," *Multimedia Systems Journal*, vol. 1, no. 13, pp. 35–50, Mar. 2007. [Online]. Available: http://www.springerlink.com/content/1367547006386758/

[53] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform resource identifiers (uri): generic syntax," 1998.

[54] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2006, pp. 41–52.

[55] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich, "Analysis of the Storm and Nugache Trojans: P2P Is Here," *;login: The USENIX Magazine*, vol. 32, no. 6, pp. 18–27, Dec. 2007. [Online]. Available: http://www.usenix.org/publications/login/2007-12/pdfs/stover.pdf

[56] "Analysis of phatbot," Website, http://www.secureworks.com/research/threats/phatbot/.

[57] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling, "Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm," in *LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–9.

[58] "Alureon botnet," Website, http://arstechnica.com/security/news/2011/07/4-million-strong-alureon-botnet-practically-indestructable.ars.

[59] P. Porras, H. Saïdi, and V. Yegneswaran, "A foray into confickers logic and rendezvous points," in *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2009.

[60] "Analysis of sinit," Website, http://www.secureworks.com/research/threats/sinit/.

[61] G. Memon, R. Rejaie, Y. Guo, and D. Stutzbach, "Large-scale monitoring of dht traffic," in *IPTPS '09: Proceedings of the 8th International Workshop on Peer-to-Peer Systems*, 2009.

[62] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage, "Spamalytics: an empirical analysis of spam marketing conversion," in *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2008, pp. 3–14.

[63] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2006, pp. 189–202.

[64] M. Steiner, D. Carra, and E. W. Biersack, "Faster content access in kad," in *P2P 2008, 8th IEEE International Conference on Peer-to-Peer Computing, September 08, 2008, Aachen, Germany*, 09 2008.

[65] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.

[66] C. Dixon, T. Anderson, and A. Krishnamurthy, "Phalanx: withstanding multimillion-node botnets," in *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2008, pp. 45–58.

[67] G. Memon, R. Rejaie, Y. Guo, and D. Stutzbach, "Montra: A large-scale dht traffic monitor," *Computer Networks*, 2011.

[68] Y. Song, M. E. Locasto, A. Stavrou, A. D. Keromytis, and S. J. Stolfo, "On the infeasibility of modeling polymorphic shellcode," *Machine Learning*, vol. 81, no. 2, pp. 179–205, 2010.

[69] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 181–192, 2007.

[70] D. Lagutin, K. Visala, and S. Tarkoma, "Publish/subscribe for internet: Psirp perspective," *Towards the Future Internet Emerging Trends from European Research*, vol. 4, pp. 75–84, 2010.

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

[72] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. G. Liu, and A. Silberschatz, "P4p: provider portal for applications," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 351–362.

[73] M. Varvello and M. Steiner, "Traffic localization for dht-based bittorrent networks," in *NETWORKING 2011*. Springer, 2011, pp. 40–53.

[74] "Akamai," Website, 2012, http://www.akamai.com.

[75] "First data on changing netflix and cdn market share," Website, 2013, http://www.deepfield.net/2012/06/first-data-on-changing-netflix-and-cdn-market-share/.

[76] "Level 3," Website, 2013, http://www.level3.com/en/products-and-services/data-and-internet/cdn-content-delivery-network/.

[77] "Limelight," Website, 2013, http://www.limelight.com/.

[78] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, "Globally distributed content delivery," *Internet Computing, IEEE*, vol. 6, no. 5, pp. 50–58, 2002.

[79] C. Huang, A. Wang, J. Li, and K. W. Ross, "Measuring and evaluating large-scale cdns," in *Internet Measurement Conference (IMC)*, 2008.

[80] A.-J. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante, "Drafting behind akamai: inferring network conditions based on cdn redirections," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 6, pp. 1752–1765, 2009.

[81] "Ip anycast," Website, 2013, http://en.wikipedia.org/wiki/Anycast.

