CERIAS Tech Report 2011-15
Improving Internet Infrastructure: BGP Predictability and Cloud DNS Performance
by Ravish Khosla
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

IMPROVING INTERNET INFRASTRUCTURE: BGP PREDICTABILITY AND CLOUD DNS PERFORMANCE

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Ravish Khosla

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2011

Purdue University

West Lafayette, Indiana

Dedicated to my parents for their endless love and support.

ACKNOWLEDGMENTS

This dissertation and my graduate degree would not be possible without the contribution of several people. I would like to thank my advisors Dr. Sonia Fahmy and Dr. Y. Charlie Hu for their assistance throughout my PhD. Both of them provided me support in many ways, helping me define my research projects while providing valuable feedback. I would also like to thank other members of my advisory committee Dr. Xiaojun Lin and Dr. Ramana R. Kompella for their time in supervising my graduate research.

One of the important aspects of my graduate career is learning in the true sense of the word - for which I am indebted to professors and teaching assistants of all my courses and to Purdue University for providing the necessary infrastructure and the environment for learning. My interaction with people at Purdue has certainly provided me with valuable experiences.

Special thanks are due to the department of Electrical and Computer engineering at Purdue University for providing me a teaching assistantship, which not only funded the early part of my graduate education but also provided me an invaluable opportunity to interact with smartest brains from all around the world. The staff at the ECE Graduate Office, Matt Golden and Michelle Wagner, have been invaluable, especially in my final semester.

The contribution of my family in my graduate education has been immense, since they have supported me in every step of the way by providing valuable guidance and encouragement. All my friends at Purdue have also contributed either by balancing my life or by providing me obstacles crossing which made me a stronger person. Finally, I believe that the weather at West Lafayette, which has cheered me up on numerous occasions, has been an unlikely contributing factor in my success.

TABLE OF CONTENTS

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

BGP	Border Gateway Protocol
AS	Autonomous System
ASN	Autonomous System Number
TCP	Transmission Control Protocol
IP	Internet Protocol
OSPF	Open Shortest Path First
RIP	Routing Information Protocol
IS-IS	Intermediate System to Intermediate System
MRAI	Minimum Route Advertisement Interval
MED	Multi-Exit Discriminator
IANA	Internet Assigned Numbers Authority
DNS	Domain Name System
URI	Uniform Resource Identifier
ISP	Internet Service Provider
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
RTT	Round Trip Time
TTL	Time To Live
CNAME	Canonical Name
ROC	Receiver Operating Characteristic
AUC	Area Under the ROC Curve
TP	True Positive
FP	False Positive
TN	True Negatives

FN	False Negatives
CDF	Cumulative Distribution Function
GDNS	Google DNS
GFE	Google Front End
VGFE	Virtual Google Front End
DDoS	Distributed Denial of Service
CDN	Content Distribution Network
GTM	Global Traffic Management
PSTN	Public Switched Telephone Network

ABSTRACT

Khosla, Ravish. Ph.D., Purdue University, December 2011. Improving Internet Infrastructure: BGP Predictability and Cloud DNS Performance. Major Professors: Sonia Fahmy and Y. Charlie Hu.

The Internet has witnessed explosive growth over the last few decades, steadily evolving into a worldwide communication medium capable of supporting myriads of applications. While several efforts have been undertaken to improve the reliability of best-effort Internet communication, their adoption has been virtually nonexistent due to the lack of incentive for change and the presence of heterogeneous networks not controlled by a single entity. Moreover, the Internet structure is rapidly evolving into a flatter one composed of large organizations or clouds which hampers any efforts of retrofitting the existing Internet.

In this dissertation, we study two of the most important components of the Internet infrastructure, namely Routing and Domain Name System (DNS). We aim to find predictability in Internet routing, specifically the existence of Internet routes to prefixes, collection of IP addresses. We hypothesize that the Internet under Border Gateway Protocol (BGP), the de-facto interdomain routing protocol, while seemingly unpredictable, has a structure whereby prefix similarity can be exploited to successfully predict availability of Internet routes and route failures. We build data mining based prediction models using real-world routing data and find that this is indeed the case and the future availability of a prefix can be predicted by observing it for a limited time period and using the learned models. We also formulate *BGP molecules* which are the set of Internet prefixes that have similar propensity to become unreachable from portions of the Internet, *i.e.* to fail. We use these molecules in four

failure prediction schemes, among which a hybrid scheme achieves 91% predictability of failures with 99.3% coverage of prefixes in the Internet.

We study how DNS as an Internet infrastructure has evolved by investigating cloud-based DNS, which is the result of moving DNS services to the cloud. We perform a case-study of a recently launched cloud-based DNS, namely Google external DNS. A novel technique for geolocating data centers of cloud providers is developed and used to show that a query to Google DNS may not be redirected to the geographically closest Google data center. We also study Akamai-hosted content retrieval through cloud-based DNS and find that the client perceives worse performance as compared to the use of local DNS to retrieve content. The reasons for this poor performance are investigated and we explore the design space of methods for cloud-based DNS systems to be used by clients retrieving content. Client-side, cloud-side, and hybrid approaches are presented and compared, with the goal of achieving the best client-perceived performance. Our work yields valuable insight into Akamai's DNS system, revealing previously unknown features.

Finally, we present our vision of the evolution of the current Internet to the future cloud-based Internet, while specifying the *lightning* or interaction among clouds. We posit that while the cloud offers several advantages for hosting services, blindly using the cloud for every service can cause poor performance. Instead, a carefully balanced approach can usher a smooth transition from current Internet systems to the cloud-based Internet of tomorrow.

1. INTRODUCTION

The Internet was formed in the 1970s and 80s as a medium to interconnect heterogeneous networks and facilitate communication and information sharing among people all over the world. Indeed, the Internet has experienced exponential growth over the past couple of decades with around two billion Internet users as of March 2011 [1], which is around 30% of the world's population. Several applications have been developed to run over the Internet from communication based applications like voice and video communication, video streaming, news delivery, interactive gaming, file sharing, to critical applications like Internet banking, trading in financial markets, business transactions, patient monitoring in hospitals and air traffic control which require high availability, reliability and security [2]. With the advent of mobile devices, Internet connectivity on the move is gaining pace, with mobile users set to overtake fixed Internet users in a few years [3]. The Internet itself is rapidly evolving especially with the move to cloud computing in recent times. The goal of ubiquitous communication seems achievable, and a well-functioning healthy Internet is an important requirement for achieving the goal. This dissertation sheds light on the Internet infrastructure which is the backbone of the Internet, enabling all applications that run on it.

We now define some terms used throughout this dissertation that are used to study the "health" of the Internet. *Availability* is defined as the readiness for correct service or more precisely as the percentage of time a system is online and functional over the operating time duration of interest [4,5]. We define *predictability* as the ability to successfully predict the quality of service (QoS) metric of availability in the future.

1.1 Internet Availability

Reliable interconnectivity of the entire world at all times is the basic function that the Internet is expected to perform. Yet, this is not the case since the Internet exhibits frequent and numerous failures that reduce its availability as illustrated below.

Border Gateway Protocol (BGP) (Section 2.1.1) is the de-facto interdomain routing protocol in the Internet and is used for advertising existence of routes to Internet prefixes, which are collection of IP addresses. BGP is prone to several routing pathologies, which cause routing instability in the form of unnecessary announcements and withdrawals of routes to prefixes. Craig Labovitz *et al.* conducted a study in late 1990's of routing updates in backbone routers in the Internet and found that routing messages exchanged are dominated by pathological updates which do not represent real routing changes [6, 7]. They found that the majority of Internet backbone paths exhibit a mean-time to failure (MTTF) of 25 days or less, and a mean-time to repair (MTTR) of twenty minutes or less [6]. Our previous work [8] examined Internet routing data for the months of March to November 2007 (obtained from RouteViews [9]) and computed the MTTF to be between one and two months and MTTR to be between one and two days with median time to repair to be 17 minutes to one hour. This shows that routing failures happen less frequently in today's Internet; however, the recovery time is fairly large on the average. Another follow-up study to Labovitz's work [10] conducted in 2007 found that pathological BGP updates account for a lesser but still significant proportion (16%) of all routing dynamics. Such routing instability reduces the availability of advertised paths to portions of the Internet.

There have also been several failures in the Internet, some of which are mentioned below:

- April 1997 – A misconfigured router maintained by a Virginia service provider injected an incorrect routing map into the global Internet claiming that it provided optimal routes to all Internet destinations. This caused many Internet providers to divert traffic to that service provider and the resulting con-

gestion and router overload shut down the Internet backbone for around two hours [6, 11].

- August 1998 – Connections to ".net" Internet servers failed for many hours because of a misconfigured Internet database server [6].

- November 1998 – A malformed routing control message led to persistent, pathological oscillations and communication failures between most Internet core backbone routers for a period of several hours [6].

- April 2001 – AS3561 propagated improper route announcements from one of its downstream customers which led to global connectivity problems [12].

- September 2001 – Code Red/Nimda worm attack caused significant BGP session resets and routing instability to a significant portion of the Internet [13].

- October 2002 – Improper router filtering rules caused the internal routers of WorldCom to become overloaded and crash, which caused repeated announcements and withdrawals [14]. This caused significant destabilization of the Internet, causing many customers to experience high response times and packet loss to many websites.

- October 2002 – A series of Distributed Denial of Service (DDoS) attacks were launched against all thirteen root Domain Name Servers and popular websites which caused some of them to go offline [14].

- December 2006 – Earthquakes in Taiwan caused widespread outages to several countries in the region which lasted for around 51 days [15].

- January 2008 – Undersea cables in the Mediterranean Sea were cut which caused significant disruptions and increase in web latencies to much of the Middle East, Asia, and North Africa for a period of several weeks [15–17].

- February 2008 – An Internet prefix belonging to YouTube was mistakenly announced by a telecom operator in Pakistan, causing an outage for over two hours [15].

- The authors of [18] studied the reachability of a particular set of prefixes belonging to the U.S. Department of Defense (DoD). They found significantly high global unreachability (when none of the Internet hosts in their study can reach a prefix) durations with 17% of unreachability durations being longer than one hour.

- Significant numbers of attacks take place on popular websites and other Internet infrastructure on a daily basis. Moore *et al.* [19] observed between three and five thousand DDoS attacks per week in 2001. An Internet security report released by Arbor Networks in 2011 states that DDoS attacks have gone mainstream in 2010, increasing 1000% since 2005, crossing 100 Gbps for the first time ever [20].

Typical availability values of systems in our daily lives are "four-nines" (99.99%) or higher [5]. The US Public Switched Telephone Network (PSTN) was found to have better than 99.999% availability during a two year period in the 1990s [21]. AT&T, one of the dominant telecom providers in the US, expected its switches to fail for no more than two hours in 40 years [22], with implied availability greater than five nines or 99.999%. Aviation is also considered as one of the critical industries in our daily lives. The aircraft accident rate in 2010 was one accident for every 1.6 million flights or better than six nines (99.9999%), the lowest in aviation history [23].

There have been several studies which quantify availability values of Internet routes. Labovitz *et al.*'s work on Internet backbone showed that only 25-35% of Internet routes have availability greater than 99.99% [6]. 10% of routes even exhibited an availability of less than 95% [6]. V. Paxson conducted end-to-end route measurements between 37 Internet sites and found that routing pathologies occur with a probability of 1.5% to 3.4%, leading to an availability lesser than 99% [24]. In a more recent study, Andersen *et al.* showed that an average Internet path experi-

ences an outage lasting more than 30 minutes with probability 0.1%-0.4% [5]. Also, they found web server availability to be less than 99.7% [5].

The results presented above show that the Internet is unreliable since routing failures happen often and its availability numbers are lower compared to most reliable systems.

1.2 Internet Predictability

While the metric of availability is certainly very important in evaluating a computer system, we posit that its predictability is at least as important as the metrics themselves. Predicting the value of or a bound on the availability enables the system owner to evaluate the claimed availability of the system and to take appropriate measures, if necessary, to increase its value. A system with 99.9% availability may be looked down upon if it is expected to function with a five-nines availability; however, it will be considered perfect if it meant to function at that availability level. Similarly, predictable downtime of a service is usually much more acceptable than a sudden, unpredictable one.

In the context of the Internet, one can certainly calculate the availability of the Internet either by studying historical data, for example through RouteViews [9], or by conducting measurement studies (Section 2.2). However, that does not necessarily mean that one would be able to predict these metrics in the future. In fact, Internet failures are usually not predictable as can be inferred by the failure incidents mentioned in Section 1.1. While some of the failures are caused by unpredictable natural events unrelated to the Internet like earthquakes and undersea cable cuts, a substantial number of failures are caused by reaction of Border Gateway Protocol to frequently occurring events like connection failures between Internet hosts, routing misconfiguration and changes in routing policy of an organization which is part of the Internet. Some failures are caused by no external events but by mere routing pathologies or oscillations in BGP itself. We contend that the Internet is far too

important a system to not be sufficiently predictable and predictability is required to ensure satisfactory performance of Internet applications. This dissertation advances us towards the goal of Internet predictability.

1.3 Challenges

There are several challenges in predicting Internet availability. The Internet, by definition, is a collection of heterogeneous networks, owned by different organizations which set their own routing policies based primarily on their business interests [25]. These policies may interact in unpredictable manner causing loss of advertised reachability between two Internet hosts even when a physical link exists between them. For example, two ISPs A and B may be connected through their customer C, but will not be able to reach each other through C, since C does not provide transit service for its providers [26]. This connectivity loss can also happen on a transient basis, when a primary path between two hosts, which provides reachability, fails and the backup path cannot be traversed due to routing policies [27]. The Internet is comprised of around 39,000 Autonomous Systems (ASes) [28], hence the complexity of interactions between them is unpredictable, espcially because business agreements between organizations which shape routing policies are publicly unknown.

Failures of paths themselves are usually unpredictable, as the failures may be caused by optical fiber cuts, router reboots or congestion which can be difficult to predict. A study of link failures in Internet backbone by Iannaccone *et al.* found that about half of link failures cannot be attributed to scheduled maintenance [29] and hence are usually unpredictable. In fact, it is difficult to find the actual reasons behind a link failure [29]. The authors of [29] found that 10% of link failures last longer than 20 minutes, which is very high as compared to low downtime required for ensuring high availability. Failures may not be recoverable either due to non-existence of backup paths, or the backup paths may share the same physical infrastructure, resulting in correlated failures of both the primary and the backup paths [5]. Even

when the link failures recover in milli-seconds [30], it may take several minutes before BGP converges to a new valid route [31] and this unpredictable convergence time adds to uncertainty in availability [5]. Path exploration by BGP is the reason for slow convergence of BGP to alternate routes [32] and several route oscillations may occur before an alternate path is found. This problem is exacerbated by configurable BGP parameters like Minimum Route Advertisement Interval (MRAI), which rate limits advertisements sent by a BGP host [33]. BGP route flap damping [34] is another mechanism to reduce propagation of unstable routing information, which can exacerbate routing convergence sometimes by upto an hour [35]. Due to the interaction of these flexible mechanisms and the diversity of routing policies, BGP is not guaranteed to converge [36].

Another reason for the unpredictability of the Internet's availability is the potential for unexpected failures caused by external attacks or misconfigurations. As pointed out in Section 1.1, routing misconfiguration is widespread and it causes substantial loss to Internet availability. The potential of anyone to announce any Internet prefixes without authentication can lead to prefix hijacks [15], which will cause reachability failure to the intended destination. Other security attacks, like DDoS attacks launched from large numbers of Internet hosts are unpredictable and often cause widespread damage. Worms in the Internet can lead to surges in routing messages causing routers to overload and crash, which also lead to loss of reachability [13]. The magnitude of such surges is impossible to predict as it depends on the extent of worm infection of Internet hosts, which in turn depends on their security characteristics like operating system, security software etc. Attacks on DNS infrastructure, *e.g.* [14], can lead to an unpredictable amount of damage in resolving DNS queries critical at ensuring availability and the damage again depends on the extent and sophistication of the attack and the security properties of DNS servers.

It should also be pointed out that complete loss of reachability is not the only way Internet availability is reduced. Even when the Internet infrastructure announces a path to a particular host, the path may experience high latency and intermittent

availability. Such problems can be caused by network congestion, either due to short term traffic spikes (flash crowds *e.g.* those that caused Target's website to crash [37]) or a long term increase in traffic, for example by file sharing applications [5].

Finally, one of the important challenges in predicting Internet availability is its constant evolution. Not only is the Internet growing in terms of the number of Autonomous Systems (ASes) and announced prefixes [28], the Internet itself is changing in its structure. For example, most of the inter-domain traffic has migrated to large content providers [38]. This has in turn led to a flatter Internet where content is being delivered directly to the customers from the large content providers bypassing the traditional Internet backbone [38]. Mergers and acquisitions of various technology leaders contribute towards the Internet evolution as well. Depeering, especially of Tier-1 ASes, can lead to substantial changes in Internet routing and can happen due to disputes, like the one between Cogent and Level3 [39]. Also, the recent trends in cloud computing of moving software and services to the cloud is proof of rapid Internet evolution [40]. Even Internet infrastructure services like the Domain Name System (DNS) are being offered through the cloud by providers like Google [41] and OpenDNS [42].

In summary, there are several challenges in ensuring Internet predictability, which places restrictions on the applications which can be supported through the Internet. This has led to private backbones being setup by organizations for providing services and content [38]. For example, Labovitz *et al.* found that Google migrated the majority of its video and search traffic to its own backbone and direct interconnects with consumer networks [38]. This dissertation seeks to overcome these challenges and provide predictability to the steadily evolving Internet infrastructure.

1.4 Scope of the Dissertation

In this dissertation, we study the predictability of inter-domain routing, which is a critical component of the Internet infrastructure. We do not aim to predict the events

that affect the availability but rather the behavior of the Internet in response to the events as reflected in the availability metric. Specifically, we predict availability of Internet routes to arbitrary destinations by using statistical models which are learned using availability information of other random destinations. Neither do we predict other properties of inter-domain routing like stability, convergence, safety, security and integrity nor do we aim to find causes of failures, in which sufficient research already exists (Section 2.2.1). This dissertation also does not study failure diagnosis which has been studied in the literature as well.

The second part of this dissertation focuses on studying another integral part of the Internet infrastructure, namely Domain Name System (DNS). We do not study native DNS provided by the Internet Service Provider (ISP) as it has existed in the Internet for the last few decades. Instead, we study the evolved DNS in the form of cloud-based DNS. The impact of this evolution on applications like content retrieval is also studied. We do not study other evolving features of the Internet for example, the move to mobile Internet and multimedia content.

1.5 Problem Statement

This dissertation aims to study the predictability of the Internet infrastructure and the implications of its evolution. Specifically, we propose the following hypotheses.

- It is possible to predict the long-term availability of a prefix by observing it for a short period of time, using statistical models which are constructed by studying properties of other prefixes in the Internet. We also hypothesize that prefix characteristics which correlate with its availability can be found.

- We hypothesize that there are patterns in the reachability failures of various Internet destinations as represented by prefixes. While the causes for failures may be unpredictable, for any given prefix of interest, it is possible to find other prefixes in the Internet that have tendency to fail along with it.

- It is possible to use these patterns in reachability failures to improve the server selection schemes used in the Internet, which can improve the availability of the services.

- There exist solutions that can be applied to cloud-based DNS so that a client's performance while using cloud DNS is comparable to that when using native DNS provided by its ISP.

1.6 Contributions

In this dissertation, we study two components of the Internet infrastructure which enable worldwide communication, namely interdomain routing and DNS. Our work sheds light on the time dimension of the infrastructure by considering both short-term prefix failures and long term availability metric. The key contributions of this dissertation are as follows:

- We study which characteristics of an Internet prefix correlate with its availability (Chapter 3). We then use the information to build statistical models to predict long term availability of a prefix by observing its characteristics for a short period of time. The models are learned by observing characteristics of unrelated prefixes for a learning period. To our knowledge, this is the first work that uses prefix characteristics to predict availability.

- We coin the term *BGP molecules* which are set of Internet prefixes which have similar propensity to become unreachable or fail as a prefix of interest (Chapter 4). We investigate various metrics that quantify the propensity of a prefix to fail. Four failure prediction schemes are developed with and without the use of BGP molecules among which a hybrid scheme achieves 91% predictability of failures with 99.3% coverage of prefixes in the Internet.

- We present an application of BGP molecules in increasing control-plane availability of a Content Distribution Network in Section 4.6. We perform a case

study of Akamai's CDN, using experiments to demonstrate that a significant percentage of queries for Akamai-hosted content exhibit lower than five-nines availability over a period of a few weeks. Our scheme based on BGP molecules not only results in a near-perfect availability for all content queries, but also has the potential of reducing client-perceived latency to the servers returned.

- We develop a novel technique for geolocating data centers of cloud providers that use IP anycast (Section 5.3). The use of IP anycast implies that various data centers use the same IP address, hence conventional IP geolocation schemes cannot be used. Our technique is measurement based and lightweight and does not require setting up of any infrastructure.

- We perform a measurement study of one of the cloud-based DNS systems, namely Google external DNS (Section 5.4). Our study reveals that Google does not necessarily redirect its clients to its nearest data center. Our study also includes resolution of various websites through native DNS and Google DNS and comparing the results returned. We also perform Google searches using Google DNS and native DNS and find that the servers returned using Google DNS appear to be chosen using server load as primary criterion. This study provides insight into one of the major cloud providers of cloud-based DNS, which is an evolution of the traditonal native DNS.

- We study Akamai-hosted content retrieval using cloud-based DNS and demonstrate its poor performance as compared to the use of native DNS (Section 5.5). Our study uses Google DNS as an example of cloud DNS and diagnoses the reasons for the poor performance by using our geolocation technique for cloud providers (Section 5.3). We propose various solutions to this problem that would enable cloud DNS to perform comparably to native DNS, including a hybrid client-cloud approach that a client can use in today's Internet. To our knowledge, this is the first approach that tackles the problem of identifying nearby

Akamai content servers while leveraging cloud-based DNS. Our work yields valuable insight into Akamai's DNS system, revealing previously unknown features.

- Finally, we extrapolate the current Internet trends to develop a model of futuristic cloud-based Internet (Section 6.2). We investigate the cloud interactions under our model, exploring how performance and economic incentives can drive these interactions. We also investigate whether moving all services to the cloud is a good idea. Our work sheds light on the future of the Internet and the transition from the current Internet to the cloud-based Internet of tomorrow.

1.7 Outline

This dissertation is organized as follows. Chapter 2 explains the current Internet infrastructure and the Internet evolution currently underway. The chapter also presents existing research relevant to our dissertation.

Chapter 3 presents our work on predicting prefix availability using statistical models. We investigate which features of a prefix correlate with its availability and their relative importance in predicting it. We compare three different prediction models while investigating various time durations over which the models are learned.

Chapter 4 discusses our new prefix grouping BGP molecules and presents various metrics that can be used in constructing them. The chapter also evaluates various failure prediction techniques that can be used to predict unreachability incidences of a prefix of interest. We also present a novel application of BGP molecules in improving the availability of Content Distribution Networks.

Chapter 5 presents the impact of the DNS evolution in the Internet to cloud-based DNS. We study DNS used by content providers like Akamai and cloud DNS providers like Google. Our scheme to geolocate data centers of cloud providers which use IP anycast is presented and is used in the measurement study of Google DNS. The chapter also studies performance of Akamai-hosted content retrieval using cloud

based DNS and presents various techniques to improve client performance while using cloud DNS.

Finally, Chapter 6 presents our vision of the future Internet and its unique challenges, while concluding the dissertation. The chapter also presents future work on the predictability of Internet infrastructure and its evolution.

2. BACKGROUND AND RELATED WORK

The Internet relies on several components to ensure successful worldwide communication. In this chapter, we describe two of those components which are relevant to this dissertation, namely routing and Domain Name System (DNS). This chapter also presents a summary of research that relates to this dissertation. Specifically, we present literature that deals with dependability properties of the Internet like its availability and its predictability. We also present work on studying properties of routing and DNS, as well as Internet evolution, which includes cloud computing and the move of DNS services to the cloud. Finally, since this dissertation uses statistical techniques, this chapter also presents literature on the use of such techniques in networking research.

2.1 Internet Infrastructure

The Internet is an interconnection of various networks all over the world. These networks are owned and managed by several organizations and are hence called Autonomous Systems (ASes). ASes are assigned Autonomous System Numbers (ASNs) by the Internet Assigned Numbers Authority (IANA) [43]. Each AS owns a block of IP addresses, categorized as prefixes. A prefix is a collection of a particular number of IP addresses with its length denoting the number of bits in the prefix, and the remaining bits for the IP addresses it aggregates. For example, a prefix with length 24 (/24 prefix) aggregates 8 bits of the 32 bit IP address length or 256 IP addresses. The IP addresses are also allocated to ASes by IANA. As of this writing in September 2011, there are about 39,000 Autonomous Systems (ASes) in the Internet with around 375,000 prefixes [28]. Around 16,500 ASes originate only one prefix, whereas the highest number of prefixes originated by an AS is 3563 [28].

For using a service hosted on the Internet anywhere in the world, a host has to know two pieces of information - who (which IP address) to reach and how to reach it. The first piece is provided by Domain Name System (DNS) which translates URIs to IP addresses in the Internet. Routing provides the other piece of the puzzle by enabling a host to reach an IP address. Hence, two of the most important components of the Internet infrastructure are routing and DNS and we study them in subsequent subsections.

2.1.1 Routing

Routing in the Internet aims to provide a means for any host to reach any other IP address in the Internet, *i.e.* it provides reachability to an IP address. There are two major types of routing in the Internet - interdomain and intradomain. Intradomain routing is implemented within an AS, and the AS can choose any routing protocol it wishes to within its domain. The intradomain routing protocols used can be either link-state, *e.g.* Open Shortest Path First (OSPF) [44] and Intermediate System to Intermediate System (IS-IS) [45], or distance vector, *e.g.* Routing Information Protocol (RIP) [46]. Interdomain routing is implemented between ASes, so that any host in the world can connect to any other IP address located in any other AS. Border Gateway Protocol (BGP) [47] is the de-facto interdomain routing protocol in the Internet.

Figure 2.1 depicts the existence of interdomain and intradomain routing in the Internet between two communicating entities, an end-host (source) which contacts a server (destination). The end-host gains access to the Internet through an Internet Service Provider (ISP) labeled AS 1 in the figure, which can connect to one or more ASes. In this particular example, the traffic from the source to the destination traverses the ASes 1 and 5 enroute to the destination (as shown by the bold arrows). The path chosen is one of the many possible paths (AS1-AS2-AS4 is another path) and is determined by BGP, the interdomain routing protocol. The figure shows also sample internal topology of an AS, which contains several routers interconnected to

one another. Traffic enters through a router and exits through another, and the actual sequence of routers traversed within the AS is determined by the intradomain routing protocol.

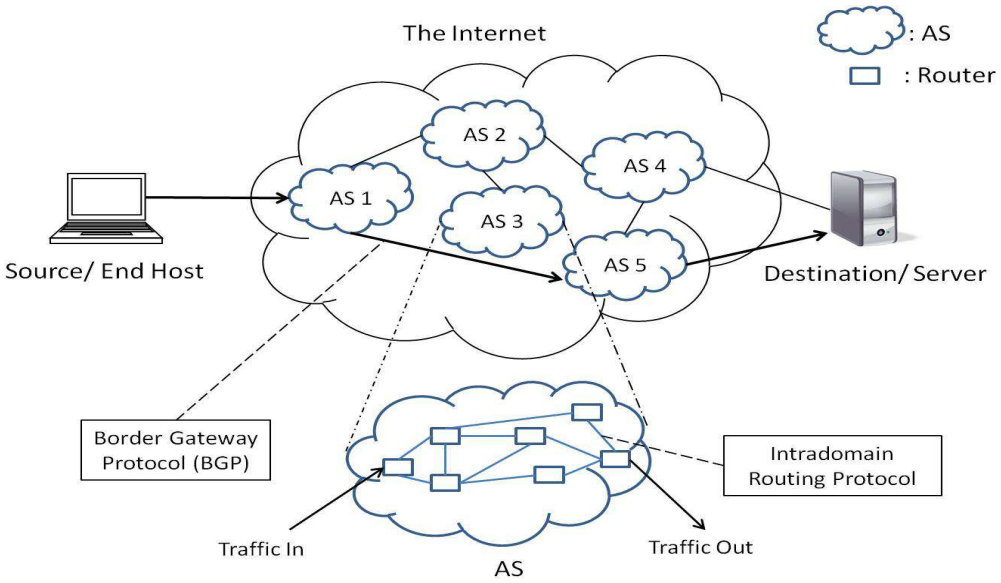


Fig. 2.1. Routing in the Internet with the bold arrows indicating traffic flow from the source to the destination

BGP communicates routes to Internet prefixes throughout the Internet through BGP sessions between border routers of ASes, as shown by the lines interconnecting various ASes in Figure 2.1. There are two types of messages used by BGP, namely Announcements and Withdrawals. Announcements announce routes from an AS to a prefix with some attributes whereas withdrawals withdraw those routes. There are several attributes of announcements, of which AS path and next hop are most important. AS path denotes the sequence of ASes that need to be traversed from the AS originating the advertisement to reach the destination prefix. Next hop is the next hop IP address enroute to the destination prefix. There are also other attributes like Local Preference, Community, MED and Aggregator [47] which are used in various ways to apply routing policies [25]. Upon receipt of an announcement or a withdrawal, a BGP peer, which runs a BGP session, evaluates the message and uses its attributes

in a BGP decision process to decide whether the new message gives it a more desirable path to reach the destination prefix [25]. The BGP decision process is influenced by various policies of the AS where the BGP peer is located. In case the existing path to the prefix is changed, the peer may decide to announce the new path depending upon its route export policies. Thus BGP is a policy based path vector protocol exhibiting significant flexibility in the choice of the path to a prefix, thereby satisfying business and traffic engineering considerations of an AS [25].

2.1.2 Domain Name System (DNS)

The Domain Name System [48] is an integral part of the Internet infrastructure, since it provides translation of human readable names like "purdue.edu" to an IP address. The DNS resolution process of converting a name to an IP address starts with the client contacting its local DNS resolver with the DNS query. If the resolver does not know the result of the query (by caching a previous result), it then contacts top-level root nameservers, which are provided to it by configuration through a reliable source like a system administrator. The top-level nameservers then delegate the query to other nameservers which are authoritative for their respective domains till the required IP address is found or an error is encountered. The multiple queries can be initiated automatically by a recursive DNS resolver or they can be initiated by the client itself if the resolver is non-recursive. Figure 2.2 shows an example of a recursive query for "cs.purdue.edu" initiated by a host. The query which is directed to the host's DNS resolver first goes to the root nameserver which delegates it to the authoritative nameserver of ".edu" domain, which returns "purdue.edu" nameservers. These "purdue.edu" nameservers then redirect the query to "cs.purdue.edu" authoritative nameservers which return the IP address of "cs.purdue.edu" to the DNS resolver which forwards the result to the client.

The DNS mechanism uses extensive caching at various levels to prevent repetitive queries, thereby saving time and network traffic. Thus, after returning the IP address

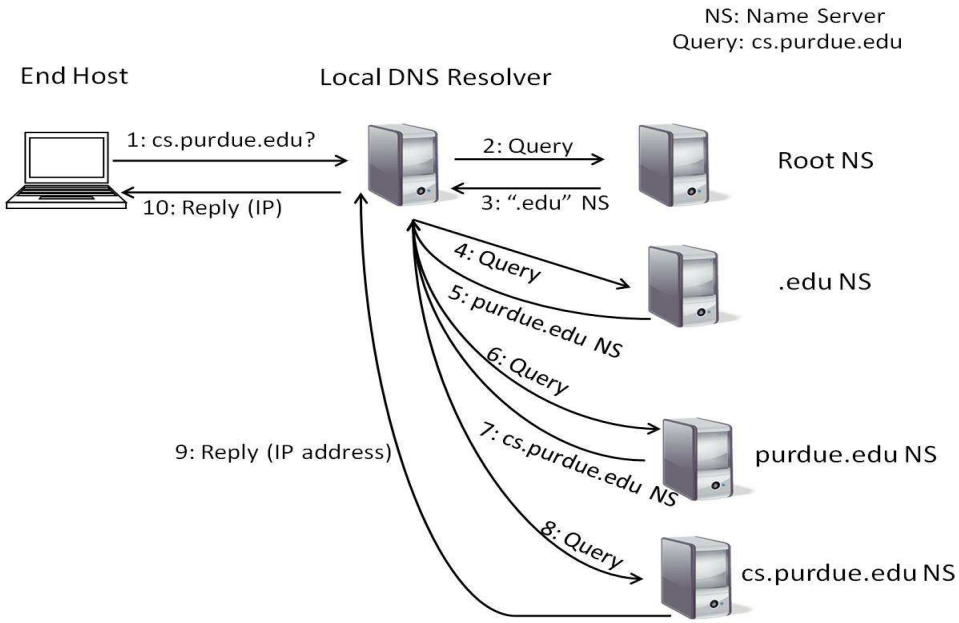


Fig. 2.2. Example resolution of a DNS query through a recursive DNS resolver

of "cs.purdue.edu" to the host, the DNS resolver in Figure 2.2 caches the reply for a certain period of time to serve future requests. Any of the nameservers shown in the figure can also cache the results to reduce the number of DNS queries. Due to the critical nature of DNS servers, especially the root nameservers, they are replicated to make the system resilient to failures and attacks [49].

2.2 Internet Dependability

There exists a vast body of literature on studying Internet availability, some of which have been mentioned in Section 1.1. In this section, we present work related to both routing and DNS dependability.

2.2.1 Routing Dependability

Analyzing the resilience of the Internet to failures and malicious attacks has been the subject of extensive research. Wu *et al.* develop a realistic failure model of the Internet, trying to locate the reliability bottlenecks using simulations on Internet topologies [27]. Their results indicate that the Internet may not be as resilient as is often thought to be. This is because of the policy based nature of BGP which does not gurantee reachability even if two nodes in the Internet are physically connected. Similar results are obtained by the authors of [50], who construct an AS graph model and define Internet resilience by the graph-theoretic metrics of strongly connected components and the proportion of nodes that can be reached in the graph. Other papers which analyze network robustness based on graph-theoretic properties are [51, 52]. The authors of [53] develop both static and dynamic graph-theoretic models of the Internet, considering Internet growth. Their results indicate that the Internet is robust to failures only if the failures are random, which confirms the results of [50] that targeted attacks can cause significant harm to the Internet.

Studies on routing availability have found that the availability numbers are low, with a significant number of routes exhibiting availability of less than 95% [6] and very few routes have availability numbers greater than 99.99% [5,6,24]. The mean-time to failure (MTTF) of Internet routes is usually found to be of the order of several days and the mean-time to repair (MTTR) is found to be under one hour [6, 8]. Dahlin *et al.* use HTTP and traceroute datasets between source-destination pairs and find average availability between 93% and 99.6% [54]. They also find that failure durations are heavy-tailed and can last for as long as 100,000 seconds (around a day).

Routing dynamics and convergence has also been studied extensively in the literature. Pathological routing messages constitute a significant percentage of routing updates exchanged [6, 7, 10], which leads to unhealthy routing dynamics. BGP is known to suffer from slow convergence which can last several minutes [31, 35] due to path exploration [32] and configurable BGP parameters (Section 1.3). Zhao *et*

al. studied the routing performance of a set of DoD prefixes and found that there were periods when the prefixes seemed to be globally unreachable [18]. These periods were significant, with only about 16% shorter than two minutes and 17% longer than an hour. The authors also found that in periods of routing stress like the Nimda worm attack, the updates contributed by these prefixes could be upto 13 times those of a typical Internet prefix due to certain BGP attributes [18]. During the Code Red/Nimda attack in 2001, a 30-fold increase in BGP updates by around 3% of the prefixes was observed, which was caused by extensive session resets and slow convergence of BGP [13]. Similar results were observed during the 2003 Slammer worm attack, where upto 100 times the normal number of updates were logged from a couple of ASes [55]. These results show the unhealthy nature of BGP dynamics, not just under stress but also in normal routing conditions. However, it is worth noting from these studies that only a few prefixes of the Internet contribute significant number of updates. This observation is confirmed by authors of [56]; their results indicate that 0.1% of the prefixes in the routing table contribute 10% of the routing updates. Popular destinations of the Internet which receive high traffic volume are usually much more stable in routing than the prefixes which receive less traffic [57].

The impact of BGP instability has also been a topic of recent studies. The authors of [58] design an online system that not only identifies significant routing disruptions (by correlating updates), but also estimates their impact on the flow of traffic by assigning a weight or a popularity index to each prefix and computing the total weight of prefixes that are affected by an event. Agarwal *et al.* [59] investigate whether BGP dynamics affect intra-domain traffic fan-out, by studying traffic within a tier-1 ISP network. The authors correlate the BGP routing table changes with packet traces and conclude that the traffic fanout is virtually unaffected by the routing changes. Wang *et al.* [60] inject routing changes on the PlanetLab testbed and investigate the changes in the end-to-end performance by active probing using the metrics of packet loss, delay and reordering. They find that routing failures contribute to end-to-end packet loss. Further, they find that iBGP configurations, MRAI timer values, and

failure locations have a significant impact on routing failures. A related problem (addressed in [61]) is how instabilities in the control plane affect the data plane. A measurement study is conducted to test the interaction between the planes by looking at the traffic towards a sink behind a BGP beacon with the beacon going up and down at regular intervals. The authors find that the data plane metrics of delay, drop, jitter and reordering are not significantly affected by changes in the control plane.

Significant work also exists on diagnosing the causes of poor Internet dependability in an effort to improve dependability. Studies such as [58, 62, 63] aim at finding the root cause of BGP dynamics. Caesar *et al.* [62] designed an Internet Health Monitoring System to analyze routing updates and identify the cause and location of the routing change that led to the update(s). The system correlates the updates observed in three dimensions: prefixes, time, and views. Updates which are likely caused by a single event are grouped together and classified into one of the equivalence classes of events. Feldmann *et al.* perform a similar study to locate the origin of a routing instability [63]. They also correlate the updates to identify the instability cause as either internal to an AS or at the edge between the two given ASes, using simulations to validate the results. Diagnosing the root cause based on BGP data alone can be error-prone. As pointed out in [64], several routing changes are not visible to eBGP, *e.g.*, internal changes within an AS. Partially incomplete BGP data can lead to incorrect diagnosis. The authors of [64] propose a troubleshooting service to be implemented in the network using an Omni server for each AS. However, this approach requires the participation and cooperation of ASes which may not be practically possible. Identifying BGP updates caused by "major" events (which affect reachability to many ASes) in the presence of noise from other BGP updates caused by minor events is studied in [65]. The authors use Principal Component Analysis (PCA) to extract clusters of ASes whose prefixes are likely affected by the same events, which enables them to cluster update streams. The WhyHigh tool [66] diagnoses the high latency to Google's data centers using active measurements correlated with BGP routing data. The tool found inter-domain routing inefficiencies and packet queuing

as the primary causes of latency inflation. Saxena *et al.* [67] analyze and compare the content distribution frameworks of YouTube, Dailymotion, and Metacafe, based on measurements from geographically distributed PlanetLab nodes. They investigate the variation in service delay with the user's geographical location, and with video characteristics such as age and popularity. Hubble is a Internet monitoring system that detects data plane reachability problems [68]. It uses BGP feeds and active measurements to pinpoint routing problems in real-time.

The tremendous interest in studying and diagnosing problems related to Internet dependability naturally leads to several pieces of work that present suggestions on improving the dependability [5]. Availability of paths between two hosts can be improved by using routing over multiple paths. Systems which use multiple path routing include Detour [69], Resilient Overlay Network (RON) [5], NATRON [70] and Scalable One-hop Source Routing (SOSR) [71]. Multi-homing or the use of multiple local links has been shown to improve communication latency [72]. Various multi-homing strategies have been discussed in the literature [73, 74]. Caching and prefetching of content *e.g.* web objects has also been shown to increase availability [54, 75, 76]. The authors of [77] propose to use BGP policy relaxation to enable BGP to use the physical links which are policy disallowed to recover from Internet failures. They also suggest to use Internet Exchange Points (IXPs) to forge new BGP sessions between ASes on the fly, as needed by routing failures. This implies creation of new AS relationships as required to recover from failures and may not be viable given the business decisions behind AS relationships. Wang *et al.* propose that reliability should be provided as an Interdomain service by having different networks providing redundancy for each other as mutual backup [78]. The authors develop algorithms to efficiently utilize network resources in case of failures and validate their technique through simulations on real world data.

As the size of the Internet grows steadily [28], Internet routing scalability is an important research topic [79]. There are two facets to routing scalability, namely routing table size and the rate of BGP updates. Bu *et al.* study BGP routing table

growth using RouteViews [9] and find that multi-homing, load balancing, address fragmentation, and failure to aggregate aggregatable addresses are the top causes of increasing BGP routing table size [80]. Out of these causes, address fragmentation contributes most to the routing table size whereas the contribution of load balancing grows the fastest, followed by multihoming. The fastest growing prefixes in the routing table have lengths greater than 17 and less than 25, and hence serve local rather than global interests. These results are similar to those obtained by a six year study of routing table growth [81]. Controlling the rate of routing table growth has been the subject of many research works, with suggestions ranging from applying policy filters [82], using forgetful routing [83] to changing the routing strategy itself [84, 85].

Huston *et al.* predict that the rate of BGP updates (churn) will increase at a rate faster than the BGP routing table size [86], which makes the scalability problem important. Elmokashfi *et al.* study the increasing rate of BGP churn in the Internet [87] and find that the updates have increased by 200% in a three year period of 2005-2007. The authors also perform what-if studies and study their scalability implications. Their results indicate that increased connectivity at the Internet core is the primary reason for increasing number of BGP updates, hence a flat Internet core is much more scalable than a vertically deep core [87]. A recent 2010 study by Elmokashfi *et al.* [88] analyzed five years of routing data to investigate the causes of BGP churn and identified baseline and daily peak churn as the two churn components. They found that the baseline churn increases at a slower rate than the BGP routing table growth rate, whereas the peak churn is an order of magnitude higher than the baseline churn.

Inter-domain routing security is certainly a very important factor in ensuring Internet dependability. Techniques to trace and prevent Distributed Denial of Service (DDoS) attacks [89–93] can improve the reliability of the Internet infrastructure. Butler *et al.* survey various attacks on BGP and the proposed solutions to enhance BGP security [94]. The authors indicate that while the solutions have not been

implemented due to the scale and complexity of the Internet, significant progress has been made in ensuring a reliable Internet in the future.

Table 2.1 summarizes the various facets of routing dependability discussed in this subsection.

Table 2.1
Various facets of routing dependability and relevant research

Routing Dependability Facet	Relevant Research
Internet robustness to failures and attacks	Graph-theoretic approaches [50–53], Simulations on Internet models [27]
Measurement studies of Internet availability	Route Availability [6, 24], Web server availability [5, 54]
Routing dynamics and convergence	Pathological routing messages [6, 7, 10], Slow convergence [31, 32, 35], Specific events [13, 18, 55]
Impact of BGP instability	On intra-domain traffic [59], On data plane metrics [58, 60, 61]
Diagnosis of routing failures	Locating root cause of BGP dynamics [58, 62, 63, 65], Internet monitoring and troubleshooting systems [62, 64, 66, 68]
Increasing routing dependability	Multiple path routing [5, 69–71, 78], Multihoming [73, 74], BGP policy relaxation [77]
Routing scalability	Routing table scalability [80–85] Routing message scalability [86–88]
Routing security	Preventing DDoS attacks [89–93], Enhancing BGP security [94]

2.2.2 DNS Dependability

DNS dependability has also been the subject of several studies. Jung *et al.* studied DNS performance using a measurement study and found that around a quarter of lookups give no answer, while a significant fraction (13%) yield a negative response [95]. The median lookup duration is found to be less than 100 ms but the performance degrades with higher number of referrals. The authors of [96] measured dependability characteristics of DNS through measurements. They found that the majority of DNS servers are highly available, however a significant fraction had "one 9" or less availability. Average recovery time was found to be of the order of hours.

Liston *et al.* [97] conducted a large-scale study of DNS performance as observed by clients and found that performance metrics like mean response time vary widely depending upon the client location whereas system adminstrator controlled metrics like TTLs exhibit low variation. This work underscored the need for a well-formulated study to prevent bias in DNS dependability studies.

Even though redundancy is built into the DNS system [49], operational choices and configuration errors create dependency among redundant servers decreasing its robustness [98]. The authors suggest proper operational choices *e.g.* placing redundant DNS servers in different geographical locations and mechanisms for detecting and eliminating DNS configuration errors to achieve DNS reliability.

The DNS infrastructure is the constant target of malicious attacks because of its inherent importance in the proper functioning of the Internet. Popular attacks on the DNS infrastructure include attacks against all thirteen root DNS servers in 2002 and 2007, attacks against Akamai DNS infrastructure in 2004 and against UltraDNS in 2009 [14, 99]. [99] lists major attacks on the worldwide DNS infrastructure since 1996.

There have been several efforts to secure or propose changes to the DNS infrastructure to make it resilient to attacks and misconfigurations, thereby increasing its dependability. Park *et al.* [100] present a cooperative DNS lookup service, CoDNS,

which can augment existing nameservers by creating a pool of peer nodes that aid in lookups in case of failure of local DNS. [101–103] propose solutions to attacks on the DNS infrastructure. RFC 3833 [104] analyzes various threats to the DNS system, and describes how DNSSEC [105], the security extension of DNS, handles these problems. The authors of [106] present a comprehensive taxonomy of attacks on DNS infrastructure identifying four major categories of DNS attacks, namely DNS hacking, routing table poisoning, packet mistreating and Denial of Service. They also present various current and futuristic solutions for securing DNS.

2.3 Internet Predictability

While significant work exists on analyzing Internet dependability, relatively fewer studies have studied Internet predictability, which is the focus of this dissertation. In this section, we present work related to Internet predictability and point out the uniqueness of our work.

Predictability of network performance has been extensively studied, *e.g.*, in [24,54, 107]. These studies focused on end-to-end loss, delay, and throughput, as measured by active probes. Several network distance estimation techniques have been proposed in the literature, *e.g.* IDMaps [108], GNP [109], Vivaldi [110], Meridian [111], PlanetSeer [112] and non-metric approaches [113, 114]. These techniques use embedded coordinate system, constructed using measurements, to predict the latency between arbitrary nodes by using their corresponding vector distance in the coordinate space. However, these techniques suffer from being agnostic to the Internet structure, which lends them incapable of predicting detour routes or paths which do not follow the coordinate system [115]. There have also been research efforts in predicting TCP throughput of bulk file transfers, *e.g.* [107, 116–119]. All these techniques predict network properties in the data plane which is orthogonal to our work of predicting properties in the control plane (Chapters 3 and 4).

An information plane for distributed services, iPlane [115, 120, 121] has been developed at University of Washington for detecting data plane reachability problems and predicting data plane paths and their properties. iPlane is a measurement infrastructure that continuously performs measurements to maintain an Internet atlas annotated with path attributes. The atlas is used to predict paths between two arbitrary Internet hosts and subsequently its properties like latency, bandwidth and loss rate. iPlane Nano [121] is a lightweight version of iPlane which runs as a peer-to-peer application on client machines for performance prediction. The work in this dissertation on predicting Internet properties like availability (Chapter 3) differs from iPlane in several respects. We predict BGP advertised control plane prefix availability as opposed to predicting data plane metrics between end hosts in the Internet. In this sense, our work is complementary to iPlane. Furthermore, iPlane only provides an estimate of the availability in the data plane through loss rates, which may not be indicative of the advertised availability of a prefix. In fact, iPlane Nano [121] claims loss rate stationarity by stating that 66% of paths which were lossy at a time instant were lossy 6 hours later. Not only is this at a low granularity (since failures happen in order of seconds) but also this does not tell us anything about the availability of a particular path, let alone of a prefix when viewed from various vantage points. iPlane probes an end host (.1) of a prefix (known to be responsive to ICMP or UDP probes) once per day, collecting reachability samples for the end host. Even if one assumes that the end host's availability is the same as that of the prefix and that one can find a responsive end host in a prefix, the low-frequency availability samples collected are not sufficient to predict availability. Our approach of observing updates in the control plane has the advantage that, barring collection errors, *all* updates are recorded showing every up or down state change of a prefix, thus providing a continuous estimate of availability without the need for a new infrastructure.

There have been a couple of research papers on predicting reachability failures of Internet prefixes in the data plane. Feamster *et al.* [122] correlate BGP dynamics with active probing measurements in an aim to understand the relationship between the

data plane and the control plane. They find that BGP messages correlate with only half of the data plane failures, however BGP traffic is a good indicator that a failure has recently occurred or is about to occur. Their results indicate that a combination of observing BGP traffic and reactive routing can avoid many data plane failures. Zhang *et al.* [123] predict the impact of routing changes on the data plane. They aim to predict reachability problems based on problematic ASes on the AS paths of the routing updates observed for a prefix. Their results indicate that a majority of prefixes became unreachable after routing changes. Overall, their prediction model performs fairly well, achieving a 90% accuracy at predicting data plane failures with a false positive rate of less than 15%. These works are complementary to failure prediction as described in our dissertation in Chapter 4. This is because we focus on predicting control plane failures, *i.e.* loss of advertised reachability of a prefix, *without* taking into account any of its failures in its past. Success of our control plane failure prediction mechanism coupled with the results of [122, 123] imply that our technique will also be fairly reliable at predicting data plane failures.