[82] "Level 3," Website, 2013, http://www.akamai.com/dl/akamai/economu_mapping_the_internet.pdf.

[83] B. Augustin, B. Krishnamurthy, and W. Willinger, "Ixps: mapped?" in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 2009, pp. 336–349.

[84] E. Nygren, R. K. Sitaraman, and J. Sun, "The akamai network: a platform for high-performance internet applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 2–19, 2010.

[85] "Netflix," Website, 2013, http://www.netflix.com.

[86] "Hulu," Website, 2013, http://www.hulu.com.

[87] V. K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z.-L. Zhang, "Unreeling netflix: Understanding and improving multi-cdn movie delivery," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 1620–1628.

[88] V. K. Adhikari, Y. Guo, F. Hao, V. Hilt, and Z.-L. Zhang, "A tale of three cdns: An active measurement study of hulu and its cdns," in *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*. IEEE, 2012, pp. 7–12.

[89] S. Triukose, Z. Wen, and M. Rabinovich, "Measuring a commercial content delivery network," in *Proceedings of the 20th international conference on World wide web*. ACM, 2011, pp. 467–476.

[90] "Sandvine global internet phenomena report 1h 2013," Website, 2013, http://www.sandvine.com/downloads/documents/Phenomena_1H_2013/Sandvine_Global_Internet_Phenomena_Report_1H_2013.pdf.

[91] "Cisco vni forecast," Website, 2013, http://www.cisco.com/en/US/netsol/ns827/networking_solutions_sub_solution.html#~forecast.

[92] "Comcastic p4p trial shows 80http://arstechnica.com/uncategorized/2008/11/comcastic-p4p-trial-shows-80-speed-boost-for-p2p-downloads/.

[93] "Triad," Website, http://www-dsg.stanford.edu/triad/.

[94] D. Cheriton and M. Gritter, "Triad: A new next-generation internet architecture," 2000.

[95] M. Gritter and D. R. Cheriton, "An architecture for content routing support in the internet," in *USITS'01: Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems*. Berkeley, CA, USA: USENIX Association, 2001, pp. 4–4.

[96] J. Postel, "RFC 791: Internet protocol," 1981.

[97] D. Perino and M. Varvello, "A reality check for content centric networking," in *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*. ACM, 2011, pp. 44–49.

[98] Ashok Narayanan and David Oran, "NDN and IP Routing Can It Scale?" 2012, http://trac.tools.ietf.org/.

[99] "Facebook," Website, 2013, http://www.facebook.com.

[100] "Twitter," Website, 2013, http://www.twitter.com.

[101] "Royal pingdom," Website, May 2013, http://royal.pingdom.com/2013/01/16/internet-2012-in-numbers/.

[102] "Netcraft web server survey," Website, May 2013, http://news.netcraft.com/archives/category/web-server-survey/.

[103] P. Ganesan, K. Gummadi, and H. Garcia-Molina, "Canon in G major: designing DHTs with hierarchical structure," in *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*. IEEE, 2005, pp. 263–272.

[104] D. Lagutin, "Redesigning internet-the packet level authentication architecture," *licentiate's thesis, Helsinki University of Technology, Finland*, 2008.

[105] D. Clark, J. Wroclawski, K. Sollins, and R. Braden, "Tussle in cyberspace: defining tomorrow's internet," in *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4. ACM, 2002, pp. 347–356.

[106] L. Gao, "On inferring autonomous system relationships in the internet," *IEEE/ACM Transactions on Networking (ToN)*, vol. 9, no. 6, pp. 733–745, 2001.

[107] "Cidr," Website, May 2013, http://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing.

[108] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-level multicast using content-addressable networks," in *Networked Group Communication*. Springer, 2001, pp. 14–29.

[109] M. Castro, P. Druschel, A.-M. Kermarrec, and A. I. Rowstron, "Scribe: A large-scale and decentralized application-level multicast infrastructure," *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 8, pp. 1489–1499, 2002.