2.4 Internet Evolution

Any study on the Internet cannot ignore its dynamic nature. The Internet is an ever-changing collection of heterogeneous networks. One of the aspects of the change is that the Internet is continuously growing with new networks being added on a regular basis [28]. For instance, the number of ASes has grown from around 2000 in 1996 to around 39000 in September 2011. The Internet inter-domain traffic grew at an annualized rate of 44.5% from 2007 to 2009 and was estimated to be around 39 Tbps in 2009 [38]. Dhamdhere *et al.* [124] studied the Internet evolution for a period of ten years from 1998 to 2007. They found that the Internet's growth has been in two phases - with exponential growth of ASes and links observed upto mid-2001 and linear growth thereafter. However, the average AS path length observed in routing tables has remained about the same at 4.2 hops. The authors also observed

that the Internet changes not only with the birth of new ASes and death of existing ones but also by rewiring, *i.e.* change of connectivity of existing ASes. In fact, at least 75% of all link births and deaths are attributed to AS rewiring with the highest rewiring occurring in Content Providers and Access Providers. This indicates that the Internet is continuously evolving with ASes changing their connectivity to best meet their business needs.

Recent years have seen the emergence of large data centers, especially of content providers like Akamai, Google, Facebook, Microsoft and Yahoo [125]. The decrease in price of transit traffic [126] coupled with the dominance of services and content on the Internet has led to rewiring of the Internet [38, 127]. This has led to significant changes, *e.g.* content providers building their own global networks for carrying their traffic, cable Internet service providers like Comcast offering wholesale national transit, and transit ISPs doubling up as Content Distribution Networks (CDNs) [38]. Labovitz *et al.* studied the Internet over a two year period of 2007 to 2009 and found that content providers like Google and Comcast are now in the top 10 contributors of inter-domain traffic, along with traditional Tier-1 ISPs [38]. The authors also found that thirty out of approximately thirty thousand ASes contribute a disproportionate 30% of Internet inter-domain traffic, of which Google is the largest and fastest growing. Other results of [38] indicate that Google migrated the majority of its video and search traffic away from transit providers to its own network infrastructure which directly interconnects with consumer networks. The consequence of these emerging Internet trends is that the Internet is getting *flatter* with content increasingly being delivered directly from the providers to the customers bypassing the traditional Internet *backbone* of Tier-1 providers like Sprint, Level3 and Global Crossing [38, 125, 127].

2.4.1 Data Centers

The emergence of large data centers has led to substantial research in the networking issues faced by these data centers. Traffic characteristics of data centers need to

be studied for purposes such as data center design, traffic engineering, load balancing, and estimating popularity of content and services. Traffic inside a distributed query processing cluster is studied in [128]. The authors study both macroscopic characteristics of congestion, server communication as well as microscopic characteristics of flow durations and inter-arrival times. Their results indicate that the statistics are more regular *e.g.* large "elephant" flows occur for much lesser time than that in ISPs. This is likely due to the the fact that the data center is managed by a single entity, with tight coupling of computing, storage and networking. Benson *et al.* examine spatial and temporal variations in link loads and losses in 19 data centers [129] and find that the links at the data center core are more heavily utilized than the ones at the edge, and these edge links exhibit higher losses. YouTube data center traffic is studied in [130], using sampled flow-level data from a tier-1 ISP. The authors study the interplay between the ISP and YouTube, observing that the ISP performs early-exit routing, and the traffic from client to YouTube always enters YouTube's network at the PoP nearest to the source PoP, irrespective of the destination data center. The load balancing strategies used by YouTube and their effects on the Tier-1 ISP are also studied.

Chen *et al.* studied inter-data center traffic of five Yahoo! datacenters [131]. They found that the datacenters are hierarchical in nature with smaller satellite data centers and larger primary data centers. Novel techniques to separate the inter-data center (D2D) traffic from the data center to client traffic (D2C) are developed. The D2D is found to be quite dominant and exhibits lower variance than the D2C traffic. They also found that many of the services provided by Yahoo! have correlated traffic which enables their placement in the same datacenter. The trend of deploying satellite data centers maintaining persistent connections with the primary data centers is studied in [132]. The authors answer questions about the number of satellite data centers that should be deployed along with their deployment locations and their peering connectivity for best client performance.

The presence of several data centers, which can be geographically spread out all over the world raises the issue of which data center a client needs to be redirected to. This problem is known as Global Traffic Management (GTM) problem [133] and is a dynamic optimization problem, which aims to optimize the performance an incoming client request will receive in real time. GTM not only needs to take care of the characteristics of the client request, like the type of request and the client location, but also needs to consider internal factors such as the network conditions between data centers and the data center load. Various GTM schemes are surveyed in [133], and we present some of them here for completeness. The authors of [130] find that YouTube, before its integration with Google, performs location-agnostic load balancing, redirecting clients to data centers not based on their location but based on the size of the data centers. While this solution is simple, it degrades the client latency significantly, since a client may be redirected to a far-off data center.

An alternative is to use geography based redirection, where commercial GeoLocation databases are used to map client IPs to geographic locations, and the data center closest to the client is selected to serve the client request [134, 135]. Such solutions suffer from the lack of dynamic adaptation to network conditions [133]. Another GTM solution uses IP Anycast [136], where all data centers announce the same IP address. This solution is also used by Google Public DNS, which announces the IP addresses 8.8.8.8 and 8.8.4.4 through IP anycast [137]. This technique suffers from the observation that the anycast closest data center may not always be the closest data center to a client as pointed out by [138,139] and our case study of Google DNS in Section 5.4.

Other GTM solutions include measurement approaches, both active and passive. Active measurements are conducted to get an idea of the network conditions so that an incoming client request can be redirected to the data center, where it'll observe the least latency. This technique is used by Akamai which uses large-scale Internet measurements for this purpose [140]. Passive monitoring can also be used to estimate the latency between the clients and the data centers. For example, one can use the

TCP handshake to estimate this latency. However, such solutions will suffer from redirection of the clients to sub-optimal data centers occasionally [66, 141], which makes this technique unattractive.

While GTM techniques decide which data center a client should be redirected to, there is need for a mechanism to achieve this redirection. This is achieved by the use of DNS infrastructure. When a client queries a CDN nameserver for obtaining a server that will serve its request, after being redirected through all higher level namservers (Section 2.1.2), the nameserver uses its GTM solution to pick a server within an appropriate data center and returns it to the client. This technique is used by Akamai, which uses two levels of DNS servers [142]. We discuss Akamai DNS in detail in Section 5.2.

2.4.2 Cloud Computing

Another strong emerging trend in the Internet is the emergence of cloud computing. While certainly not a new technology, it is making significant strides with software and services being delivered over the Internet [40]. We define a *cloud* as an organization in the Internet, which provides any Internet service be it content hosting, transit, or DNS. A cloud can consist of multiple Autonomous Systems (ASes) of the current Internet. Indeed, there are organizations in today's Internet which have multiple ASes – a whois lookup of Google in ARIN [143] yields ASes 15169, 36039, and 36040 as belonging to Google. We make no assumptions on the geographic spread of a cloud: it can consist of one or more data centers potentially spread all over the world, which are connected by high-speed links. Cloud computing offers several advantages, such as cost reduction because of no direct infrastructure setup, potential availability of infinite resources on demand, and payment for only the resources used [40].

While the construction of low-cost data centers was certainly an important factor in the shift to cloud computing, other factors like new technology trends and business opportunities have provided the impetus for the move to the cloud [40]. For example,

recent business trends of Internet payment using Paypal, which requires an email and a credit card to transfer money, has replaced the extensive infrastructure required previously to accept credit cards over the Internet. Cloud services enable common users to host their content and deliver it all over the world with no direct infrastructure involvement. The move to real-time mobile applications and the appetite for content has led to extensive data center infrastructure, which can easily provide virtual machine cycles for several other services like computing, and generate revenue.

There are four major types of cloud services: storage, infrastructure, platform, and software [144, 145]. One can use the cloud for simple services like storing email, or for complex services like service hosting. Recently, even basic services like the Domain Name System (DNS) are being offered through the cloud by providers like Google [41] and OpenDNS [42]. Major cloud service providers include Amazon (Elastic Compute Cloud and Web Services [146]), IBM (Smart Cloud) [147], Facebook's social network [148], Google (App Engine [149], Docs [150] and public DNS [41]), Oracle (SaaS [151]), Microsoft (Cloud Solutions and Office 365 [152]), and Apple (iCloud [153]).

Armbrust *et al.* list several challenges to cloud computing [40] and we discuss some of them here. Providing high availability of cloud services is a top priority for any cloud provider, especially since many cloud services compete with services running on an end user's machine, which have high availability. The issues are similar to those discussed in Section 1.3. Another challenge to cloud computing is the existence of many cloud service providers which are not interoperable with one another. The authors of [40] list this lack of interoperability as the second biggest obstacle to cloud computing after ensuring high cloud availability. The lack of standards for operating cloud services has led to users being unable to switch cloud services from one provider to the other. Oftentimes, a user's data is locked with the cloud provider and may even be lost if the provider fails. We recognize this important issue of interactions among clouds and discuss it further in our visions of the future Internet in Section 6.2.

The Internet infrastructure is also undergoing changes with the emergence of cloud computing. The traditional DNS service offered by ISPs is now moving into the cloud. A client can simply configure his browser to use external DNS resolvers and this can provide him content resolution, perhaps with better performance and security [137,154]. Cloud-based DNS systems like those hosted by Google, OpenDNS, and L3DNS [41, 42] have recently gained popularity. The authors of [139] estimate that around 2.5% of worldwide clients, visiting a popular website in their experiments, are using public DNS systems. The authors find that Google Public DNS is growing rapidly with about 0.5% of all clients using it within six months of its launch. We study Google public DNS further in Chapter 5.

Ager *et al.* [155] compare two cloud-based DNS systems, namely Google DNS and OpenDNS. Their measurement study shows that local DNS provided by the ISP far outperforms public DNS in terms of lower latency from the client, which increases their responsiveness. However, a bigger problem in using public DNS systems is that the servers returned upon resolution are usually not close to the client. We encounter the same phenomenon in our studies presented in this dissertation (Section 5.5). While the authors of [155] show that the content servers returned by cloud-based DNS systems can be in different ASes from the client, they do not investigate causes and solutions to the problem. We do so in this dissertation for Akamai, a deeply distributed popular CDN, in Section 5.5.

2.5 Statistical Techniques in Networking

We conclude this chapter with a discussion of some research efforts that employ statistical techniques in networking. We use a fair amount of statistical techniques ourselves in this dissertation, using data mining techniques to predict prefix availability in Chapter 3 and statistical techniques to predict failures using BGP molecules in Chapter 4.

Chang *et al.* [156] cluster routing updates into events based on the announcing peers and similarity of AS paths using *descriptive modeling* as the data mining technique. This technique is used for summarizing the data and improving understanding of the data features. In contrast, we use *predictive modeling* in Chapter 3 to predict prefix behavior, specifically availability, given the observed values of prefix attributes. Zhang *et al.* [123] predict the impact of routing changes on the data plane by using likelihood ratio tests. We also use this test to evaluate the success of our failure prediction application, similar to [122]. The authors of [65] use Principal Component Analysis (PCA) to extract clusters of ASes whose prefixes are likely affected by the same events, which is then used to cluster update streams. PCA is a data exploration technique which reduces the dimensionality of the data while preserving its variability. This is used by the authors to identify a few underlying events that cause the updates affecting a lot of prefixes.

Beverly *et al.* [157] predict round-trip latencies to random network destinations using Support Vector Machines (SVM) regression as the prediction technique. They achieve a fair performance with the latency prediction within 30% of the true value for three-quarters of a real-world dataset. SVMs are also used by [158] to discover a neighbor with the least communication cost in a P2P network and to predict TCP throughput in [159]. A Bayesian classifier for predicting a host's operating system from TCP/IP packet headers is studied in [160]. An IP address clustering algorithm is presented in [161], which clusters IPs from the 32 bit address space using supervised learning, such that the IP addresses within a same cluster share a property *e.g.* latency or membership in a botnet.

Change Detection is an important statistical approach, which has been applied to research in many fields, including networking. The technique develops a model for normal behavior and detects deviation from this behavior. Krishnamurthy *et al.* [162] develop a new change detection technique, namely sketch based change detection, which can be used to detect traffic anomalies. Sketches enable the authors to build compact summaries of the traffic, which are then fed to various time series prediction

models to detect changes, which happen when the forecasting error is high. Bloom filter is a space efficient randomized data structure which has been used extensively in the networking community. [163] provides a detailed survey of how bloom filters have been applied to many networking areas, which include peer-to-peer networks, routing and measurement. In summary, applying the extensive research conducted in the statistics community to networking can lead to innovative insights, and we follow this approach in this dissertation.

3. PREDICTING PREFIX AVAILABILITY

This chapter presents our work on predicting availability of an Internet prefix. We motivate the problem in Section 3.1, following up on Section 1.2. We then define the problem that we study in Section 3.2. Section 3.3 describes our datasets, and Section 3.4 describes our methodology and metrics. In Section 3.5, we compare results from three prediction models and study the effect of classification attributes and using certain more predictable prefixes on prediction results. Finally, Section 3.6 concludes the chapter.

3.1 Motivation

Continuous prefix reachability over time is crucial for the smooth operation of the Internet. This is captured using the metric of *availability*, defined as the time duration when the prefix is deemed reachable divided by the total time duration we are interested in. Prefixes belonging to highly popular services such as CNN, Google, and YouTube need to be highly available, and a disruption of more than a few minutes is generally unacceptable. Internet Service Providers (ISPs) such as AT&T and Sprint usually provide availability guarantees on their backbone network through Service Level Agreements (SLAs) [164, 165]. However, content providers are more interested in the availability of their services and content as observed from various points in the Internet, and a routing path being advertised is critical to maintaining traffic flow to their data centers. Attempts at defining policies so that SLAs can be extended to several ISPs [166] and at defining and estimating service availability between two end points [167] in the Internet have had limited success. Meanwhile, several reachability problems have occurred in the Internet, as described in Section 1.1.

Measuring prefix availability is non-trivial without an extensive measurement infrastructure comprising many vantage points. Additionally, *data plane* measurements are inherently discontinuous, as they take reachability samples at periodic time instants. The reachability estimate they compute increases in accuracy as the sampling interval is made smaller, at the cost of increased burden on the prober and elevated network traffic. Moreover, the observations need to be made over a long period of time to obtain a reasonable estimate. A shortfall in measured availability requires a *reactive* approach that corrects the problem after the fact. Our work takes a *predictive* approach to solve the *availability prediction problem,* i.e., predicting the advertised *control plane* availability of prefixes, as observed from multiple vantage points in the Internet.

While our framework predicts long-term control plane availability, control plane has been shown to have some positive correlation with the data plane in the literature. Wang *et al.* [60] studied the correlation between control plane and data plane events and found that control plane changes mostly result in data plane performance degradation, showing that the two planes are correlated. The authors of [123] found that data plane failures can be predicted using routing updates with about 80-90% accuracy for about 60-70% of the prefixes. Transient events like routing convergence and forwarding loops result in temporary reachability loss in the data plane, most of which last less than 300 seconds [123]. However, since we are concerned with the long term availability metric considering at least a few days at a time, the percentage of time that the control plane and data plane paths mismatch should be insignificant compared to the time over which our availability values are computed.

Data plane reachability can exist even when control plane paths are withdrawn due to the presence of default routes [168]. However, it is not possible to predict the existence of default routes, as they depend on intermediate ASes between the source and the destination. There is no agreed upon method to detect the existence of default routes either, though some initial efforts have been made by the authors of [168] by controlling announcements and withdrawals of certain prefixes allocated

to their ASes. Our work considers only control plane availability and hence actual prefix availability could be higher in the data plane if default routes are present. As can be seen from the discussion above, establishing the correlation between the two planes is by itself a challenging topic [168] and detailed study of this is beyond the scope of this work.

In this chapter, we compute attributes during a short duration observation period of publicly available routing information (*e.g.*, from RouteViews [9]) and develop a prediction model based on information on other Internet prefixes. Thus, our approach does not need additional measurement infrastructure apart from RouteViews [9], which has been maintained by the University of Oregon for several years.

A predicted long-term advertised availability value which falls short of requirements could lead to changes in BGP policies of the ISP regulating the advertisement of these prefixes to the rest of the Internet. For example, one can increase the penalty threshold associated with route flap damping for the routes to a high availability requirement prefix (like a business customer) to ensure higher availability [25]. Changing BGP attributes such as MED and community, or aggregating prefixes, can increase the perceived prefix availability or aid traffic engineering [25].

This work can optimize Hubble [68] – a system that studies black holes in the Internet by issuing traceroutes to potentially problem prefixes, and then analyzing the results to identify data plane reachability problems. Currently, Hubble uses BGP updates for a prefix as one of the potential indicators of problems, focusing on withdrawals and AS path changes. We can enhance this technique by using the prefixes for which the predicted availability falls below a threshold as the potentially problem prefixes. This will increase detection accuracy of black holes. Our work also complements a data plane loss rate prediction system such as iPlane [120].

Other applications of our work include Content Distribution Networks (CDNs), cloud computing applications, VoIP applications, and P2P networks. CDNs and cloud computing applications can use the highest predicted availability replica/server to redirect the clients to. VoIP implementations can use predicted availability of relay

nodes along with latency and loss rate estimates for better performance. Our work can also be applied to peer to peer networks, where ensuring content availability is a primary concern amid extensive peer churn. One can modify the incentive mechanisms of BitTorrent [169] by unchoking the BitTorrent peers which are parts of a highly available prefix, in addition to considering their download rate and latency/loss rate estimates. Our system eliminates the need for storing information about peers at clients that are not currently downloading from these peers but may do so in the future.

The key premise of this work is that Internet prefix characteristics convey valuable information about prefix availability. We argue that prediction models are viable even if prefixes whose availability is to be predicted and prefixes used for learning prediction models are unrelated (*e.g.*, learning and predicted prefixes are not in the same AS). This is because an important factor causing paths to prefixes from various vantage points to go up or down is BGP path convergence, caused by BGP reaction to path failure or policy changes. This, combined with the fact that operator reaction to path failures is relatively standard, and that AS policy changes, *e.g.*, AS depeering, typically affect several prefixes at a time, supports this premise. We therefore use randomly selected prefixes from RouteViews to learn models, and then predict availability of other prefixes. This theme is common in other disciplines, such as medicine, where one uses known symptoms of patients with a diagnosed disease to try to diagnose patients with an unknown condition. To the best of our knowledge, no other work has exploited the similarity of prefixes in the Internet; a few studies, *e.g.*, [123] applied predictive modeling in the context of BGP, but they only examined problem ASes in the path to a particular prefix (Section 2.5).

While we focus on predicting prefix availability using observed routing updates, our prediction framework can be easily extended to predict other prefix properties of interest. We formulate hypotheses about how attributes of a prefix such as prefix length and update frequency relate to its availability, and prove or refute them based on our data. We show that past availability of a prefix is inadequate for accu-

rately predicting future availability. Our availability predictions from three models are compared to measured availability values from RouteViews.

3.2 Availability Prediction Problem

We define the *availability prediction problem* to be the prediction of the BGP-advertised long-term availability of a prefix, given its attributes computed by observing BGP updates (for example, through RouteViews), and the availability and attribute information of other prefixes, collected for a short duration of time. Advertised availability is critical in maintaining smooth traffic flow to these prefixes. Going back to our patient analogy, given the symptoms and known diseases of some patients, one can use test results of a new patient to diagnose the new patient's disease. Our "test results" are the updates observed for a prefix for a limited period of time, which are used to predict its long-term availability.

In this chapter, we compute availability in the *control plane* by marking the time of an announcement of a prefix as the time when it goes up and a withdrawal as the time when it goes down and matching our predictions against this computed availability. The chance of an event getting lost only exists if the update associated with the prefix is not recorded. Spurious announcements and withdrawals are filtered as described in Section 3.3.

Rather than predicting continuous values of availability, we discretize availability, and predict the availability *class* of a prefix for some time period in the future, based on information collected from the past that is used to train prediction models. This is because, for diagnosis or detection purposes, our interest lies in predicting whether the availability value is above or below an acceptable threshold (*e.g.*, that advertised in SLA), and not the specific value of the availability. Discretizing also gives us an added advantage of use of confusion matrix-based measures, *e.g.*, false positives, to assess prediction performance. Using continuous availability values causes problems in defining error measures because a miss in high availability values (*e.g.*, 99% predicted

as 94%) counts more than a miss in lower values (*e.g.*, a predicted 35% instead of 40%) because of attached importance to higher values. In this chapter, we validate our predictions by computing the "future" availability class and comparing it with the predicted class. However, this is purely for validation of our prediction schemes – in a real deployment, we will not have the availability classes of the future, just our predictions.

In this chapter, we seek answers to the following questions for our framework:

1. How to discretize availability? How many classes and what threshold values should be used?

2. Given a set of prefixes with their associated attributes and availability classes, how accurately can one predict the availability classes of other prefixes, and which prediction models work best?

3. How to extract and represent prefix attributes from RouteViews data? Which attributes of a prefix are most important in predicting availability? For example, are more specific prefixes (ones with longer length) less available than less specific ones? Do prefixes that generate more updates have lower availability?

4. How large should a set of prefixes be such that if we learn our prediction model from this set, it will give accurate results on unseen prefixes?

5. How long should one observe prefix attributes so that its availability can be accurately inferred?

3.3 Datasets

The routing tables (RIB files) and updates are obtained from RouteViews [9], which is run by University of Oregon. Specifically, our data was taken from route-views2.oregon-ix.net which contains the Routing Information Bases (RIBs) and Updates in MRT format. The bzipped data has typical sizes of 0.8 GB per day of RIB

files (sampled every 2 hours) and about 25 MB per day of update files (written every 15 minutes), which total about 25 GB per month of data.

We preprocess the data using libbgpdump version 1.4.99.7 [170] to convert the files from the MRT format to text. We reduce the storage space required by removing unused fields. We only keep the timestamp, peer IP, prefix, and the type of update (announcement or withdrawal), except when studying additional attributes of Announcements in Section 3.5.6. After preprocessing and filtering table transfers (as described below), we have about 14-18 GB of gzipped RIB and update files per month of data.

We utilize data from January to October 2009 to build and test our prediction models. The months span a reasonable time period to prevent biasing our model selection process towards datasets from a particular timeframe when some routing event (such as an undersea cable cut) may have occurred.

A problem with using raw updates from RouteViews is that they also include routing table transfers which are caused by session resets between a monitor and a peer [171]. These spurious updates are an artifact of the update collection methodology. Zhang *et al.* [171] developed the Minimum Collection Time (MCT) algorithm to identify BGP routing table transfers by computing the collection time, which is the time it takes for most of the prefixes in the routing table to be announced. We used scripts kindly contributed by the authors to identify table transfers in our RouteViews data. We executed the table transfer identification algorithm from the point of view of every peer available in our dataset. A peer in our dataset is defined as any vantage point that is present in any routing table entry and at least one update, which ensures that we have some observations from the peer. This definition yields 41-43 peers in our dataset. The table transfer detection scripts report tuples of the form (PeerIP, Starting Time, Duration) which identifies a table transfer, as observed by the peer specified by its IP, by its starting time and its duration. We developed a script that uses this information to remove the table transfers from the update files obtained from RouteViews. We use these filtered updates for all further processing.

3.4 Methodology

We define a *combination* as a (peer, prefix) tuple, which implies that the prefix was observed by the peer in the RouteViews dataset. We compute the availability of these combinations and use that for building our prediction models. The notion of availability of a prefix is with reference to an observation point in the Internet. For the RouteViews data, these observation points are the peers. They are fairly well spread out over the world, enabling one to observe the availability of prefixes from various points in the Internet. Note that these peers are not the same as the RouteViews monitors, which passively collect data about routing tables and updates from the AS routers (peers) which actually observe prefixes. It is these peers and the prefixes they observe that we refer to as *combinations*. In what follows, a combination is *up* or *down* when the peer associated with the combination has the corresponding prefix in an announced or withdrawn state, respectively.

BGP supports aggregation of prefixes [172], and prefixes are frequently aggregated and deaggregated for implementing routing policies like traffic engineering [25]. In a routing table, there can be several prefixes which are more specific versions (subprefixes) of other prefixes in the table [173]. However, the relationship between the prefixes and their subprefixes can be complicated since these can be announced from different origin ASes. This can happen if the customer of an ISP announces a subportion of the prefix allocated to the ISP. Routing policies can change over time and the announcements of the subprefixes can vary depending upon transient conditions like network load. Misconfigurations can also cause a subprefix to be announced for a short duration, making it indistinguishable from the announcements caused by traffic engineering. Since routing policies are unknown, distinguishing the time when the prefix is unannounced because of a covering prefix or when it is withdrawn due to BGP or network conditions is difficult, and this needs to be handled by an availability metric aggregated across prefixes. Hence, computing an aggregate long-term availability of prefixes which are subprefixes of other prefixes is a challenging task. In this

work, we treat each announced prefix separately as a part of the (peer, prefix) tuple defined above. Formulation and computation of aggregate availability across more specific prefixes and their covering prefixes is left as future work.

We learn the prediction models from a *training set*, which consists of the combinations with known attributes computed during the learning period and availability class labels during the period. We then predict the availability of a disjoint set of combinations, which we call the *test set*. The disjointness is necessary to prevent overfitting [174] so that the model performs well on unseen test data and to permit a realistic evaluation of the model. After the prediction model is learned using the combinations from the training set and the information from the learning period, it is applied to the attributes of the combinations of the test set (computed during the learning period) to predict their availability classes in the future. Thus, the training and test sets are disjoint in both the combinations used and in the time period they span. If we denote the learning period as t_l and the future prediction duration as t_p, then for each test combination, we apply the prediction model to its attributes learned from t_l and we validate the availability prediction by comparing it to its availability during t_p. The learning and future prediction durations are contiguous, i.e., the prediction duration starts right after the learning duration ends.

In this work, the combinations present in the training and the test sets are randomly chosen from the set of combinations "visible" in the training and test durations t_l and t_p. We define a combination to be visible in a time duration t if it exists in the first routing table of the period t (for preventing boundary effects) or in any of the updates in the time duration. Thus, a combination has an equal chance of appearing in the training and the test sets if it appears at least once in the first routing table of t_l or an update in the period $t_l + t_p$. Since the learning period t_l and prediction period t_p are contiguous, $t_l + t_p$ represents the total time starting from the first update of t_l to the last one of t_p. This random selection of combinations prevents biasing our prediction results towards a specific group of combinations which may be related,

e.g., combinations containing prefixes from a specific AS may make it easier to predict availability of combinations containing prefixes from the same AS.

We define the *percentage learning duration* as the ratio $t_l/(t_l + t_p)$ which evaluates the percentage of the duration $t_l + t_p$ that is used in learning. The larger this ratio, the easier the prediction since less of the future is unknown. We evaluate the quality of our prediction models by varying this ratio among 0.1, 0.25, 0.5, 0.75, and 0.9. For each of the values of this ratio, we experiment with values of t_l, where t_l = 1, 7, 19 and 30 days. Thus, we have 20 data points for evaluating each prediction model. The rationale behind this is that the availability distribution may be different when computed over different periods of time. We want to investigate this difference and the effect it has on prediction for the same values of $t_l/(t_l + t_p)$, but different values of t_l.

The prediction models considered in this work are described in detail in Section 3.5. We use Weka [174], a Java open-source data mining software, for evaluating the models. Weka provides implementations of standard prediction models and data mining techniques for analyzing the performance of the models.

3.4.1 Discretizing Availability

We discretize the continuous availability value into availability classes which we predict using observed attributes. The process of discretization uses thresholds as parameters, the number and values of which have to be decided. The choice of these parameters is based on the prediction goal. If one aims to find prefixes that do not meet high availability requirements, a single threshold can discretize availability into *high* and *low* classes. If one aims to find prefixes which have both high and low availability values, one should use two thresholds to discretize availability into *high*, *medium*, and *low* classes.

The computation of the availability of a combination for a particular time period proceeds as follows. The first routing table of the period is used to initialize the

state of each combination present in the table to up (or announced). The learning duration of t_l considers all the combinations found in the first routing table of January 2009 and in the updates recorded in the duration t_l. We maintain the state of each combination at each point in time, and at the time of each state change (as indicated by an update), we record a downtime or an uptime. If the state of a combination changes from Announced (A) to Withdrawn (W), an uptime is recorded, whereas a change from W to A leads to the recording of a downtime. After processing all update files, we add an extra up or downtime depending upon the last state of the combination. For example, if the last state change was to W and was reported at time t_1, and if the data period ended at time t_2, we add a downtime with value $t_2 - t_1$. The availability of the combination is computed by noting the time that the combination was up (cumulative uptime) divided by the total time in which we are interested. Hence, a combination that only appears in the first routing table of the month and has no updates for the duration under consideration will have an availability of 1.

We use data from January 2009 to study the effect of discretization. Table 3.1 shows the availability statistics for four values of t_l starting from the beginning of January 2009 (i.e., t_l=19 days means data from Jan. 1 to Jan. 19). The second column shows the number of (non-trivial) availability values that are considered in computing these statistics, where one value corresponds to one combination. The first quartile, median, and third quartile are the values below which 25%, 50%, and 75% of the availability ordered combinations lie, respectively. Only the first quartile is shown in the table since the median and 3^{rd} quartile are 1 for all t_l. The table shows a steady increase in the number of combinations as more days are considered, since previously undiscovered combinations are found in newer update files. These new combinations were not present in the first routing table of the month; otherwise they would have been found for $t_l = 1$ day. These are expected to be low availability combinations. This is validated by the fact that the first quartile and mean of the combinations show a decreasing trend with these newly added combinations. The variance of the

availability increases as lower values are added to the set of predominantly higher availability values.

Table 3.1
Availability statistics of January 2009 for different values of t_l.

t_l	Number of Combinations	1^{st} Quartile	Mean	Variance
1 day	10545170	1	0.9975	0.0018
7 days	10700675	1	0.9897	0.0078
19 days	10959231	0.999988	0.9743	0.02041
30 days	11476218	0.999882	0.9604	0.02966

This trend of lower availability values with longer durations motivates us to study four different values of t_l with the same $t_l/(t_l+t_p)$ ratio. The difference in availability distributions for durations t_l and t_p not only depends on the value of $t_l/(t_l+t_p)$, but also on the value of t_l. This effect is seen in Table 3.2 which shows the percentage difference in the mean availability of the learning and test durations (t_l and t_p) for different values of t_l and the same value of $t_l/(t_l+t_p)$.

Table 3.2
Percentage difference of mean availability between the training and test sets for different t_l, $t_l/(t_l+t_p)$=0.1

t_l	Mean availability of learning duration t_l	Mean availability of test duration t_p	% Difference in availability of t_l w.r.t. t_p
1 day	0.9975	0.9853	-1.22
7 days	0.9897	0.9204	-6.99
19 days	0.9743	0.8367	-14.13
30 days	0.9604	0.7996	-16.74

These statistics play an important role in the choice of discretization thresholds. To study this, we start with a ternary class label (values *high, medium,* and *low*), and choose two different threshold sets of (0.99, 0.50) and (0.99999, 0.50), with the higher threshold demarcating *high* and *medium* and the lower one differentiating the *medium* and *low* classes. This enables us to compare the percentage share of *high* under the two threshold sets, which is listed in Table 3.3. The *medium* percentage can be easily calculated since the percentages of *high, medium,* and *low* add up to 100%. If we choose a relatively lower valued threshold for *high,* e.g., 99%, the class distribution will be highly skewed, with most combinations (around 91-94%) having *high* availability. With a 0.5 threshold for the *low* class label, about 1-4% of combinations fall into that category. However, the prediction problem is more difficult with a 0.99999 threshold for *high* than with 0.99, since there is a higher chance of combinations that have *high* availability in the learning period to fall below the 0.99999 threshold in the test period. We verified this observation by evaluating the prediction models of Section 3.5 on datasets with the two thresholds for the *high* class and found that the model performance for 0.99 threshold is indeed higher than that with 0.99999, validating that the former is an easier prediction problem. Based on these observations and the significance of "five nines" availability [175], we use a single threshold of 0.99999 and a binary class label. However, to find combinations with very low availability, we can easily extend our framework to two thresholds and a ternary class label.

Table 3.3
Class distributions when discretizing availability

t_l	% *High* with 0.99 Threshold	% *High* with 0.99999 Threshold	% *Low* with 0.5 Threshold
1 day	93.89%	67.92%	1.02 %
7 days	93.09%	67.25%	1.68%
19 days	91.89%	66.19%	2.74%
30 days	91.09%	66.13%	3.59%

3.4.2 Computing Attributes

We now investigate the attributes of the (peer, prefix) combinations to be extracted from the RouteViews data. The attributes are computed for the learning period with the aim of predicting (future) availability classes for the test set. Our goal is to compute the attributes from publicly available information from RouteViews, which contains both routing tables and updates for various combinations. We choose not to use the routing tables because they provide time snapshots of prefixes which can be reached by peers, and we are interested in availability, which is a continuous time metric. The updates collected from RouteViews have the advantage that (barring errors) all the updates for a particular combination will be recorded. Knowing the announcement and withdrawal times for a combination, we can easily compute its availability. Comparing this computed availability with the predicted availability validates prediction results.

The attributes of a combination are selected to relate to its availability (Section 3.4.3), and to be easily computable given the observed updates for the learning period so that the learning system is fast. It is important to note that the attributes we select do not necessarily cause high/low availability; we are looking for correlation *not causality*. Correlation is sufficient for a prediction model to be successful.

We hypothesize that longer prefixes will have lower availability since they represent smaller networks which are more likely to go up or down. From [57], it is known that popular destinations, which are expected to have high availability, are stable, i.e., have fewer updates. Hence, in addition to prefix length, we also compute update frequency, which is the average number of updates observed for the combination in a time window of one hour (averaged over the learning period). The period of one hour is chosen so that the update frequency numbers are neither too large nor too small.

Furthermore, by recording the time when a combination goes up/down, we compute two additional attributes, mean uptime and mean downtime, called the Mean Time to Failure (MTTF) and Mean Time to Recovery (MTTR), respectively. It is

important to note that MTTF and MTTR are computed for the learning period, and hence the predicted availability for the time-disjoint test set is not a direct function of these values.

In summary, we compute the following attributes for the learning period from routing updates observed through RouteViews: (1) Prefix length, (2) Update frequency, (3) Mean Time to Failure (MTTF), and (4) Mean Time to Recovery (MTTR).

We opt not to use information about to which AS a prefix belongs or the AS path to a prefix in this work. This is because we want to keep our prediction model free from constraints of specific ASes or AS paths that can change. We defer the investigation of how prefixes are similar across the same AS or neighboring ASes in the AS topology to Chapter 4.

Although we compute the attributes of every combination with at least one recorded uptime or downtime, we downsample this set of combinations (of about 11 million as in Table 3.1) to a set of 10,000 combinations with their attributes, and use that to build and test models. Downsampling does not significantly affect the accuracy of models since prediction models typically learn well with a few hundred instances. We evaluate the performance of the models with increasing number of learning instances and on larger test sets in Section 3.5. An advantage of downsampling is the computational efficiency of building and testing the models.

3.4.3 Demarcating Availability using Attributes

In this section, we quantify whether the four attributes discussed in the last section indeed convey information about the availability class. We divide the 10,000 combinations into ones that have *high* and *low* availability for the month and compute statistics for the attributes of each of the two groups. We show the means and variances of all the attributes for a typical value of $t_l = 19$ days in Table 3.4. The results for other values of t_l were similar.

Table 3.4
Attribute statistics of each class for learning period of $t_l = 19$ days

Attribute	*High* Class		*Low* Class	
	Mean	Variance	Mean	Variance
Prefix length	22.04	6.41	22.72	4.25
MTTF (s)	1587480	3.76E+10	777844	2.92E+11
MTTR (s)	0.201002	3.52	58882.2	4.57E+10
Update frequency (/hr)	0.0244	0.7339	0.0795915	0.5694

We use the paired t-test to test for equality of the means of each of the attributes of the two classes. We employ the Welch t-test [176, 177] which assumes that the two populations have normal distributions and their variances cannot be assumed to be equal (which is true for our data). The normality assumption is valid due to the Central Limit Theorem (CLT) and because we have about 3000-7000 samples in each class. We find that the means of each of the four attributes are significantly different at 1% significance level for each of the four learning periods. This shows that the attributes show a statistically significant correlation with the availability class labels. For most of the attributes, their variances for the *low* class are higher because the class covers a wider range of availability values.

Our intuition that the combinations with longer prefix lengths have lower availability is confirmed. The mean prefix lengths of the *high* and the *low* availability classes usually differ by about 0.7, or about 3% (which is statistically significant) while the median and first quartile differ in length by 1 and 2 respectively, with the higher value for the *low* class. The consistency of the results across each of the four values of t_l is convincing of the correlation between prefix length and availability class. We conjecture that this is because shorter prefixes represent larger, more stable networks while small portions of the address space can be announced and withdrawn

frequently for multihoming or load balancing purposes. Further, it is more likely that a longer prefix representing a smaller network goes down than a larger network.

The MTTF of a *high* availability combination is higher than that of a *low* availability one by about 85% on average, whereas the MTTR is almost 100% lower. The difference becomes larger as t_l increases. This result is intuitive: a *high* availability prefix has a long uptime before it fails, and when it does fail, it quickly comes back up (well within one second on average). The average frequency of updates observed for a *high* availability prefix is about 77% lower than for *low* availability ones. The maximum frequency of updates observed has an even larger difference, about 2 updates/hour for the *high* vs. about 14 updates/hour for the *low*. These results are explained by the fact that a high availability combination stays up for a long period of time, and hence has fewer updates. The difference in attribute values of the *high* and *low* classes increases with t_l, showing that these attributes correlate well with the availability class since availability computed over a longer duration is more indicative of the actual availability.

Assuming update frequency distribution in each month is a normally distributed random variable (valid because of CLT), we construct a 99% t-Confidence Interval (CI) for the average update frequency of a combination. The mean update frequency of a combination, averaged over all 11.5 million combinations of Table 3.1 is about 0.03/hr and the variance is about 0.28. The upper bound of the CI is computed to be about 1.4 updates an hour. Thus, if we observe more than an update for a combination in about 43 minutes, on average, we are 99% certain that it will have *low* availability.

The conclusion from this section is that the selected prefix attributes perform well in demarcating the availability classes. The correlation of the attributes with the availability class is consistent with our intuition.

3.4.4 Learning and Evaluation

We learn several models in this chapter to predict the availability class of combinations. The performance of each model is studied using *n-fold incremental cross-validation*. In this technique, the dataset is divided randomly into n parts, called *folds,* while maintaining the class distribution of the dataset in the fold (i.e., *supervised sampling*). The model is then learned using the known attributes and class labels of n-1 folds (called the *training set*), and applied to predict the class labels of the remaining fold (the *test set*). Each fold is left out at a time, resulting in n learned models and corresponding performance results. The training and the test sets are disjoint in order to get an unbiased estimate of model error. The algorithm is run k times, each time with a different random seed so that different n folds are constructed in each run. Thus, for each training set size, we have nk performance values, and we report the mean value.

As the number of instances to learn a model increases, the model performance on test data typically improves, but with diminishing returns. We study this using *learning curves.* A model is successively learned using increasing training set sizes (from each of the n training sets) and its performance on the test set is plotted against the training set size. A typical shape of a learning curve is an increasing exponential; the performance increases, and then flattens after a certain number of instances is reached.

We now describe the performance metrics used to evaluate a model when it is applied to the test set. Any classification algorithm can be studied using a confusion matrix, as shown in Table 3.5, which gives all possible combinations of the true and predicted class. In what follows, the class label *high* is treated as a *positive* class, and the label *low* is treated as a *negative* class. True Positives (TP) are commonly referred to as *hits*, true negatives (TN) as *correct rejections*, false positives (FP) as *false alarms* and false negatives (FN) as *misses* [178].

Table 3.5
Confusion Matrix with class label *high* as positive and class label *low* as negative

True Class			
		High (Positive)	*Low* (Negative)
Predicted Class	*High* (Positive)	True Positives	False Positives
	Low (Negative)	False Negatives	True Negatives

The confusion matrix can be used to compute several performance measures, the most common of which is *accuracy*, defined as: $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$, where TP and TN are the true positives and negatives respectively, and FP and FN are the false positives and negatives respectively. The True Positive Rate (TPR) and the False Positive Rate (FPR) are defined as: $TPR = \frac{TP}{P} = \frac{TP}{TP+FN}$, and $FPR = \frac{FP}{N} = \frac{FP}{FP+TN}$.

The *Kappa statistic* measures the agreement between predicted and observed values, correcting for agreement that occurs by chance. It is computed as: $\kappa = \frac{P(o)-P(e)}{1-P(e)}$, where $P(o)$ is the proportion of observed agreement between the observed and predicted values, and $P(e)$ is the proportion of times the values are expected to agree by chance. Complete agreement corresponds to $\kappa = 1$, which will be the best predictor, whereas $\kappa = 0$ for a random predictor, and κ = -1 indicates complete disagreement between the values.

Unfortunately, confusion matrix-based measures can be misleading with a skewed class distribution, which happens when the proportion of *high* availability (positive) and *low* availability (negative) instances in the sample are unequal. For example, a trivial algorithm which predicts every availability value as *high* will have 90% accuracy on a dataset which has 90% *high* values. The measures use data from both columns of a confusion matrix (two rightmost columns of Table 3.5), and hence are sensitive to the proportion of instances in the two columns [179]. From Table 3.3, we observe that there can be significant class skew, which render these measures inappropriate. A better metric is obtained by using Receiver Operating Characteristic

(ROC) curves [174], which plot the TPR versus the FPR. ROC curves are independent of class skew because they use a strict columnar ratio from the confusion matrix [178, 179]. We use the Area Under the ROC Curve (AUC) as a performance metric. The AUC of a classifier is equivalent to the probability that it will rank a randomly chosen *high* instance higher than a randomly chosen *low* instance. A perfect classifier has an AUC of 1.

We compare the results from our prediction models to those obtained using a random classifier, which acts as a baseline for comparison. A random classifier randomly chooses either of the class labels with equal probability. Such a classifier has an AUC of 0.5, since it has about as many TPs as FPs. The reason for comparison to a random classifier is that we need to be sure that any learning-based model performs better than the random classifier. Otherwise, one could effectively toss a coin and decide the class label, making a trivial predictor the best one.

While ROC curves work well for most classifiers, they are not directly applicable for models that do not produce any ranking of instances in terms of probabilities of being classified as *high* and *low*. This is because one plots a ROC curve by varying the threshold that decides between *high* and *low*. This enables the model to produce different classifications resulting in various (FPR, TPR) points in ROC space. A model which does not produce instance ranking has no threshold to vary; hence, it gives a single point in the ROC space instead of a curve. For such a model, an option is to randomly order the instances predicted as *high* and *low*, and then rank them to produce a ROC curve. We describe the details of this scheme in Section 3.5.1.

3.5 Model Evaluation

In this section, we study three prediction models using the metrics in Section 3.4.4. As mentioned in Section 3.4.2, we work with 10,000 combinations and their attributes, downsampled from the set of all combinations. We do 10-fold incremental cross-validation as described in Section 3.4.4; thus n=10. We conduct k=5 runs, generating

a different set of 10 folds each time. Hence, we have 50 performance measures for each model averaged to give an output measurement.

We start with a simple baseline prediction model in Section 3.5.1. This model does not learn based on other combinations, and simply predicts the availability of one combination at a time. We then investigate more sophisticated machine learning based models.

3.5.1 Simple Prediction

The simplest approach to predict the availability of a combination is based on the simplistic assumption that the future is the same as the past. The *past availability* of a combination is its availability during the learning period t_l. This prediction approach does not learn a model based on other combinations, but merely predicts the same availability for a combination as the discretized value of its *past availability*. Thus, if the past availability exceeds 99.999%, the predicted class label is *high*, otherwise it is *low*.

This is a model where no instance ranking is performed; only hard classifications are made. Therefore, we compute confusion matrix-based measures. These measures, computed for various values of t_l and $t_l/(t_l + t_p)$ and averaged over $nk = 50$ runs, are listed in Table 3.6.

The results show that while the TPR of the simple model is high, its FPR is high as well. However, this simple classifier outperforms a random classifier (as indicated by the κ statistic) and hence forms a baseline model to which other sophisticated models can be compared. As $t_l/(t_l + t_p)$ increases, the prediction problem becomes easier as more data is available for learning. Hence, the accuracy of the model increases, while its FPR reduces. As t_l increases, the availability distribution becomes more diverse and hence the model typically performs worse.

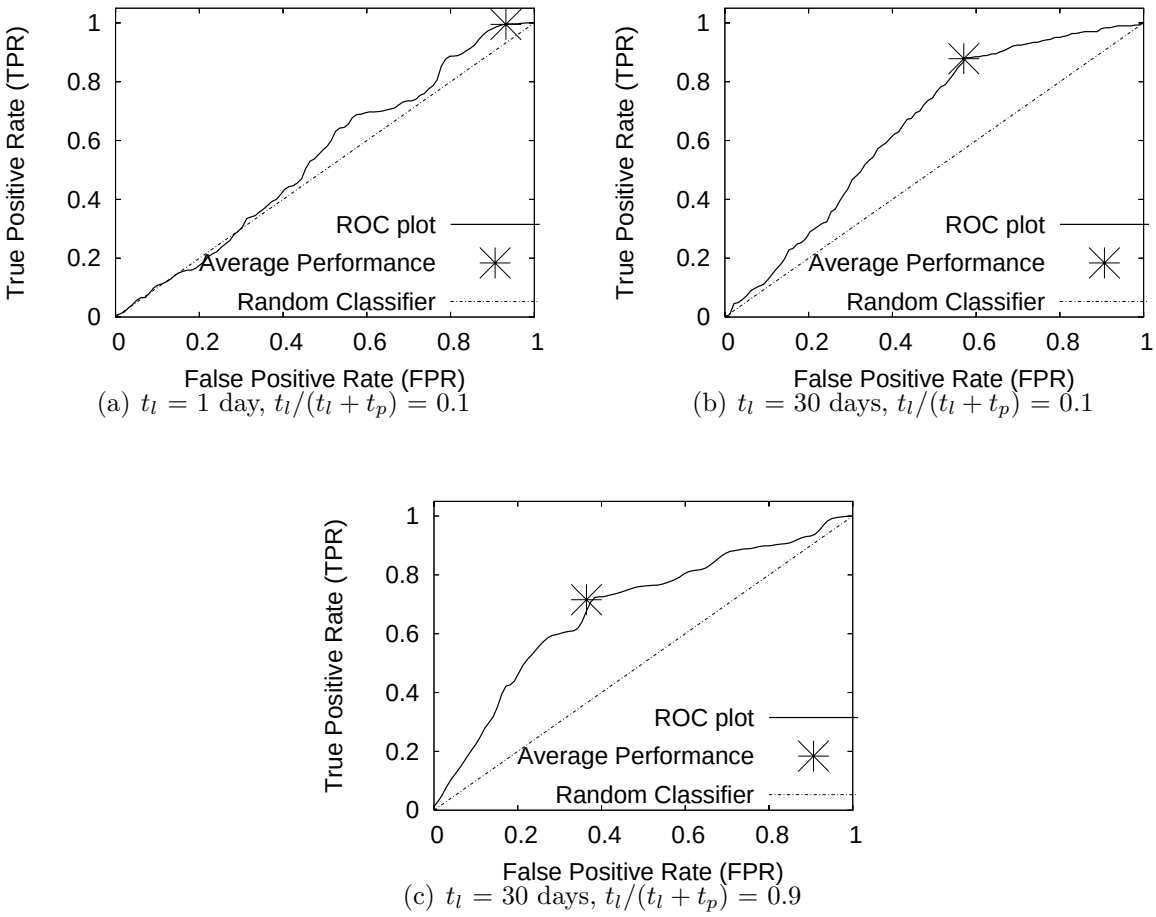
We now use ROC-based metrics to evaluate this classifier. The model gives a single point in the ROC space (since it does not perform instance ranking), so we modify the

Table 3.6
Results of the simple prediction model

t_l	$t_l/(t_l+t_p)$	Accuracy (%)	TPR	FPR	κ	AUC
1 day	0.1	88.60	0.9950	0.9322	0.1022	0.5261
	0.25	96.79	0.9940	0.8085	0.2641	0.5877
	0.5	97.99	0.9928	0.7160	0.3207	0.6224
	0.75	98.69	0.9911	0.5670	0.3175	0.6900
	0.9	98.90	0.9900	0.4766	0.1803	0.7272
7 days	0.1	60.02	0.9717	0.8451	0.1353	0.5599
	0.25	73.87	0.9502	0.8013	0.1928	0.5774
	0.5	83.98	0.9421	0.7410	0.2403	0.5962
	0.75	89.37	0.9271	0.7242	0.1575	0.5813
	0.9	91.22	0.9224	0.5907	0.1281	0.6713
19 days	0.1	54.10	0.9107	0.6777	0.1917	0.6163
	0.25	67.98	0.8933	0.6281	0.2816	0.6326
	0.5	76.01	0.8620	0.5315	0.3481	0.6641
	0.75	78.30	0.8201	0.4652	0.2726	0.6748
	0.9	78.66	0.7953	0.4082	0.1355	0.7015
30 days	0.1	57.17	0.8613	0.5720	0.2242	0.6414
	0.25	65.53	0.8379	0.5231	0.3133	0.6548
	0.5	70.81	0.7961	0.4752	0.3242	0.6606
	0.75	73.45	0.7544	0.3723	0.2955	0.6895
	0.9	71.08	0.7154	0.3641	0.1346	0.6728

algorithm to draw a ROC curve. We take a typical run of the model with confusion matrix measures close to their average values. The instances which are classified as *high* and *low* by the model are randomly reordered within their respective groups, and then the instances are ranked with the (predicted) *high*s higher than the *low*s.

We vary the prediction threshold, and record the TPR and FPR for each threshold, as in Algorithm 2 of [179] to compute the points on a ROC curve.



(a) $t_l = 1$ day, $t_l/(t_l + t_p) = 0.1$

(b) $t_l = 30$ days, $t_l/(t_l + t_p) = 0.1$

(c) $t_l = 30$ days, $t_l/(t_l + t_p) = 0.9$

Fig. 3.1. ROC plots for the simple prediction model.