[110] "A border gateway protocol 4 (bgp-4)," RFC, 2006, http://www.ietf.org/rfc/rfc4271.

[111] S. Turner and M. Lepinski, "An overview of bgpsec," 2011.

[112] R. Austein, G. Huston, S. Kent, and M. Lepinski, "Rfc 6486: Manifests for the resource public key infrastructure (rpki). internet engineering task force (ietf), 2012."

[113] T. Griffin and G. Wilfong, "An analysis of bgp convergence properties," in *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 4. ACM, 1999, pp. 277–288.

[114] "Caida," Website, Jan. 2012, http://www.caida.org/.

[115] "Caida as-rank," Website, May 2013, http://as-rank.caida.org/.

[116] B. Quoitin and S. Uhlig, "Modeling the routing of an autonomous system with c-bgp," *Network, IEEE*, vol. 19, no. 6, pp. 12–19, 2005.

[117] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling tcp throughput: A simple model and its empirical validation," in *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 4. ACM, 1998, pp. 303–314.

[118] J. Aikat, S. Hasan, K. Jeffay, and F. D. Smith, "Discrete-approximation of measured round trip time distributions: A model for network emulation." GENI Research and Education Experiment Workshop, 2012.

[119] "Caida walrus," Website, Apr. 2013, http://www.caida.org/research/performance/rtt/walrus0202/.

[120] A.-J. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante, "Drafting behind akamai: inferring network conditions based on cdn redirections," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 6, pp. 1752–1765, 2009.

[121] V. Ramasubramanian and E. G. Sirer, "The design and implementation of a next generation name service for the internet," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 331–342, 2004.

[122] B. Ager, N. Chatzis, A. Feldmann, N. Sarrar, S. Uhlig, and W. Willinger, "Anatomy of a large european ixp," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. ACM, 2012, pp. 163–174.

[123] J. C. Cardona Restrepo and R. Stanojevic, "Ixp traffic: a macroscopic view," in *Proceedings of the 7th Latin American Networking Conference*. ACM, 2012, pp. 1–8.

[124] "Streamingmedia blog," Website, 2013, http://blog.streamingmedia.com/the_business_of_online_vi/2010/09/video-accounts-for-51-of-total-us-internet-traffic-heres-a-breakdown.html.

[125] "Youtube," Website, 2013, http://www.youtube.com.

[126] "Cdn pricing," Website, 2013, http://blog.streamingmedia.com/the_business_of_online_vi/2012/05/cdn-pricing-stable-survey-data-coming-this-week.html.

[127] "Akamai becoming primary cdn for netflix, but at a very low price," Website, 2013, http://seekingalpha.com/article/191356-akamai-becoming-primary-cdn-for-netflix-but-at-a-very-low-price.

[128] "Netflix shifts traffic to its own cdn; akamai, limelight shrs hit," Website, 2013, http://www.forbes.com/sites/ericsavitz/2012/06/05/netflix-shifts-traffic-to-its-own-cdn-akamai-limelight-shrs-hit/.

[129] "At&t cdn services," Website, 2013, http://www.business.att.com/enterprise/Service/hosting-services/content-delivery/distribution/.

[130] "An overview of transparent caching and its role in the cdn market," Website, 2013, http://blog.streamingmedia.com/the_business_of_online_vi/2010/10/an-overview-of-transparent-caching.html.

[131] W. Jiang, R. Zhang-Shen, J. Rexford, and M. Chiang, "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*. ACM, 2009, pp. 239–250.

[132] D. Mazieres, M. Kaminsky, M. Kaashoek, and E. Witchel, "Separating key management from file system security," *ACM SIGOPS Operating Systems Review*, vol. 33, no. 5, pp. 124–139, 1999.

[133] "amule," Website, 2013, http://sourceforge.net/projects/amule/.

[134] "Vuze," Website, 2013, http://www.vuze.com/.