The ROC curves for the simple prediction model for some typical values of t_l and t_p are depicted in Figure 3.1. The plots show the original model performance (in Table 3.6) as a point ("star") on the ROC plots, along with the performance of a random classifier. The performance of simple prediction is clearly better than a random classifier for most cases, but there are occasions when it performs as good as or slightly worse than a random one as in Figure 3.1(a). This is especially true when

t_l is small, and hence future availability is quite different from past availability. As the average accuracy in Table 3.6 is reasonably high, this emphasizes the inadequacy of accuracy as a metric to evaluate performance models. Hence, we use ROC metrics, like area under the ROC curve (AUC). The AUC is computed, using Algorithm 3 of [179], for a typical run (confusion matrix based measures close to their average values of 50 runs). Because of inherent randomness in reordering and ranking the instances, the typical run will give different AUC values when run with different random seeds; the average of 50 different AUC values is reported in Table 3.6.

The results highlight the importance of ROC curves. For example, classifier A for $t_l = 30$ days, $t_l/(t_l + t_p) = 0.1$ (Figure 3.1(b)) is worse than classifier B for $t_l = 30$ days, $t_l/(t_l + t_p) = 0.9$ (Figure 3.1(c)) using AUC as the metric. However, examining the ROC curve, we see that for higher FPRs (around 0.8), classifier B outperforms classifier A. The overall inferior performance of classifier A is because it performs similar to a random classifier for low FPRs. Hence, if our operating region is at low FPR, classifier B is better, whereas classifier A is better for high FPRs.

3.5.2 Naive Bayes Model

The Naïve Bayes model predicts a *high* or *low* class label using the attribute vector $\mathbf{X} \equiv \{X_1, X_2, \ldots, X_m\}$ based on Bayes rule [174]. It makes the "naïve" assumption that the attributes are conditionally independent given the class label. However, the model is often used even when its assumption does not hold due to its simplicity. The model computes, for each instance, the probability of each class label given its attribute set and the independence assumption, using the training set to estimate $P(X_i|C)$ and $P(C)$, where C is the class label. Hence, instance ranking is naturally produced by the model which can be used to produce ROC curves.

We evaluate the model on each of the values of t_l and t_p using learning curves. The model is learned on increasing size training sets, and its performance is evaluated on the 10 different test sets produced by incremental cross-validation. We plot a

typical learning curve in Figure 3.2, using both accuracy and AUC as performance measures. The plots for the other time durations lead to similar conclusions. The accuracy initially increases at a fast rate when the number of training instances is increased, and tapers off afterwards. However, the AUC remains relatively stable with the increase in number of training instances. In what follows, we use the entire training set to train the Naïve Bayes model to achieve the maximum accuracy without sacrificing AUC. This ensures that the model is trained to its potential.

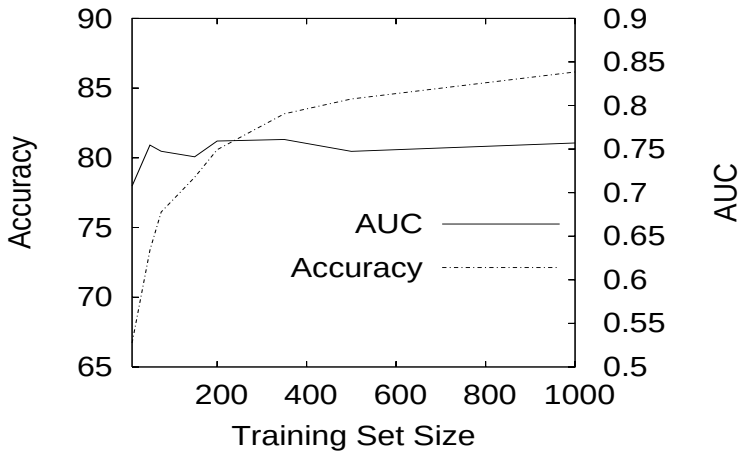


Fig. 3.2. Naïve Bayes learning curves for t_l=30 days, $t_l/(t_l + t_p)$=0.9

We now compare the Naïve Bayes model to the simple prediction model of Section 3.5.1. We use the accuracy and AUC as measures for comparison. The results are given in Table 3.7. The results show that the Naïve Bayes model yields a higher AUC than the simple model for all cases. The accuracy values of Naïve Bayes are close to those of the simple model, except when learning from 30 days of data, where for a smaller prediction period t_p, the accuracy is significantly better with a high variance (around 26.3), while for a higher prediction period t_p, the accuracy is significantly lower with a low variance (around 2.3). This is because this model assumes that the attributes are conditionally independent given the class label. The model uses the frequencies in the training set as estimates of the probability distributions

of the attributes. These estimated distributions are valid only when the period of parameter estimation, i.e., learning period, is not too different from the prediction period. When $t_l = 30$ days, the period of the training and test sets differ by a few days to months (except when $t_l/(t_l+t_p) = 0.5$) and hence have different distributions. This leads to different accuracies since this metric is highly dependent on class skew.

We consider the better metric, AUC, and investigate whether the higher AUC values of the Naïve Bayes model are statistically significant. If so, Naïve Bayes would be a better prediction model than the simple model. We use the Welch t-test [176,177] to test for equality of the performance measures (means) of the distributions of the two samples (simple and the Naïve Bayes). We perform the test on the AUCs of the two models for each of the four months, using the mean values shown in Tables 3.6 and 3.7, and the sample variances computed using the $nk = 50$ data points. We compute the degree of freedom ν using the Welch-Satterthwaite equation, and round it to the nearest integer for t table lookup using [180]. We find that the null hypothesis of equality of the means is rejected for every month at 5% significance level. This means that the AUC of the Naïve Bayes model indeed exceeds that of the simple model at 5% significance level. Table 3.8 shows the details of the test for some typical values of t_l and t_p.

Finally, we compare the Naïve Bayes model to the simple model using ROC curves. The plot for $t_l = 30$ days and $t_l/(t_l + t_p) = 0.1$ is illustrated in Figure 3.3. The figure shows that the Naïve Bayes model dominates the simple model throughout most of the ROC space. For the same FPR, its TPR is higher and hence it is closer to the ideal point in ROC space. The implication of these results is that a model which learns based on other prefix combinations like the Naïve Bayes classifier will typically outperform prediction without learning, despite its naïve assumptions. This confirms that availability is predictable using the attributes we measure. It is also worth noting that this better performance in terms of TPR and FPR in the ROC space again points to the inadequacy of accuracy as a metric: even though the Naïve Bayes model has

Table 3.7
Results with Naïve Bayes model and % change from simple model

t_l	$t_l/(t_l + t_p)$	Accuracy (%)	% Change in Accuracy from Simple Model	AUC	% Change in AUC from Simple Model
1 day	0.1	88.51	-0.09	0.6044	14.89
	0.25	96.70	-0.09	0.6568	11.76
	0.5	97.94	-0.045	0.7097	14.02
	0.75	98.66	-0.034	0.7924	14.84
	0.9	98.82	-0.074	0.8159	12.19
7 days	0.1	59.85	-0.288	0.6341	13.25
	0.25	74.20	0.444	0.6290	8.96
	0.5	84.06	0.10	0.6355	6.59
	0.75	87.89	-1.65	0.6473	11.35
	0.9	89.95	-1.39	0.6990	4.12
19 days	0.1	54.61	0.94	0.6761	9.70
	0.25	68.04	0.08	0.6956	9.95
	0.5	76.09	0.10	0.7173	8.01
	0.75	77.35	-1.21	0.7173	6.31
	0.9	77.38	-1.63	0.7304	4.12
30 days	0.1	46.14	-19.29	0.6930	8.04
	0.25	59.23	-9.61	0.7009	7.04
	0.5	70.29	-0.73	0.7009	6.10
	0.75	80.03	8.95	0.7394	7.24
	0.9	83.40	17.33	0.7538	12.05

much lower accuracy than the simple model for these values of t_l and t_p, it is better in ROC space.

Table 3.8
Paired t-test results of comparing AUC of Naïve Bayes model and the simple model

t_l	$t_l/(t_l+t_p)$	Statistic Value	ν	t-value for 5% Significance
1 day	0.1	15.80	98	1.984
7 days	0.25	8.46	64	1.998
19 days	0.5	8.75	98	1.984
30 days	0.75	7.28	81	1.99

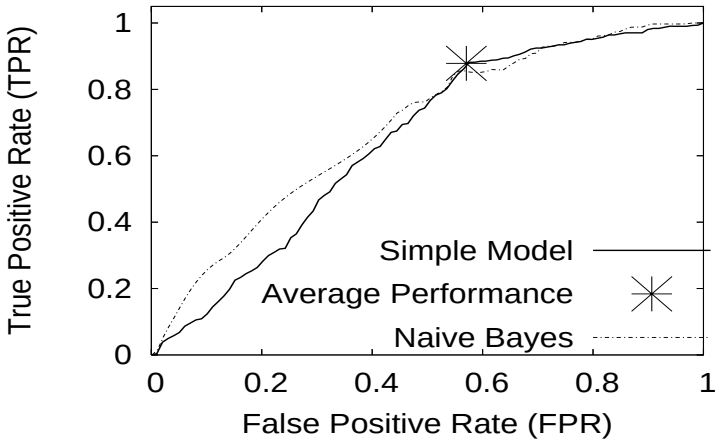


Fig. 3.3. ROC plots for Naïve Bayes and simple model for t_l=30 days, $t_l/(t_l+t_p)$=0.1

3.5.3 Decision Trees

A *decision tree* is a recursive divide-and-conquer classifier, which divides the instances based on one attribute at a time in a top-down fashion until the leaves of the tree are reached [174]. The decision to split on an attribute is typically used to maximize some metric of information gain, so that the splitting of instances will lead to increased clarity on the class labels of the instances themselves. This classifier has the advantage that it is interpretable, since the attributes of the classifier are ranked

from the root node downwards in the order of importance, and rules to classify an instance can be read off the decision tree. We use the C4.5 algorithm developed by Quinlan [181] to build decision trees, which uses reduction in entropy when splitting the instance set S based on an attribute A as the information gain metric to build the tree. The entropy is a measure of randomness of a random variable and is defined by:

$$Entropy = -\sum_{x \in \mathbf{X}} p(x) log_2 p(x) \tag{3.1}$$

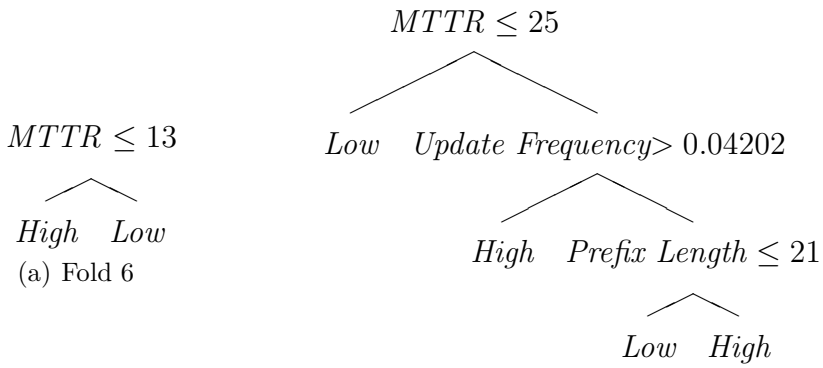
A reduction in entropy, achieved by splitting the instance set S into two (or more) parts based on an attribute A, is used as the information gain (IG) criterion.

$$IG = Entropy(S) - \sum_{v \in values(A)} \frac{|S_A|}{|S|} Entropy(S_A) \tag{3.2}$$

Pruning the tree is necessary to avoid overfitting to the training data, and for constructing a general enough tree to perform well on unseen test data. In Weka, the J4.8 classifier implements the C4.5 algorithm [174], and one can choose to consider the unpruned tree, or prune it based on different criteria. C4.5 pruning (the default) uses an estimate of the error on the training set. An alternative is to use Reduced Error Pruning (REP) [182], which holds back some of the training data as a fold and uses that to estimate the error. The advantage of REP is that it can lead to more unbiased estimates of the error; the disadvantage is that it uses less data for tree building.

We use the unpruned, C4.5-pruned, and REP trees, and find that the accuracy and AUC metrics are not significantly different among them. However, at very small training set sizes, holding out instances for REP can lead to insufficient training data, which results in lower AUC. Nonetheless, we decided to use REP because of the advantages of a tree which avoids overfitting and because we will work with sufficiently large datasets. We observe that our results have a high variance. This is a typical property of decision trees, since a small difference in the training data can cause different branches to be constructed. For example, with 200 training instances in each of the 10 folds, we find decision trees with different structure and attribute

values (two are shown in Figure 3.4). The right branches of all nodes are for a "Yes" decision and the left branches are for a "No" decision. While the decision trees shown all use *MTTR* as their root node, different trees use different numbers and values of attributes to make decisions. This increases variance in classification results, causing mean results to appear worse.



(a) Fold 6

(b) Fold 8

Fig. 3.4. Decision trees for $t_l = 30$ days, $t_l/(t_l + t_p) = 0.1$ constructed with 200 training instances.

A method to reduce the variance of decision trees is to use *bootstrap aggregating (bagging)* [174]. Bagging combines an ensemble of unstable, high variance, predictors into a stable predictor. We apply the bagged decision tree classifier to predict availability with the underlying baseline classifier chosen to be decision trees with REP. Ten decision trees are learned for each of the 10 folds of the dataset, and they are then voted on to produce the *high* or *low* class label. The learning curve for a typical case is shown in Figure 3.5. The curve demonstrates that the performance measures flatten with increase in training set size, which confirms that pruning is successful in preventing overfitting.

We now apply the bagged decision tree model learned from the entire training dataset of around 9000 combinations to predict availability for the values of t_l and t_p considered earlier. The average results over $nk = 50$ points are given in Table 3.9. As before, we perform significance tests, and find that AUC increases for $t_l = 1$ and

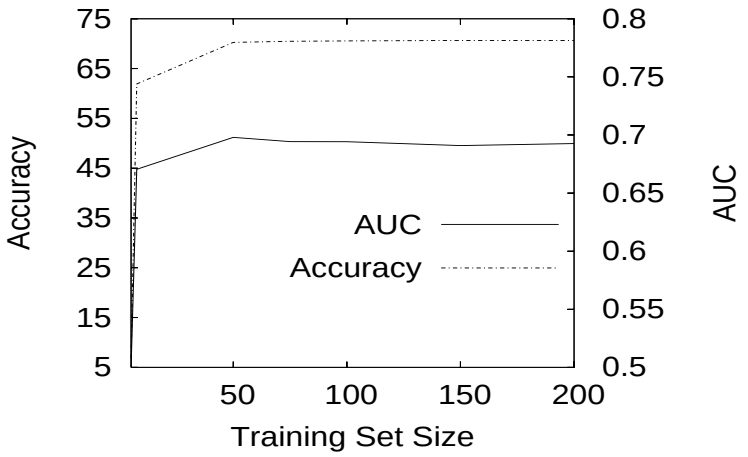


Fig. 3.5. Learning curve for bagged decision trees, $t_l = 30$ days, $t_l/(t_l + t_p) = 0.1$

7 days are significant at 5% significance level, except for $t_l/(t_l + t_p) = 0.9$ for $t_l =$ 1 day, and $t_l/(t_l + t_p) = 0.75$ and 0.9 for $t_l = 7$ and 19 days. The results reveal that bagged decision trees perform well w.r.t. Naïve Bayes when the learning period is shorter (up to a couple of weeks) and the prediction period is longer, i.e., when $t_l/(t_l + t_p)$ is small. This is because as diversity of the data increases, the bagged decision trees adapt to the diversity by building complex trees, which do not generalize well to future datasets. This cannot be corrected by pruning since the diversity is in the time domain and occurs in nearly every combination, so holding out a set of combinations for pruning does not necessarily help.

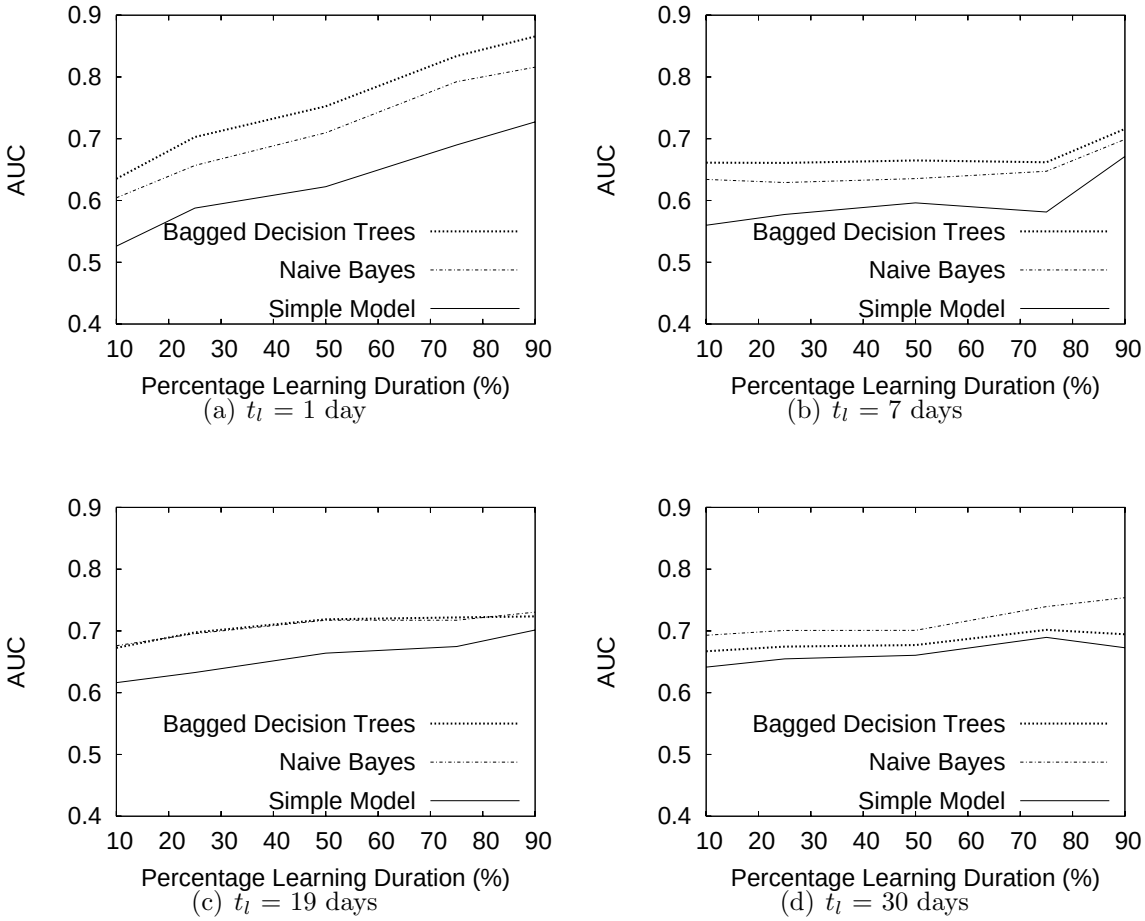
3.5.4 Learning Duration

We now study the effect of learning duration on the prediction results of all the models we have considered. There are two facets to this problem: the learning duration as a percentage of the overall period of interest, i.e., $t_l/(t_l + t_p)$, and the value of the learning duration itself. Lowering the percentage learning duration means that we have a shorter time to learn the attributes of various combinations, leading

Table 3.9
Results with bagged decision trees and % change from Naïve Bayes model

t_l	$t_l/(t_l + t_p)$	Accuracy (%)	% Change in Accuracy from Naïve Bayes Model	AUC	% Change in AUC from Naïve Bayes Model
1 day	0.1	87.81	-0.80	0.6352	5.10
	0.25	95.54	-1.21	0.7027	6.99
	0.5	96.61	-1.37	0.7525	6.04
	0.75	97.22	-1.46	0.8339	5.24
	0.9	97.36	-1.48	0.87	6.09
7 days	0.1	60.24	0.67	0.6613	4.29
	0.25	74.91	0.96	0.6609	5.05
	0.5	83.42	-0.76	0.6648	4.60
	0.75	87.41	-0.55	0.6619	2.26
	0.9	90.23	0.32	0.7159	2.41
19 days	0.1	54.95	0.6147	0.6726	-0.52
	0.25	68.35	0.46	0.6976	0.28
	0.5	75.96	-0.17	0.7188	0.21
	0.75	77.45	0.13	0.7218	0.62
	0.9	76.63	-0.96	0.7235	-0.94
30 days	0.1	56.83	23.17	0.6671	-3.73
	0.25	65.24	10.15	0.6745	-3.75
	0.5	70.85	0.79	0.6771	-3.39
	0.75	73.44	-8.23	0.7018	-5.09
	0.9	70.69	-15.23	0.6945	-7.87

to a reduction in prediction accuracy, whereas increasing this percentage improves prediction results, since there is more information available.



(a) $t_l = 1$ day (b) $t_l = 7$ days

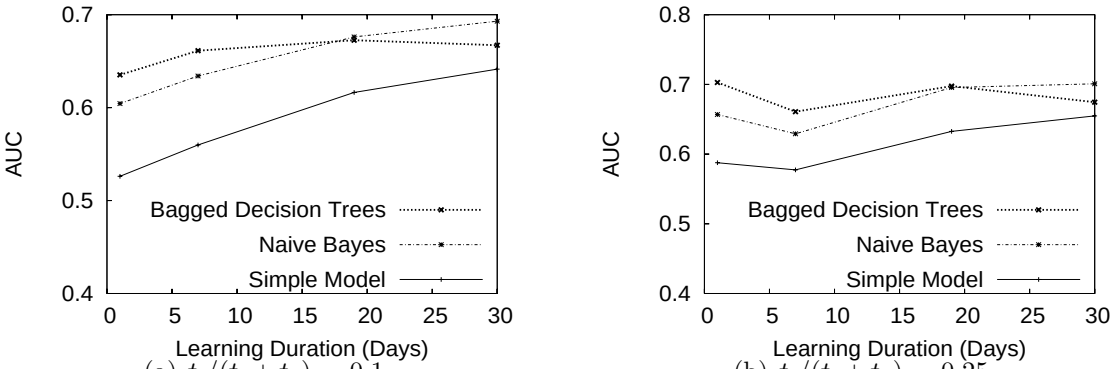
(c) $t_l = 19$ days (d) $t_l = 30$ days

Fig. 3.6. Effect of percentage learning duration $t_l/(t_l + t_p)$ on prediction performance for different values of t_l

The plot of AUC against percentage learning duration for various values of the learning duration t_l is shown in Figure 3.6. The results show that the prediction performance gracefully degrades as the amount of data available for learning is reduced. The decrease is much more steep when the learning duration t_l is low, e.g., 1 day, and this effect almost disappears when t_l reaches 30 days. This result implies that one can predict long-term availability by learning from only a short learning period,

as long as the period spans a few days, e.g., a week. This gives further credence to the feasibility of availability prediction. The bagged decision tree model performs the best for all learning duration percentages when the learning duration t_l is less than around 3 weeks. Beyond that value of t_l, Naïve Bayes performs best.

We also plot the change in the AUC for each of the models when increasing the learning duration t_l, keeping the percentage learning duration $t_l/(t_l + t_p)$ constant. Two typical plots are shown in Figure 3.7. The plots show that for the same percentage learning duration, as more learning data is available (higher t_l), the performance of all the models improves, except when $t_l = 30$ days. The plots also show that the crossover point between the performance of bagged decision trees and Naïve Bayes is about three weeks, as indicated above.



(a) $t_l/(t_l + t_p) = 0.1$ (b) $t_l/(t_l + t_p) = 0.25$

Fig. 3.7. Effect of learning duration t_l on prediction performance for different values of $t_l/(t_l + t_p)$

Based on the results, we can conclude that an availability prediction system using bagged decision trees can learn from a few days of routing data logs. Our system can be adapted to *real time deployment* by sliding the time window of the learning period to always learn from the most recent data. For example, if we learn from a week of data, we can slide our learning window by a day at a time to always learn from the most recent past week. Predicting the availability for about three times the

learning duration gives accuracy and AUC of around 75% and 0.66 respectively. If these performance measures are acceptable, one can triple the prediction duration at every stage. If we are learning our prediction models from the most recent week of data, we can predict the availability for three weeks into the future maintaining this level of performance. If one desires higher performance, one should reduce the prediction duration for the same learning duration, i.e., increase the percentage learning duration. Our prediction framework allows the system administrator to trade off prediction performance and prediction duration.

3.5.5 Classification Attributes

We now study the importance of attributes in the prediction process, by studying the effect of using different sets of attributes on the output metrics of Naïve Bayes and bagged decision trees. We start with the bagged decision tree results from Table 3.9, and remove certain attributes of the combinations, so that less data is available to the prediction model. The degradation in various performance metrics is studied; as degradation increases, the importance of the removed attribute subset increases. We present typical results of removal of some of the attributes for $t_l = 30$ days, $t_l/(t_l + t_p)$ $= 0.1$ in Table 3.10. We choose these values of t_l and t_p as this represents a long enough learning period and a hard prediction problem: predicting availability for 9 times the learning duration. The results for the other values of t_l and t_p were similar albeit with different values. We choose AUC for comparison because of its strength as a performance metric as described earlier. The first column of the table indicates which attributes of the combinations were used for prediction. Along with using subsets of the four attributes from Section 3.4.2, we also use the the attribute of *past availability* to build prediction models. This attribute is used in the simple model and we seek to study the performance of machine learning-based prediction models which use this attribute to predict availability.

Table 3.10
Percentage change in performance metrics with subsets of attributes for $t_l = 30$ days, $t_l/(t_l + t_p) = 0.1$. All percentage changes are w.r.t. results of the corresponding models from Table 3.7 and Table 3.9

Attributes Used for Prediction	% Change in AUC for Naïve Bayes	% Change in AUC for Bagged Decision Trees
Past Availability	-12.03	-9.8
MTTF	-4.43	-1.97
MTTR	-14.67	-1.93
Prefix Length	-15.22	-24.96
Update Frequency	-18.18	-7.12
Prefix Length and Update Frequency	-9.94	-3.13
MTTF and MTTR	-3.75	-1.65
MTTR, Prefix Length and Update Frequency	-4.00	-1.33
MTTF, Prefix Length and Update Frequency	-0.03	0.25

These results lead us to the following conclusions. Performance significantly degrades (AUC is 10-12% lower) when only past availability is used. Combining this with the results of the simple model, we conclude that past availability is not an adequate metric for prediction of future availability. Prefix length and update frequency are weak attributes, with prefix length being the weakest since using it alone causes the AUC to decline by 7-25%. MTTF is the most important attribute since using it alone causes the least drop in AUC among any single item attribute set. Using either MTTF or MTTR with prefix length or update frequency, or MTTF and MTTR together, causes the AUC and accuracy to be within 4% of their values when no attribute is removed. MTTF combined with the prefix length and update

frequency give very close results to those obtained when MTTR was also added to the set, further confirming that MTTF is the strongest attribute (complemented by the use of prefix length and update frequency). We also experimented with adding *past availability* to these attribute subsets and found that the performance did not change significantly. Thus, there exists no subset or superset of the four attributes used that would cause significantly better results than the four attribute set we have chosen.

It is intuitive that MTTF is the most important prediction attribute and MTTR is the next important, since the time to fail or recovery will characterize the availability of a combination; a *high* availability combination should have a high MTTF and low MTTR.

It is also interesting to note that the Naïve Bayes model is much more sensitive to the removal of attributes than bagged decision trees. This is because the intrinsic assumption used in this model is that attributes are conditionally independent given the class label. This assumption cannot be used when there is only one attribute. Among multiple attributes, the results will depend upon the degree of conditional independence between the attributes. The bagged decision tree model, in contrast, builds decision trees based on various attribute values. While attribute removal does hurt its performance, the trees formed based on other attributes are still reasonably accurate, unless the prediction attribute is weak, e.g., prefix length.

3.5.6 Additional Attributes

We now investigate whether the prediction accuracy can be improved if we add additional attributes that we have not considered in this work so far. In Section 3.4.2, we gave the rationale for the selection of attributes to be the relation between these attributes and the availability of the prefix. There are other attributes of BGP updates [10, 183] such as AS path, community, MED, and aggregation, which we have not considered in this work. This is because we believe that these attributes are not

significantly related to availability as much as MTTF, MTTR, update frequency, and prefix length. For example, repeated announcements with different AS path do not change the Announced or Withdrawn status of prefixes. If the prefix flaps frequently with announcements and withdrawals, affecting availability, this will be captured by our update frequency metric.

The additional attributes that we consider in this section are: (1) Average AS path length percentage change of the changed AS path w.r.t. the old AS path, averaged over all announcements, (2) Fraction of times AS path length changes over all announcements, (3) Fraction of time the aggregator attribute changes over all announcements, (4) Fraction of time the community attribute changes over all announcements, and (5) Fraction of time MED attribute changes over all announcements. As usual, we compute these attributes for each combination and use them for availability prediction. We consider these attributes one at a time, and all these five together for availability prediction using bagged decision trees. We find that the AUC results are 16% poorer on the average across these six prediction cases w.r.t. the results in Table 3.9 . The average AUC of prediction comes out to be only 0.55. Although better than a random classifier, these results are poor compared to the prediction performance achieved previously. This is explained by the fact that these attribute changes are due to AS policies for diverting traffic to the inbound prefixes by modifying existing announcements, and are less correlated with changes in the announced or withdrawn state of prefixes, which affects availability.

3.5.7 Predictability of Prefixes

Thus far, we have used a random set of (peer, prefix) combinations for training the prediction models and for testing the effectiveness of the prediction techniques. We now investigate whether certain combinations are more predictable than others. The intuition behind this is that the availability of a combination is more predictable from the attributes chosen in our work for certain kinds of prefixes than for others. There

can be several causes of BGP routing dynamics [10], and some causes are likely to be more correlated with availability, making a particular prefix group more predictable. For example, a prefix can be withdrawn and announced with a specific pattern (e.g., dependent on time of day) for traffic engineering purposes, and all prefixes which are announced according to similar policies will exhibit more predictable availability. The authors of [10] discovered both daily and weekly patterns in prefix announcements, attributed to several known and unknown causes. Y. Zhang et. al. [123] predicted data plane failures using control plane updates and also observed that certain prefixes are more predictable than others. While we leave detailed investigation of exact predictability classes of prefixes to future work, we investigate whether there are more predictable combinations in our dataset.

Our methodology is motivated by [123]. Out of all the prediction models considered in this work, only Naïve Bayes (Section 3.5.2) gives a probability of prediction of prefix availability as *high* or *low* based on its attributes. We use an option in Weka [174] to output the class prediction probabilities for each of the instances along with the true availability class. For each of the 20 sets of results from Table 3.7, we investigate the instances which were classified incorrectly. We note the probability of incorrect classification P_{inc} as the P(predicted class label - *low* or *high*) for the incorrectly classified instances output from the Naïve Bayes model. P_{inc} can never be less than 0.5 since a label is only predicted if its probability is greater than the other class label. The CDF of P_{inc} is shown in Figure 3.8. The plot shows that about 91% of the incorrectly classified instances have a class prediction probability above 0.93 when they are incorrectly classified. This implies that when a prediction error is made, the case is *not* borderline – the model almost surely predicts the incorrect class label. This gives credence to the fact that some prefixes in combinations are very poor in predictability compared to others.

We now seek to isolate the combinations which have poor prediction performance. We look at the instances incorrectly classified by Naïve Bayes for all the 20 cases of Table 3.7 and isolate the combinations which have a probability of prediction of

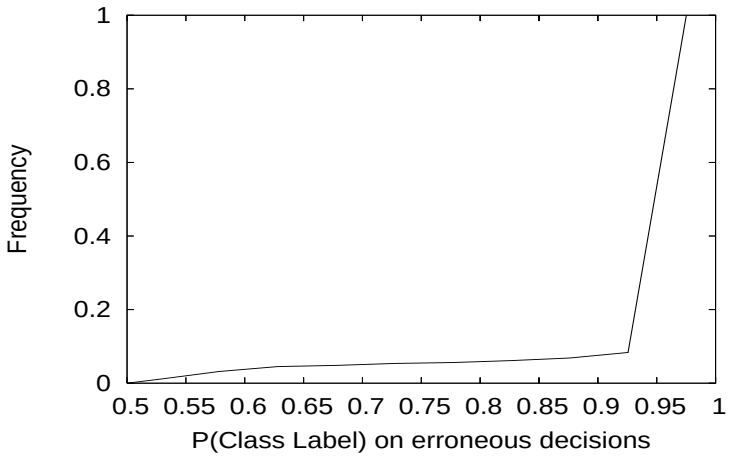


Fig. 3.8. CDF of class label prediction probability for incorrectly classified instances using Naïve Bayes

the (incorrect) class label exceeding 0.75. We chose this threshold of 0.75 since it is midway between 0.5 and 1 and we want to ignore combinations for which a slight prediction error is made. Across all the 20 cases, this gives 15,722 "poorly predicted" combinations, which is about 39.33% of the total number of unique combinations for the 20 cases.

To evaluate the prediction performance of the poorly predictable combinations versus the predictable ones, we run the bagged decision tree model from Section 3.5.3 on both sets of combinations. We show the performance as indicated by AUC for both the predictable and poorly predictable combinations in Table 3.11. The results indicate a large difference in predictability between the two types of combinations. On the average, the predictable combinations have 40.95% higher prediction performance (measured in terms of AUC) than the poorly predictable combinations.

These results indicate a close to bimodal distribution of predictability of combinations. There are some combinations which are highly predictable (having an average AUC of 0.864) and some which are poorly predictable (average AUC of around 0.5), and, on the average, a 40.95% difference in AUC exists between the two prefix sets. We conjecture that this due to the two types of reasons behind BGP dynamics:

Table 3.11
Results for predictable and poorly predictable combinations obtained from bagged decision tree model

t_l	$t_l/(t_l + t_p)$	AUC for predictable combinations	AUC for poorly predictable combinations	% difference in AUC of Col. 4 from Col. 3
1 day	0.1	1	0.4581	54.19
	0.25	1	0.5144	48.56
	0.5	1	0.5128	48.72
	0.75	1	0.5226	47.74
	0.9	1	0.4893	51.08
7 days	0.1	0.6624	0.4058	38.73
	0.25	0.7046	0.5311	24.62
	0.5	0.7807	0.5321	31.84
	0.75	0.8565	0.4783	44.16
	0.9	0.9188	0.3766	59.01
19 days	0.1	0.8384	0.3236	61.40
	0.25	0.8469	0.5589	34.01
	0.5	0.8608	0.6074	29.44
	0.75	0.8626	0.593	31.25
	0.9	0.8837	0.4033	54.36
30 days	0.1	0.6971	0.4848	30.45
	0.25	0.8122	0.5501	32.27
	0.5	0.8269	0.5777	30.14
	0.75	0.8558	0.6113	28.57
	0.9	0.8751	0.5394	38.36

planned prefix traffic engineering leading to specific update patterns, and the non-stationary nature of link failures [123]. Understanding the reasons behind varying

prefix predictability has been shown to be a difficult problem [123] because of lack of information about AS policies and limited visibility to BGP updates from vantage points. This is similar to root cause identification for BGP updates, which is a hard problem as well [58, 63, 64, 184]. We leave detailed investigation of the causes behind prefix predictability to future work.

3.5.8 Larger Test Datasets

So far in this chapter, we have used training and test sets which are constructed out of a sample of 10,000 combinations using 10-fold cross-validation. We now investigate the scalability of our models, where we apply the learned models to a large number of combinations. This may be required of a typical prediction application, if one is interested in predicting the availability of a set of prefixes from a large number of vantage points in the Internet.

Table 3.12
Percentage change in performance metrics of a large prediction dataset from Table 3.7 and Table 3.9 for $t_l = 30$ days, $t_l/(t_l + t_p) = 0.1$.

Performance Metric	% Change for Naïve Bayes	% Change for Bagged Decision Trees
AUC	1.45	-1.52
Accuracy	2.01	-0.12

To evaluate scalability, we learn Naïve Bayes and bagged decision trees from 10,000 combinations, but predict the availability of all the remaining combinations in each month (about 11.5 million). The prediction takes only about 2 minutes to complete for each of the models on a 3.6 GHz single-core machine. The prediction results for a typical case ($t_l = 30$ days, $t_l/(t_l + t_p) = 0.1$) show about a 1-2% difference from the results in Table 3.7 and Table 3.9, as illustrated in Table 3.12. We therefore conclude that our models are scalable for availability prediction of a large number

of combinations, without significant degradation in prediction quality. These results also show that the 10-fold cross-validation methodology does not suffer because of using a relatively low number (1000) of combinations in the test set.

3.6 Chapter Summary

In this chapter, we have developed a long-term availability prediction framework that uses mean time to recovery, mean time to failure, prefix length, and update frequency as attributes. These attributes are easily computable from public RouteViews data observed for a short period of time. Our framework learns a prediction model from a set of Internet prefixes, and uses that model to predict availability of other prefixes. To the best of our knowledge, this is the first work that uses the similarity of prefix behavior in the Internet to predict properties such as availability.

Our simple prediction model is a good baseline with high true positive rate and accuracy. The model, however, has a high false positive rate and low AUC. Naïve Bayes and bagged decision trees improve on these metrics, and the latter performs best especially when the learning period t_l is shorter than about 3 weeks. For longer learning periods, Naïve Bayes performs best. The Naïve Bayes model, however, is highly susceptible to a change in attributes. We recommend the use of bagged decision trees, learned from a moderate learning period of a week or two, to predict availability for longer future durations. The learning period can be a sliding window which slides with a granularity of a few days so as to feed the model with the most recent data for learning.

We also find that mean time to failure is the most important attribute for prediction followed by mean time to recovery. Past availability is inadequate to predict future availability, which is also a reason for the worse performance of the simple model. We quantify how prefix availability is related to prefix length and update frequency. Our results show that future availability is indeed predictable. The results are promising given that we are using only public information about prefixes and that

we are building our model using a random set of prefixes. Our prediction models are scalable and can be applied to large number of combinations in the Internet with little performance overhead.

The next chapter describes our work on discovering prefixes in the Internet which have similar propensity to fail. We discuss how to discover these prefixes to form BGP molecules and present its applications.

4. BGP MOLECULES: UNDERSTANDING PREFIX FAILURES

The previous chapter presented our approach on predicting long-term prefix availability based on the premise that prefixes in the Internet are similar. Hence, prediction models learnt from some prefixes can convey valuable information about predicting properties of other prefixes. In this chapter, we investigate this similarity and develop a new prefix grouping called BGP molecules.

4.1 Motivation

BGP disseminates control-plane reachability information in the Internet, announcing and withdrawing paths to prefixes. The paths can be withdrawn due to several reasons like link failures or AS policy changes, causing the prefixes to become unreachable from various portions of the Internet (Section 2.2). We refer to this as a routing failure, or just failure for short. The goal of this chapter is to seek insight into these failures. For each given prefix of interest, we determine a group of prefixes in the Internet with similar failure characteristics. We refer to this prefix group as the *BGP molecule* of the prefix of interest, since it generalizes the concept of *BGP atoms* [185]. BGP atoms are clusters of prefixes such that all BGP routers (peers) which can reach prefixes in the same atom do so using the same AS paths. We consider similarity in AS paths to prefixes as just one of the possible metrics in constructing molecules. We develop correlation coefficient metrics for comparing two prefixes in terms of their failure tendency (Section 4.3). We consider a number of prefix characteristics to determine their relationship with the correlation coefficients (Section 4.4), and show that origin AS similarity is not a sufficient indicator of similarity in prefix failure tendency, whereas AS paths to prefix from different Internet vantage points is more

relevant. BGP atoms are necessarily formed by prefixes belonging to the same AS, whereas we consider the entire set of visible Internet prefixes to find prefixes similar to the prefix of interest. Our prefix clusters, the BGP molecules, can consist of prefixes belonging to different ASes, just like a molecule can consist of atoms belonging to different chemical elements. Since the BGP molecules are prefix-specific, BGP molecules of two different prefixes of interest may have prefixes in common.

While BGP atoms were introduced to potentially aggregate BGP prefixes that are subject to the same policy [185], our goal in forming BGP molecules is formulating a fundamental unit which can be used in effective diagnosis of routing problems, ultimately improving the security and reliability of the Internet control plane. We study the potential of BGP molecules in predicting failure of the prefix of interest by considering four failure prediction algorithms, with and without the use of BGP molecules, in Section 4.5. We find that the prediction scheme without using BGP molecules, namely using prefixes that have failed with the prefix of interest in the past, is computationally intensive with medium prediction accuracy. BGP molecules are easier to compute since they are constructed using one or more routing tables or information about the geographical location, and achieve higher failure prediction accuracy. A key difference between our work and BGP atoms is that we view the formation of BGP molecules as dynamic, with molecules being formed using different prefixes over time. This can occur as prefixes are advertised with different attributes.

In Section 4.6, we consider the application of BGP molecules in improving availability of content servers of Akamai, which is a dominant Content Distribution Network (CDN) in the Internet. We consider the problem of ensuring high control-plane availability of Akamai's servers, which is defined as the time when the prefixes, to which the servers belong, are in Announced state divided by the time period of interest. We perform large-scale experiments from around 350 PlanetLab [186] nodes, which query Akamai's CDN for different pieces of content periodically for six weeks. Using the servers returned and control plane announcements of prefixes to which the servers belong, we find that the mean availability of Akamai's servers is very high

(99.98%), but still shy of the holy-grail of five-nines availability [175]. In fact, around 10% of the queries for a single piece of content lasting six weeks experienced less than five-nines availability. We use our knowledge of BGP molecules to propose a new scheme in which Akamai returns primary and backup servers from prefixes which are not in the same BGP molecule, and hence are unlikely to fail together. Our evaluation of this scheme shows that nearly all the content queries over a six-week period would have achieved greater than five-nines availability, and a very small percentage (<1%) would have availability less than 100%. We also show that given estimates of latency of servers from clients (which Akamai collects via active measurements [140]), our scheme can *simultaneously* increase the availability of Akamai's servers as observed by the client, while reducing their latency.

When we cluster prefixes into molecules, we gain information about which prefixes are likely to be affected by a single event. One can then develop a reactive routing mechanism to route around failures [122]. For instance, iPlane Nano [121] showed that intelligently selecting detours can improve the performance of routing. BGP molecules also reveal similarity in failure tendency and can be used to improve the reliability of web-based applications and cloud computing, similar to Section 4.6. Diagnosis is not the only goal of clustering prefixes; gaining a better understanding of the similarity of the prefix address space is also another goal of this work. This can lead to a better selection of prefix candidates for further inspection by data plane monitoring systems like Hubble [68], and deeper insight into the behavior of subset and superset prefixes [173] in failure scenarios.

4.2 Datasets

We obtain BGP routing tables and updates from RouteViews [9] for the month of March 2009 and process them as described in Section 3.3, removing spurious updates caused by routing table transfers. We selected this month since no known major routing event (such as an undersea cable cut) occurred, in order to produce unbiased

results. This is important because the BGP molecules will typically be used in normal operation scenarios, as significant routing events are rare.

4.2.1 Extracting AS-Specific Information

Each prefix announcement and each routing table entry is associated with an AS path and a peer, which is the vantage point that can reach the prefix through the AS path. The origin AS of the prefix is the last AS on the AS path. Thus, if peer Q uses the following AS path to prefix P: $(a_1, a_2, ..., a_{n-1}, a_n)$, where each $a_i, i = 1...n$, is an AS, then a_n is the originating AS of the prefix. We extracted 31,576 unique origin ASes out of the data visible for the month, and stored the prefixes that they originate along with an array of the times of prefix state changes. The state of a prefix can be Up (U) when the prefix is in an announced state or Down (D) when the prefix is in a withdrawn state. Each prefix has a state change array for each of the peers that can reach this prefix.

While extracting AS-specific information, we noticed two interesting cases. First, the same prefix can be associated with multiple ASes. This is the commonly known Multiple Origin AS (MOAS) problem [187] and this makes attributing the prefixes to an individual AS difficult. We found that about 1.6% of the prefixes exhibited MOAS conflicts, i.e., their origin could be attributed to two or more ASes. Since our focus in this work is not on resolving MOAS conflicts, we attribute the prefix to all of the ASes that appear to originate it. This design choice has the following implication. Since the withdrawals of a prefix by a peer do *not* carry an AS path, it is impossible to deduce which of the origin ASes for a MOAS prefix is now unreachable. Hence, we change the states of the prefix to "Down" for *all* the ASes that originate this prefix. Second, the origin AS may not be a unique AS but can be an AS_SET [47], which is an aggregate of ASes represented within curly braces $\{a_1, a_2, ..., a_{n-1}, a_n\}$ and is produced by route aggregation. If the origin AS of a prefix is an AS_SET, we keep it as a separate entity along with the prefixes that it originates. We found about

0.013% of the prefixes to be originated by AS_SETs, so the effect is less significant than the MOAS issue. Only 36 out of the 31,576 ASes (i.e. 0.11%) are AS_SETs.

The number of prefixes originated by an AS ranges from 1 to 4402 in our data. We had 329,658 prefixes in our dataset with an average of 10.44 prefixes originated by an AS. However, the distribution is highly skewed with about 42% of the ASes originating only 1 prefix and about 86.2% of the ASes originating less than or equal to 10 prefixes. A partial histogram of the prefixes originated by an AS is shown in Figure 4.1. The frequency of the ASes with number of prefixes greater than 20 (and ending at 4402) is not shown in the graph and totals 7.36%.

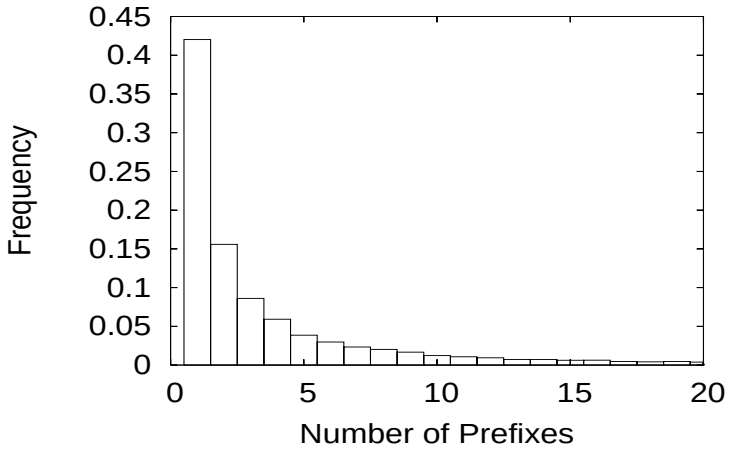


Fig. 4.1. Partial histogram of the number of prefixes originated by an AS

4.3 Metrics

We now define the metrics that we use to evaluate the failure tendency of prefixes. Since we are primarily interested in comparing prefixes to each other, we define correlation coefficient based metrics. A high value of the correlation coefficient metric indicates that the failure tendencies are close to each other. As described in Section 4.2.1, for each prefix and each peer that can reach it, we have state change sequences of the prefix recording the time when the state of the prefix goes from U

to D and vice versa. We compare the state change arrays of two prefixes when they are viewed by the same peer.

Our initial attempt at measuring similarity of failure tendency was defining a "state correlation coefficient" between prefixes. A pair of prefixes may have their first recorded state at different times (determined by inspecting their state change arrays as viewed by the same peer). Hence, we compute the "start time" of the pair by choosing the higher of the two start times of the prefixes. This is the earliest time when we have information available about both prefixes. The "end time", which is the last time we have information about both prefixes, is taken to be the timestamp of the last update in our dataset. The assumption here is that all updates will be recorded in RouteViews, and hence the state of the prefix stays the same from the time of the last state change to the timestamp of the last update of the month. Given the start time and end time of the prefixes, we subtract the total time when the state of the prefixes disagree from the total time that they agree and divide that by the time for which we have information about the two. Formally, the state correlation coefficient of the prefixes is defined as: $\frac{DD+UU-DU-UD}{DD+UU+DU+UD}$, where {xy} with x, y= U or D denotes the total time in the month long dataset when the first prefix state is x and the second prefix state is y. Clearly, the value of this coefficient can range from -1 to 1.

This definition of state correlation coefficient is sensitive to synchronization mismatches, not only in the timestamps of the recorded updates (which we assume to be correct), but also because the same routing change event affecting two prefixes can be observed at the same peer at slightly different times due to delayed convergence (which can be of the order of few minutes). Without a priori knowledge of which routing events occur (which are difficult to reliably determine, even with root cause analysis algorithms, as discussed in Section 2.2.1), it is impossible to correct synchronization delays. Our correlation coefficient computations aims to study prefix state changes *as observed* by vantage points in the Internet, hence any asynchronicity caused by Internet convergence or BGP policies should *not* be corrected for. This

asychnronicity is of the order of a few minutes, which is not too significant given that we are computing the coefficient over a month long period.

We compute the amount of time a prefix spends in the "Up" state in our entire dataset of 329,659 prefixes and find the average uptime to total time ratio to be 92.37% with a median value of 99.9955%. Since prefixes are mostly up, the state correlation coefficient is not an accurate measure of the tendency of prefixes to fail together. Therefore, we define "failure correlation coefficient" as: $\frac{DD-DU-UD}{DD+DU+UD}$, where {xy} with x, y = U or D denotes the total time in the month long dataset when the first prefix state is x and the second prefix state is y. We ignore the time when both prefixes are in the "Up" state, and only consider the time when at least one of the prefixes has failed. Like the state correlation coefficient, this can also range from -1 to 1. The failure correlation coefficient captures the correlation between the failure tendencies of two prefixes more accurately, since it evaluates whether one prefix has failed, given that the other prefix has failed. Two prefixes which are always up will have a state correlation coefficient of 1, whereas their failure correlation coefficient is undefined (which is intuitive, since no failure has been observed for either of them).

4.3.1 Baseline State Correlation Coefficient

The baseline state correlation coefficient is that of any two prefixes in the Internet, chosen at random, averaged over all prefix pairs. This computation is important since a good prefix clustering is expected to include prefixes with significantly higher failure tendency correlation than the baseline case.

Unfortunately, computing the state correlation coefficient of all prefix pairs in our dataset is a computationally infeasible task since we have 329,658 prefixes and hence about 54.3 billion prefix pairs. Hence, we resort to random sampling: we choose 1% of the ASes (or 316 ASes) randomly and only consider the 2353 prefixes originated by those ASes. As Figure 4.2 shows, the frequency of prefixes originated by an AS in the 316 AS random sample is about the same as that of the entire set of 31,576

ASes, indicating that the prefix sample is a representative one for the prefixes of the entire Internet. Although we do work on this reduced dataset for the remainder of the chapter, we claim that due to the above mentioned reason, and the randomness of our selection process, our results for correlation coefficients are unbiased. Besides, the construction of BGP molecules relies on finding *some* prefixes in the Internet which are similar in failure tendency to the prefix of interest. Limiting the sample of prefixes only makes our BGP molecule construction and the subsequent prediction mechanism look worse than it could have been had more powerful computation mechanisms been available for our study. Another reason for this reduced sample of prefixes is that we aim to make the BGP molecule construction and failure prediction an online mechanism.

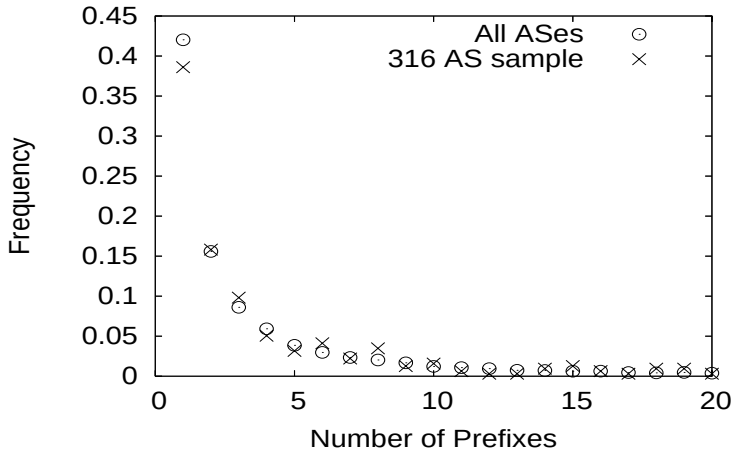


Fig. 4.2. Comparison of partial histograms of the number of prefixes in an AS in the random sample vs. for all ASes

We now treat the 2353 prefixes in these 316 ASes as one "chunk" of prefixes, and compute state correlation coefficients for the month's data among each of the approximately 2.76 million prefix pairs. The multiple values of the coefficient (obtained for each prefix pair by considering various peers that see the pair) are averaged to compute a state correlation coefficient for the prefix pair. We tackle the MOAS problem by maintaining a correlation coefficient for each ($Prefix_1$, AS_1, $Prefix_2$,

AS_2) quartet where AS_i originates $Prefix_i$ for i=1,2. We refer to this correlation coefficient as belonging to the prefix pair ($Prefix_1$, $Prefix_2$) with the understanding that the quartet implementation takes care of the originating ASes. The average state correlation coefficient of the 2.76 million prefix pairs in our dataset is 0.777 and the median is 0.99902. Its histogram is shown in Figure 4.5. The extreme values have the highest frequency with about 7.5% from -1 to -0.9 and about 84.7% from 0.9 to 1.

4.3.2 Failure Correlation Coefficient

In our 316 AS sample with 2353 prefixes, the average and the median values of uptime ratio were 94.02% and 99.998% respectively, showing that the prefixes are in the "up" or announced state most of the time. Hence, we compute the failure correlation coefficient for all the prefix pairs, whose histogram is shown in Figure 4.3. The total number of prefix pairs in the plot is 2.724 million vs. the 2.76 million pairs earlier, as the pairs where both the prefixes were up for the entire month w.r.t. every peer are omitted. The failure correlation coefficient's median is -0.999999 with mean -0.927647. This is because two arbitrary prefixes in the Internet are unlikely to have high tendency to fail together unless they share common characteristics. It is our goal to find these characteristics. The results can be likened to the fact that two arbitrary chemical atoms in a large enough sample of atoms are unlikely to be the same and hence an average "similarity coefficient" would be close to -1.

4.4 Constructing BGP Molecules

We now describe techniques to construct a BGP molecule of a prefix of interest, i.e., the set of prefixes in the Internet with similar failure tendency as the prefix. We choose the prefix of interest, one each from the 316 AS sample so that we do not bias our results towards a specific AS/prefix group. Clustering techniques from data mining [188] require that the distances amongst all pairs of points be known for

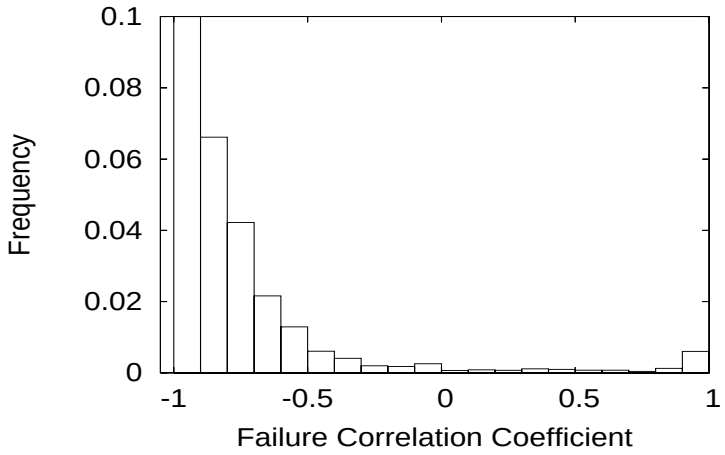


Fig. 4.3. Expanded histogram of the baseline failure correlation coefficient. First bin frequency is 0.827.

inter-cluster and intra-cluster similarity to be measured, so that points with higher similarity belong to a cluster. We have 329,658 total prefixes in our dataset, and at least one similarity measure needs to be computed for each prefix with the prefix of interest, averaged over 40 or so peers that see the prefix pair. Even if we optimistically assume that each similarity measure computation averaged over 40 peers takes 1 second to compute, it would take about four days to compute the similarity measures themselves (not counting the time required for clustering) on a single processor machine. Of course, one can improve the computing infrastructure, but that imposes restrictions on the online computation of BGP molecules. The scale of the Internet works against us in the computation of the BGP molecules, especially in a dynamic setting, where molecule membership may have to be recomputed periodically. Hence, we resort to heuristics for computation of the BGP molecules and present multiple methods in this section to construct BGP molecules. Our evaluation of the quality of the molecules is based on the state and failure correlation coefficients described in Section 4.3.

4.4.1 On a Per AS Basis

A natural way of constructing BGP molecules is to cluster prefixes based on the AS that originates them. Thus, two prefixes are in the same cluster if they are originated by the same AS. To evaluate this clustering technique, we created 31,576 clusters, one for each AS and computed the state correlation coefficient for prefixes in the same cluster, except for the ASes which originate only one prefix. Only about 58% or 18304 of the 31576 ASes contribute a value. An AS having more than one prefix will have several values of correlation coefficient for each of the prefix pairs, one for each peer (vantage point) that sees the pair. The values of the state correlation coefficient for the prefix pairs in an AS are averaged, yielding an "AS state correlation coefficient."

The AS state correlation coefficient histogram, with bin size 0.1, is given in Figure 4.4. The plot shows that about 87.6% of ASes have values between 0.9 and 1. A partial histogram which shows the lower frequencies along with their comparison to the baseline state correlation coefficient is depicted in Figure 4.5. The frequencies of bins except the last bin are much smaller, never exceeding 2.6%. The average value of the AS state correlation coefficient, averaged across 18304 ASes, is 0.927, whereas the median is 0.999887.

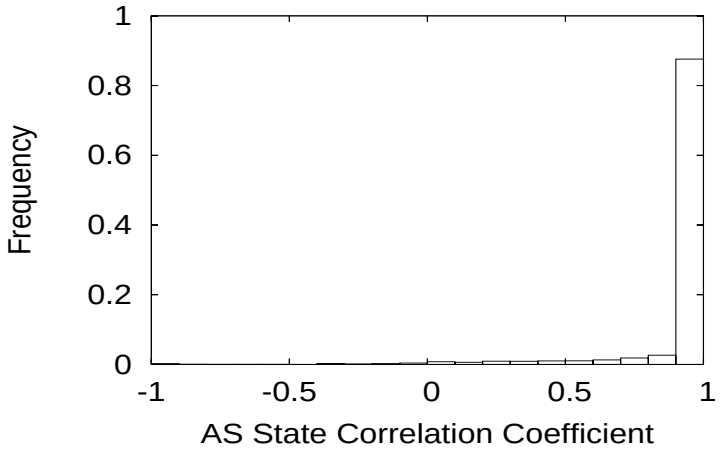


Fig. 4.4. AS state correlation coefficient histogram.

Figure 4.5 also shows the baseline plot, which is different since it has negligible frequency of correlation coefficients above 0. We can clearly see that lower values of the state correlation coefficient have a higher frequency in the baseline case than in the AS case. The average state correlation coefficient for the baseline case is 0.777 vs. 0.927 for the AS case, and the median is 0.99902 vs. 0.999887 for the AS case. These results suggest that the originating AS groups prefixes with higher similarity together.

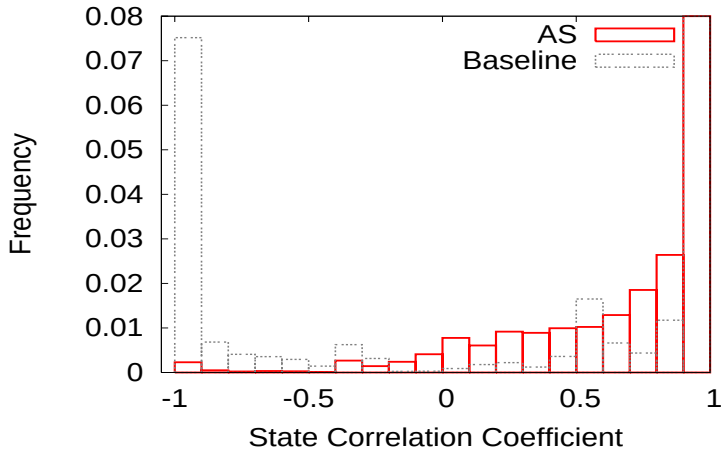


Fig. 4.5. Comparison of partial histograms of the baseline state correlation coefficient with the AS one. Last bin, between 0.9 and 1, is 0.876 (AS), 0.847 (Baseline).

We then compute the AS failure correlation coefficient, by averaging all its computed values for prefix pairs with the prefixes belonging to the same AS. The histogram of the 18104 ASes which have an AS failure coefficient is shown in Figure 4.6. The plot shows that about 19% of the ASes have an AS failure coefficient between 0.9 and 1, and the remaining frequencies are about evenly split with 5-10% frequency in each bin. Comparing Figure 4.3 to Figure 4.6, we find that the originating AS of a prefix helps determine its failure tendency. However, the nearly equal frequencies of most of the bins of Figure 4.6 and the fact that about 49.5% of the ASes have a negative failure coefficient is a driver for finding additional prefix characteristics

that influence its failure tendency. Further, when an AS has a single prefix, BGP molecules formed on an originating AS basis cannot be used to find similar prefixes. We therefore explore additional prefix characteristics in Sections 4.4.2 and 4.4.3.

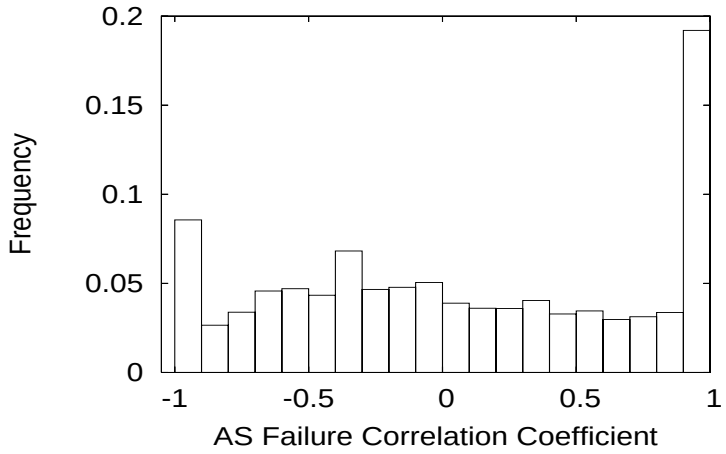


Fig. 4.6. Histogram of the AS failure correlation coefficient.

We conclude this section by studying the relationship between the AS failure coefficient and AS characteristics. We find no clear relationship between the number of prefixes originated by an AS and the AS failure coefficient. However, the geographical spread of an AS does seem to have a correlation with the AS failure coefficient. We define the AS geographical spread as the maximum geographical distance among all pairs of prefixes originated by an AS, given that an AS originates at least two prefixes. We use MaxMind's GeoLiteCity application [189] to find the latitude and longitude of the location of the dotted decimal portion of the prefix, and then compute the distance between two prefixes using the Haversine Formula [190] for computing the great-circle distance.

We compute this AS geographical spread for the 18104 ASes that have an AS failure coefficient. The minimum and maximum values of the spread are 0 and about 12200 miles respectively. We divide this 12200 miles into 20 bins of 610 miles each, and compute the average AS failure coefficient of the ASes which fall in each bin based on their spread. The results are depicted in Figure 4.7. As geographical spread of an

AS increases, on average, its failure coefficient declines. A positive failure coefficient is only obtained in the first bin of the plot. Further investigation reveals that ASes with spreads equal to zero miles have an average failure coefficient of about 0.15. ASes with spreads less than approximately 150 miles or so have a coefficient greater than 0.1, whereas for larger spreads the coefficient slowly starts drifting into negative territory. This motivates studying geographical location as a factor that affects failure tendency of a prefix, and we study this in Section 4.4.3.

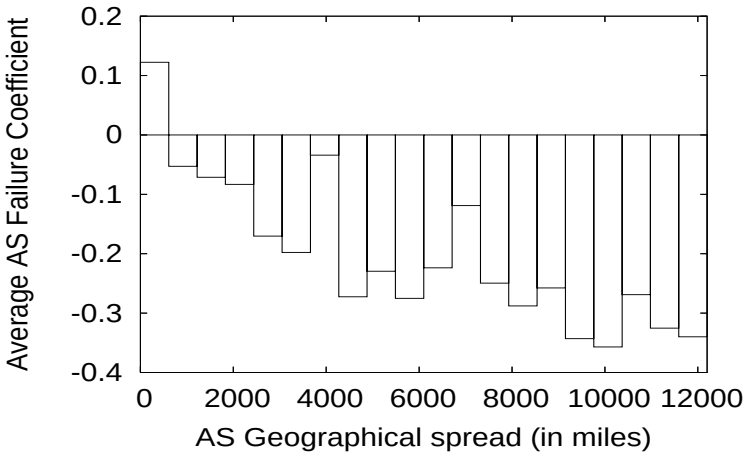


Fig. 4.7. Average AS failure coefficient w.r.t. AS geographical spread

4.4.2 Using AS Paths

In this section, we study the role of AS paths, which is the sequence of ASes advertised by BGP for routing from the vantage point to the prefix. Prefixes which have similar AS paths from one or more vantage points are expected to have similar failure tendencies. This idea of using AS paths to group prefixes was initially proposed in [185] where prefixes are grouped into BGP atoms if they have the same AS paths from every visible default router. This definition necessitates that the prefixes belonging to the same atom belong to the same AS, since the last AS on the AS path is the one which originates the prefix. We apply a broader use of AS paths since we

are interested in finding prefixes in the entire Internet which are similar to the prefixes of interest. We use the AS paths occurring in the routing tables and *remove the first and last AS* in the path. The first AS is the one belonging to the vantage point which sees the prefix and is uninteresting when we aggregate data across vantage points, whereas the last AS is the originating AS of the prefix. We also remove AS path prepending [47] since that has no implication on the sequence of ASes traversed between the vantage point and the prefix.

To investigate whether AS path similarity corresponds to similarity in prefix failure tendency, we conduct the following experiment. We form a "routing table set" containing at most one routing table for each day in the dataset (we select the table with the largest number of entries), for days when the number of entries is greater than the average number of entries in a routing table in our dataset. This eliminates short and possibly corrupted routing tables and improves computational efficiency. The reduction is meaningful because routing tables closely spaced in time are expected to have significant overlap in their entries. We obtained 30 routing tables for March 2009, yielding a "combined routing table" for the month with 14.8 million entries.

Because the combined routing table contains entries at different times, it is not unusual to have multiple entries for a prefix from the same peer's point of view. We start with the 2.76 million prefix pairs for which we have computed the failure correlation coefficient in Section 4.3.2. We narrow down this group of coefficients for computational reasons by choosing sets of increasing coefficient values from 0 to 1 differing by at least 0.02 from the previous set and choosing no more than 1000 values for each set. This reduces our group to about 60,500 prefix pairs and for each of those, we see if we have AS paths for both of the prefixes in the pair from at least one peer in our combined routing table. We then compare the AS paths from the peers, one at a time, to compute "AS path correlation coefficient" (defined in the next paragraph). These coefficients are then averaged across peers to determine an AS path correlation coefficient for the prefix pair.

Given two prefixes with AS paths from the point of view of a single peer, we remove AS path prepending and the first and last AS of the AS path. If the length of the AS path for Prefix 1 is l_1 and that of the Prefix 2 is l_2, we compute the length of the Longest Common Subsequence (LCS) using the dynamic programming algorithm of [191], and define the AS path correlation coefficient as = LCS/$min(l_1, l_2)$. Thus, the coefficient can range from 0 to 1.

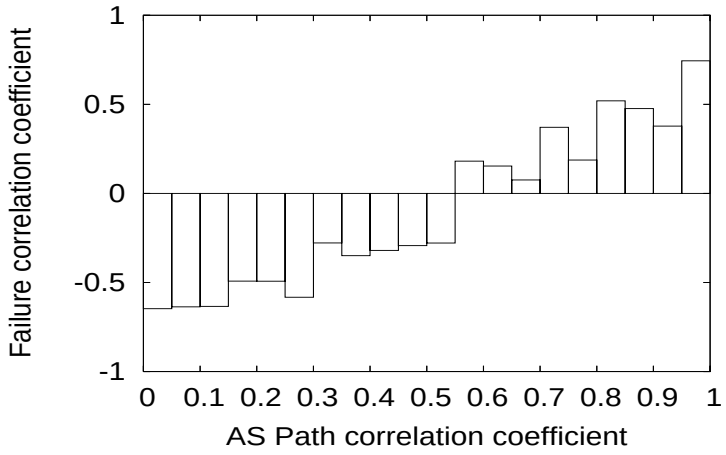


Fig. 4.8. Variation of average failure correlation coefficient of each bin with AS path correlation coefficient

We divide the AS path coefficients for the prefix pairs obtained into 20 bins and compute the average failure correlation coefficient of each bin. The variation of the failure correlation coefficient with the AS path correlation coefficient is plotted in Figure 4.8. The plot shows a clear positive correlation between the two coefficients: as the AS path correlation coefficient between two prefixes increases, their average failure correlation coefficient changes from negative to positive, changing signs at AS path coefficient = 0.55. This validates our hypothesis that AS path similarity is indeed a measure of failure tendency of prefixes.

To construct BGP molecules using AS path alone, we use only the first routing table and no future routing tables since the goal is to use the molecules constructed for failure prediction. For each of the prefixes of interest, we find its AS path sequences

w.r.t. each peer in the routing table, which are a set of AS strings, obtained from the AS paths after removing AS path prepending and the first and last AS in the path. We then search for other prefixes in the routing table which have the same AS path sequence w.r.t. the same peer as the prefix of interest, and place them in its BGP molecule. Note that this is performed w.r.t. each peer to make a fair comparison of the AS path sequence between the prefix of interest and the prefixes in the molecule. We ignore AS path sequences which are just one AS long as that is too general a comparison.

4.4.3 On a Geographical Basis

We now evaluate how geographical distance between prefixes is related to the similarity of their failure tendency. We use the reduced set of about 60,500 prefix pairs as in Section 4.4.2 and compute the geographical distance between the prefixes as in Section 4.4.1, using GeoLiteCity and Haversine's formula. We investigate whether the geographical distance between prefixes is a different dimension from their originating ASes and find that out of prefixes at the same location, about 92.5% belong to the same AS. Zero distance between prefixes does *not* imply that the prefixes belong to the same AS, which suggests that geographical distance is an independent dimension to consider. The percentage of prefixes belonging to the same AS reduces to 90% for prefixes with distance less than 150 miles and to 70% for distance less than 600 miles.

We now evaluate whether geographical distance correlates with the failure correlation coefficient of prefixes. As in Section 4.3.2, we have 20 bins of 600 miles each, and we place each of the 60,500 prefix pairs into one of the bins depending on their distance. We then compute the average failure correlation coefficient of each bin. The results indicate that increasing distance corresponds to a lower similarity in failure tendency, but only the first bin has a positive failure coefficient. We therefore investigate the maximum distance between prefixes below which they will have a high similarity in failure tendency by dividing prefixes from the first 600 miles into bins

of 50 miles each. The results, shown in Figure 4.9, suggest that prefixes with distances 150 miles or less have a fairly high failure correlation coefficient on the average, whereas those with greater distances have a negative coefficient.

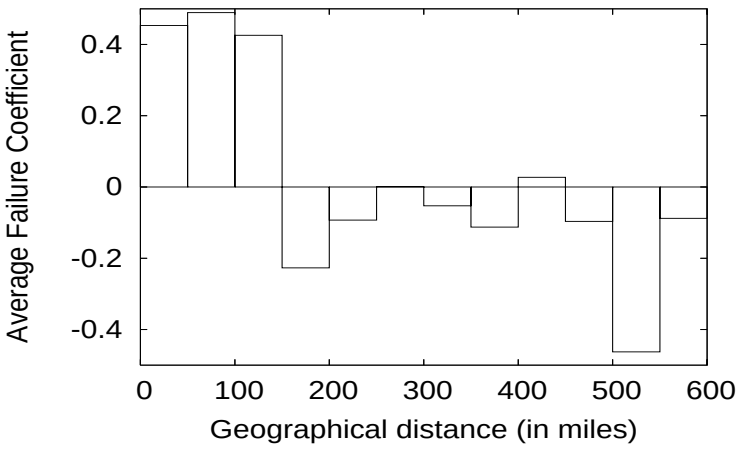


Fig. 4.9. Variation of average failure correlation coefficient with geographical distance between prefixes

4.4.4 Hybrid Scheme

From the above discussion, AS paths to a prefix are a stronger dimension than its geographical location in correlating with its failure tendency. However, frequently there are cases when AS paths alone do not yield any prefixes within a molecule of a prefix of interest. This may be due to (i) the prefix of interest is not found in the routing table used, or (ii) the AS path sequences for finding similar prefixes are only one AS long, or (iii) there are no prefixes in the routing table with the same AS path sequences. Additionally, the number of prefixes in the BGP molecule of a prefix of interest may be insufficient for prediction purposes (Section 4.5). We therefore devise a hybrid scheme for constructing BGP molecules. For the cases where BGP molecules created using AS paths alone do not exist or have an insufficient number of prefixes, we find the prefixes in the rest of the Internet which are within a threshold

distance (150 miles), and place them in the BGP molecule constructed using AS path (if non-empty). Since searching the entire set of Internet prefixes is a computationally infeasible task, we focus on our sample of 316 ASes, containing 2353 prefixes, and construct BGP molecules.

4.5 Predicting Failures using BGP Molecules

In this section, we study the most natural application of BGP molecules: prediction of future failures of a prefix using the failures of prefixes in its molecule. We first evaluate a prediction application which does not use BGP molecules, but instead uses prefixes with known failures close by in time to the failures of the prefix of interest. We then study prediction using BGP molecules, with three different ways of constructing them.

4.5.1 Failure Prediction Methodology

Our prediction methodology involves failure prediction of prefixes of interest given prefixes "similar" to it in some regard. They could be prefixes in the BGP molecule of the prefix of interest or are found similar in some other way. We select a set of 25 random "similar" prefixes for prediction purposes. The number 25 was selected to be large enough to give a meaningful sample, but small enough for low computational overhead in an online prediction application. If the number of "similar" prefixes is less than 25, we typically do not use these prefixes for prediction purposes. There are some exceptions to this that we will point out in subsequent sections when we perform prediction with insufficient prefixes for evaluation reasons. Generally, a failure of the prefix of interest is predicted if a majority of the 25 prefixes fail during a time window, which is kept as a parameter. This prediction application can be easily deployed in the real world if failures of prefixes can be observed e.g. through a live update feed. The use case of operators in ISPs typically have such a feed through peering, else

can obtain it from a public source like RouteViews [9]. We execute our prediction experiments for all 2353 prefixes of the 316 AS random sample for evaluation purposes.

We now present an example. Consider prefix 210.143.240.0/20 belonging to AS 23777 to be our prefix of interest. Its BGP molecule constructed using AS paths has 231 different prefixes. Figure 4.10 shows the indices of the 25 prefixes (numbered 0 to 24) and the time at which they fail. Since 13 prefixes fail within 1 second of each other (< time window t=300 seconds), we predict that the prefix of interest will fail in a time window of t seconds beginning at t_0. The prefix of interest failed at $t_0 + 1$. Since it failed within 300 seconds of the failure of the first prefix in the 13 prefix set, we consider this a predictable failure.

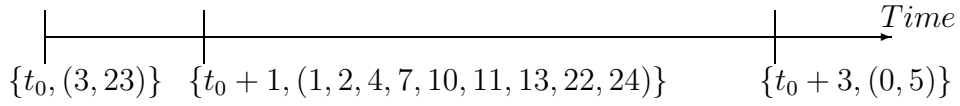


Fig. 4.10. Example of failure prediction using BGP molecules, t_0=1235877308 Unix time, Each label has {time,(list of prefix indices which fail at that time)}

4.5.2 Evaluating Prediction Quality

We now describe the technique for evaluating the quality of the failure prediction application. Let F denote the failure event of a prefix. We formulate the following hypothesis about the failure predictability of a prefix that we seek to prove or refute. Null Hypothesis H_0: F happens within a time window t when the application predicts a failure. The alternative hypothesis H_1 states the case that F does not happen given the application predicts a failure. Any evidence in support of the null hypothesis favors the success of our prediction application. We do not require exact time synchronization, since we are interested in evaluating the feasibility of the prediction application; our approach is similar to that used in [122].

We form the likelihood ratio:

$$\Lambda = \frac{P(H_1 \, is \, true)}{P(H_0 \, is \, true)} = \frac{P(No \, F \, within \, t \, | Application \, predicts \, F)}{P(F \, within \, t \, | Application \, predicts \, F)}$$

A large value of the likelihood ratio indicates that the alternative is true; hence we reject the null when $\Lambda > \gamma$ where γ is decided by using two disjoint but randomly selected sets, namely training and test sets of prefixes which are "similar" to the prefix of interest. These sets usually have 25 prefixes like Section 4.5.1, unless specified otherwise. We use the training set to find the value of γ by counting the number of instances when the alternative is true and dividing it by the number of instances where the null is true. However, γ is chosen to be at least 1, because we do not want to reject the null unless the evidence in favor of the alternative exceeds that of the null. After the value of γ is decided, we execute the same algorithm for the test set, compute Λ and reject the null if $\Lambda > \gamma$. Due to the inherent randomness in selecting the training and test sets, we perform five predictions for each prefix of interest, with different random seeds based on the current wall time so that our prediction results are not biased towards a particular choice of the sets.

Not all prediction mechanisms can function for all prefixes, for example because of an empty BGP molecule. Hence, we define *coverage* of the prediction mechanism to be the percentage of the 2353 prefixes for which a decision on predictability can be made. Out of the prefixes for which prediction is possible, the prediction methodology is either successful or unsuccessful in predicting the failures of the prefix of interest. We define *predictability* as the percentage of prefixes whose failures are predictable.

4.5.3 Naïve Prediction

We first study a Naïve prediction model which does not use BGP molecules or any prefix characteristics for failure prediction. It learns other prefixes that fail with the prefix of interest during a learning duration, and uses these prefixes to predict failure. This approach is computationally intensive since one has to find similar prefixes among 300,000 Internet prefixes. To make the problem tractable, we use our

random sample of 316 ASes containing 2353 prefixes. For each prefix of interest, we identify prefixes in the sample, which "fail along" with it during a day-long learning duration (March 1^{st}, 2009). A prefix "failing along" with another prefix implies that both prefixes fail within a time window. The window is selected to be 300 seconds to allow sufficient time for routing convergence, which has a median time of about 3 minutes [192]. Due to convergence and factors like delayed visibility of prefix failures and time synchronization of the update timestamps, we allow a 5 minute time window. A smaller time window hurts this prediction model since fewer failing prefixes are discovered.

We find 25 prefixes each for the training and test sets that fail along with the prefix of interest and predict a failure if a majority of the 25 prefixes fail within the 5 minute time window. We evaluate the success of this prediction algorithm as described in Section 4.5.2 for each of the 2353 prefixes of interest in the 316 ASes. This ensures results are not biased to a particular group of prefixes belonging to a specific AS. We use 5 random runs for each prefix of interest. Each run yields a "prediction success" result if the test set leads to $\Lambda \leq \gamma$ (learned from the training set), thereby resulting in null being accepted. Otherwise, the null is rejected and a "prediction failure" is reported. We find the predictability to be 80.6%. While this is promising, the high computational complexity of this prediction method makes it infeasible.

4.5.4 Using BGP Atoms

Before using BGP molecules, we investigate a prediction algorithm which relies solely on BGP atoms [185]. For each of the 2353 prefixes of interest, we compute the set of prefixes that have the same AS path w.r.t. every peer that sees both prefixes. Note that this means that the prefixes in the same BGP atom as the prefix of interest must belong to the same AS. The primary disadvantage of this scheme is that for most cases, we do not find any prefixes in the same atom as the prefix of interest. About 42% of the ASes have only one prefix (Section 4.2.1). Even if an AS has multiple

prefixes, it is difficult to find prefixes in the same BGP atom because multiple prefixes may be advertised with different policies for load balancing, leading to different AS paths from the same vantage point in the Internet.

The average number of prefixes in non-zero-sized BGP molecules formed using AS paths is about 11.41 vs. 2.88 for BGP atoms. The maximum number of prefixes in a BGP atom is 23 implying that we cannot run our usual prediction algorithm since we require 50 prefixes for the disjoint training and test sets. Thus, we randomly assign about half of the prefixes to the training and test sets in equal numbers, when we have at least 2 prefixes in the atom. This still only gives us a coverage of 1.66% and a predictability of 87.2%.

4.5.5 Using BGP Molecules Constructed by AS Paths

We now consider prefix failure prediction using BGP molecules constructed using AS paths as in Section 4.4.2. We investigate different failure prediction window lengths, which is the time duration in which a majority of the 25 prefixes should fail for predicting a prefix failure. We use values ranging from 60 seconds to 600 seconds. The prediction results are given in Table 4.1. The null evidence is the average number of cases where the null is true (i.e., a failure prediction coincides with the prefix failure), averaged over all the prediction runs for all prefixes of interest. The alternative evidence is defined similarly.

The prediction accuracies are higher than Naïve prediction and prediction using BGP atoms. As the failure prediction window increases, the prediction success rate reduces. At first glance, this may seem counter-intuitive. However, as the failure window increases, the evidence in favor of both null and alternative increases because there is a higher chance of a majority of the 25 prefixes to fail within a longer window. This predicts a failure, while the prefix of interest may not fail during that period. This is why the evidence in favor of the alternative increases by 64% when the window increases from 60 to 600 seconds, whereas that of the null increases by only 28% for

Table 4.1
Effect of failure prediction window on failure predictability using AS paths constructed molecules

Window (seconds)	Prediction Success(%)	Prediction Failure(%)	Null Evidence	Alternative Evidence
60	94.71	5.29	1.751	1.172
120	93.89	6.11	1.963	1.5992
180	92.07	7.93	2.144	2.187
300	91.82	5.29	2.223	2.463
600	92.05	7.95	2.241	1.919

these parameters. The alternative evidence falls for the last data point of 600 seconds, possibly because of the large window increase, since many separate failure predictions that were made are now combined resulting in fewer false predictions. Observe that the number of prefixes for which we perform prediction is 1110, hence the coverage is 47.2%. The primary reason for this low number is that BGP molecules for about 1079 or 45.86% of the prefixes do not contain any prefixes. For another 164 or 6.97% of the prefixes, the BGP molecules contain ≤ 50 prefixes.

Table 4.2 compares the performance of this AS path-based prediction with the two other prediction schemes studied so far in terms of failure predictability and coverage. The results show that using BGP molecules is the best prediction scheme studied so far with about 12% higher predictability than Naïve prediction. However, it still suffers from the disadvantage that slightly less than half of the prefixes are predictable. To remedy this problem, we study the hybrid prediction scheme in the next section.

Table 4.2
Failure predictability performance of BGP molecules constructed using three schemes; failure prediction window=300 seconds.

Scheme	Failure Predictability (%)	Coverage (%)	Disadvantage
Naïve Prediction	80.62	100	Computationally Intensive
BGP Atoms	87.2	1.66	Low Coverage
BGP molecules (AS paths)	91.82	47.2	Moderate Coverage

4.5.6 Using BGP Molecules Constructed by Hybrid Scheme

To improve coverage, i.e., percentage of prefixes for which a prediction can be made, we use the hybrid scheme (Section 4.4.4). Our "hybrid BGP molecule" is created using both geographical location and AS paths to the prefix as similarity dimensions. The "hybrid prediction scheme" operates as follows: (1) Predict failures of the prefix of interest using BGP molecules constructed using AS paths (Section 4.4.2) if they have at least 50 prefixes. (2) If the BGP molecules using AS paths are insufficient for prediction, construct molecules using geographical proximity and combine with the AS path molecule to form a hybrid molecule. This hybrid molecule is used for failure prediction, if it has at least 50 prefixes. However, for evaluation purposes, we also predict using the hybrid molecule as long as it has at least 2 prefixes.

Table 4.3 shows the coverage and predictability of this scheme along with the various components of the 2353 prefixes of interest in our 316 AS sample. Prediction using AS path molecules has a coverage of 47.17% or 1110 prefixes (Table 4.2). For the remaining 1243 cases, molecules constructed using AS paths are insufficient for prediction. Of these, 1079 cases have empty molecules and 164 cases have insufficient

prefixes. We perform prediction using the hybrid molecules for the 1243 cases in two parts. We separate the 782 cases where the hybrid molecules should be sufficient for prediction purposes, as they have greater than 50 prefixes, and achieve a predictability of 93.58%. For the remaining 461, we divide the number of prefixes in the hybrid molecule into two equal parts of training and test sets and perform failure prediction, if the molecule has at least 2 prefixes. This amounts to 444 out of 461 cases, resulting in a coverage of 18.87% and a predictability of 83.51%. Combining the prediction results of all the three cases with 1110, 782 and 444 values, we obtain a coverage of 99.28% and a predictability of 90.82%. There are only 17 cases for which we cannot perform prediction using the hybrid scheme, because the hybrid molecule has less than two prefixes. Assuming that our prefix sample was a representative one (Section 4.3.1), we conclude that the hybrid prediction scheme is the best, as it can predict failures of almost any prefix in the Internet with around 91% accuracy.

4.6 Improving CDN Availability

In this section, we study the application of BGP molecules in improving the availability of servers returned by a content distribution network (CDN). We demonstrate this application on Akamai's CDN, expecting that it can be applied to other CDNs.

4.6.1 Akamai CDN Primer

We use an example to illustrate the resolution of Akamai-hosted content by a client. Suppose a client wishes to resolve the URL *http://www.buy.com/videoclip/top-apple/81074.html* which provides information on Apple products. This URL redirects to *http://videos.buy.com/videos/buytv/2011/264/264_Apple.mp4* so the client needs to resolve *videos.buy.com*. The client starts by querying its local DNS for *videos.buy.com* and obtains the canonical name (CNAME) *videos.buy.com.edgesuite.net*. The domain *edgesuite.net* is used by Akamai for content delivery [142]. The client then queries the local DNS for this CNAME and receives another CNAME

Table 4.3
Components of the 2353 prefixes of interest in our 316 AS sample and prediction results of hybrid prediction scheme. Failure prediction window=300 seconds.

Description	Number of prefixes	Coverage (%)	Failure Predictability (%)
AS path molecules do not have any prefix	1079	45.86 %	-
AS path molecules have < 50 prefixes	164	6.97 %	-
AS path molecules are sufficient for prediction	2353-1079-164 =1110	47.17 %	91.82 %
"Hybrid" molecules having < 2 prefixes	17	0.72 %	-
"Hybrid" molecules having ≥ 2 and < 50 prefixes	444	18.87 %	83.51 %
"Hybrid" molecules having ≥ 50 prefixes	782	33.23 %	93.58 %
Total cases where hybrid molecules are used	782+444=1226	52.1 %	89.93 %
Hybrid prediction combining all techniques	1226+1110=2336	99.28 %	90.82 %

a1507.b.akamai.net in a similar fashion as the first step. Querying the DNS servers for this CNAME yields two content servers in the reply. We call this last-stage CNAME which resolves to content servers as the CNAME for short since CNAME uniquely identifies a piece of content. The servers returned by resolution of the CNAME will vary with querying clients and at various times of the day for ensuring best client performance [142]. Akamai hosts different types of content – both web sites and video

streams. The actual CNAME returned can have different forms based on the content. For example, Apple uses Akamai to host *images.apple.com* which has CNAME *a199.gi3.akamai.net*.

4.6.2 Experiments

We now describe our experiments used to show the application of BGP molecules in improving availability of Akamai's content servers. We seek to identify a set of Akamai CNAMEs with adequate breadth that will be used in this study. We start with known Akamai CNAMEs like *a1507.b.akamai.net* and observe whether changing the number (1507) or the letter (b) gives us a CNAME which resolves to an Akamai content server. The number corresponds to a channel [193], whereas the letter corresponds to the way channels are grouped. Using the above technique and observed CNAMEs for Akamai-hosted content, we discover eleven Akamai CNAME patterns, listed in Table 4.4. We find that for each of the patterns, channel numbers 0 to 4094 lead to valid CNAMEs. For example, *a0.vmg0.akastream.net* to *a4094.vmg0.akastream.net* represent valid CNAMEs resolving to Akamai content servers. This fixed number of 4095 for each of the eleven CNAME patterns is $2^{12} - 1$ which may reflect the number of bits used to store channel numbers. In the remainder of this section, we use eleven random CNAMEs, one from each pattern of Table 4.4, with a random number between 0 to 4094 as the channel number. These CNAMEs used are shown in the table as well. For example, *a3516.c.akamai.net* represents a CNAME that we use for our study from the second pattern.

To achieve credible results, we query the eleven random CNAMEs of Table 4.4 from different clients spread all over the world. We use 353 PlanetLab [186] nodes as clients in our experiments. Each node queries its local DNS servers for each of the eleven CNAMEs one by one in a loop, for about six weeks beginning August 28^{th}, 2011 and ending October 8^{th}, 2011. The period of querying each CNAME ranges from around 30 seconds to little over a minute. The query, if successful, returns two

Table 4.4
Akamai CNAMEs studied in this section with the actual CNAME used

CNAME pattern	Actual CNAME used
x = 0 to 4094 for all rows unless specified otherwise	
a{x}.b.akamai.net	a2561.b.akamai.net
a{x}.c.akamai.net	a3516.c.akamai.net
a{x}.f.akamai.net	a3886.f.akamai.net
a{x}.h.akamai.net	a3417.h.akamai.net
a{x}.k.akamai.net	a246.k.akamai.net
a{x}.l.akamai.net	a2225.l.akamai.net
a{x}.p.akamai.net	a714.p.akamai.net
a{x}.vmg0.akastream.net	a818.vmg0.akastream.net
a{x}.vmg2.akastream.net	a3237.vmg2.akastream.net
a{x}.uqg0.kamai.net	a3994.uqg0.kamai.net
a{x}.gi3.akamai.net	a40.gi3.akamai.net

content servers to which round-trip times (RTTs) are measured from the client by taking the minimum RTT of three ping packets. Both the servers and their RTTs are recorded in each iteration.

The experiments running on 353 PlanetLab nodes over a six week duration for eleven CNAMEs together complete around 240 million iterations. On the average, there are approximately 62100 iterations per CNAME per node over the six week period. We find that there are 3864 unique six-week long queries for a CNAME from a node, which we represent as a (CNAME, node) tuple.

We first investigate whether the two content servers returned by Akamai in response to a query belong to the same prefix. This is important because BGP molecules are constructed from routing data, and hence are on the granularity of prefixes. The IP-address-to-prefix mapping is performed using Cymru [194]. We collect the IP

addresses from each of the 240 million iterations and find 11,100 unique IP addresses with an average of 31 IP addresses per node observed during the six week period. These 11,100 IP addresses are then mapped to their corresponding prefix using Cymru. We then go through each of the iterations ascertaining whether the two IP addresses returned in response to a query for a CNAME belong to the same prefix or not. We compute the number of the 3864 six-week long (CNAME, node) queries for a CNAME through a node that see IPs from a different prefix (in response to a single query) at some point during the six-week measurement period. We find that 58.5% of the 3864 queries always see IPs from the same prefix. For each of the 3864 queries, an average 2.25% of the iterations see server IPs from different prefixes.

The number of Akamai prefixes seen by a six-week long (CNAME, node) query has a median value of 12 and a mean value of 14. The number of prefixes seen per node (aggregated across the CNAMEs queried) has a median value of 74 and a mean of 78.5. There are a total of 594 Akamai prefixes seen in our entire experiment over all nodes and CNAMEs for the six-week period.

Each content query yields two Akamai servers, which we refer to as a *prefix pair*. The prefix pair may consist of the same prefix repeated twice and we do not distinguish pairs based on the order of prefixes in the pair. We now go through each of the 3864 six-week long (CNAME, node) queries to find the time a prefix pair is used by Akamai to deliver a particular CNAME's content to a particular client. A prefix pair can be reused for many successive queries until Akamai changes at least one of the prefixes in the pair. If a prefix pair is first used at time t_1 and is replaced by another prefix pair at time t_2, we consider $t_d = t_2 - t_1$ as the time when the prefix pair is used. It is worth noting that a prefix pair may not be used for some duration and may be reused again after some time, in which case a new t_1 is recorded. We compute these times t_d for each of the 3864 six-week long queries and find that its mean value is 1256 seconds while its median value is 123 seconds. The mean is affected by outliers, hence median is a better estimate of the time a prefix pair is used. In the median case, a prefix pair is changed every two minutes, which confirms the dynamicity of

Akamai's content distribution network, which is known to react quickly to network conditions and load [140]. This median value of two minutes also matches the median redirection time of 100 seconds noted by the measurement study in [140].

4.6.3 Availability of Akamai's CDN

Before suggesting improvements to Akamai's content distribution scheme, we need to evaluate its availability. For this purpose, we first collect routing tables and updates from RouteViews [9] for the months of August through October. We then filter table transfers using the technique of Section 3.3. For each of the 594 Akamai prefixes seen during our experiments, we note whether they are in Announced or Withdrawn state in the control plane during the experiment period. For each prefix, we initialize its state using the routing table of August 28^{th} right before our measurements began, and then use the BGP updates to record its state changes and the corresponding times of the changes. If an update does not change the prefix state, we ignore it, since we are interested in computing the availability. If the prefix is not found in the routing table, its initial state is unknown until an update is seen for the prefix. From the RouteViews updates for the same time as our measurement period, we obtain information on the states of 584 out of the 594 Akamai prefixes. We find that 76% of those prefixes have only one state recorded for the six-week period, while 92% of the prefixes have less than or equal to 3 states. This is consistent with the observation made in Section 4.3.2 that prefixes are up most of the time.

We compute the availability of the current Akamai CDN for each of the 3864 (CNAME, node) queries. For each time period t_d that a prefix pair is announced (as computed in the Section 4.6.2), we compute the time t_a for which the prefix pair is available in the control plane. A prefix pair is considered available if either of the prefixes in the pair is in *Announced* control-plane state. We then add all the t_a's and the t_d's for the six-week measurement period and compute the availability of the

Akamai's scheme as $Availability = \frac{\sum t_a^i}{\sum t_d^i}$ where t_a^i is the i^{th} t_a value, the time a prefix pair is available out of the time t_d^i it is used by Akamai.

Having computed a single availability value for each of the 3864 (CNAME, node) queries, we compute the statistics of these values and obtain a median of 100% and a mean of 99.9879%. These numbers are high, which is to be expected since prefixes are mostly in the *Announced* state and Akamai reacts to network conditions. However, we find that there are several cases where the availability is not perfect. Only 89.8% of the queries have 100% availability, and 9.68% of the queries have less than five-nines availability, which is a benchmark value in dependability literature [175]. In the following sections, we show how we can improve these availability values by using BGP molecules.

4.6.4 Constructing Inverse BGP Molecules

To improve the availability of Akamai's content distribution scheme, for each prefix of interest whose server is used for delivering content, we need to find candidates for backup prefixes, which are unlikely to fail together with the prefix. Hence, we are interested in determining the *inverse BGP Molecule* of a prefix, *i.e.*, the prefixes which do not exhibit similar failure characteristics as the prefix. The candidates of an inverse molecule of a prefix of interest should be the prefixes whose content servers have served the same CNAME during our experiments.

We collect the IP addresses associated with each of the eleven CNAMEs from Table 4.4 across all PlanetLab nodes. We find that an average of 1509 and a median of 1177 IP addresses are associated with a CNAME. We use Cymru's IP to prefix mapping as above and find that a CNAME is associated with around 228 prefixes on the average (230 in the median case). In a real deployment scenario, Akamai's CDN can make such a list of prefixes which serve a particular CNAME and use that to construct inverse molecules.

We construct the inverse molecules using AS paths via the technique in Section 4.4.2. We construct two sets of molecules, one using the routing tables from August 2011 before our experiments began and another from September 2011 when our experiments were underway. For each of the 28 days of the August 2011 set, we select the largest routing table for each day and merge these tables to form a combined August 2011 table, which has 437,361 unique lines. We then extract the prefixes and the corresponding AS paths from these combined routing tables, and remove AS path prepending and the first and last AS of the AS path as in Section 4.4.2. For the September 2011 data, the combined routing table has 773,291 lines, which is also processed like August 2011. We use these routing tables for all further processing and call the processed AS paths as AS path sequences.

For each of the 594 prefixes, we go through each of the candidate prefixes in its inverse molecule, which serve the same CNAME and choose it to be in its inverse molecule if there are no AS path sequences in the routing table which are the same for the two prefixes. Empty AS path sequences which can be caused by removal of first and last AS in the path and AS prepending are ignored. We find that for August 2011 data, on the average 87% (median 88%) of the prefixes have at least one prefix in its inverse molecule. Empty inverse molecules can be due to a prefix not being found in the routing table or all candidate prefixes having at least one same AS path sequence. The corresponding numbers for September 2011 are mean and median of around 96%. Given a prefix has at least one prefix in its inverse molecule, the number of prefixes in an inverse molecule is around 160 for the mean and median for both the August and September 2011 datasets. These results show that there are usually sufficient number of prefixes in an inverse molecule to give our new content distribution schemes (described in Sections 4.6.5 and 4.6.6) enough choices for backup content prefixes.

We now investigate the stability of inverse BGP molecules by comparing those constructed with August 2011 data to those constructed with September 2011 data. Specifically, we seek to determine whether a prefix which has a non-empty inverse

BGP molecule in August 2011 also has one in September 2011 and if so, the proportion of prefixes in its inverse molecule in August 2011 which also exist in September 2011. We find that *all* the prefixes which have a non-empty inverse molecule in August 2011 also have one in September 2011. Our results also indicate that on the average 95% (median 98%) of the prefixes existing in an inverse BGP molecule of a prefix in August 2011 also exist in September 2011. This implies that the inverse BGP molecules are stable, which helps our proposed content distribution schemes of Sections 4.6.5 and 4.6.6. This is because inverse molecules can be constructed in a fairly offline fashion say once every month and can be used in a content distribution scheme which needs to be resolve client queries quickly. In what follows, we use the inverse molecules constructed from August 2011 data before our experiments began to evaluate improved content distribution schemes over the course of the next six weeks of our experiments.

4.6.5 Improving availability of Akamai's CDN

We now describe the first of our techniques to improve the availability of Akamai's content distribution scheme based on the inverse BGP molecules constructed in Section 4.6.4. Our schemes are simple variants of the original Akamai scheme *i.e.* Akamai's proprietary algorithms used to select content servers are still used. These algorithms supposedly work best for Akamai and take into account internal factors and network conditions which we cannot ascertain. Our scheme merely replaces the second backup server returned by a query with another one for improving availability. We go through each of our queries which are successfully resolved in our six-week of measurements and keep the primary server the same as that returned to us by Akamai, while replacing the secondary server with a randomly generated one from the inverse BGP molecule of the primary server's prefix. Choosing a prefix randomly makes this scheme very attractive for online implementation since one can compute inverse BGP molecules in an offline fashion as demonstrated in Section 4.6.5. In case

the inverse BGP molecule of the primary server's prefix is empty, we keep the query's result the same as that obtained in our experiments.

We now compute the availability of this scheme by using a technique similar to Section 4.6.3. Hence, a prefix pair used in our new scheme is considered unavailable if both the prefixes are unavailable in the control plane. We find that the average availability of the 3864 (CNAME, node) queries increases to 99.9954%. However, the real advantage of this scheme is revealed when we look at the percentage of queries with availability compared to a threshold. We find that only 0.33% of queries have availability less than five-nines which is a reduction of 96.6% w.r.t. the original Akamai scheme (Section 4.6.3). Also, 99.53% of the queries have perfect 100% availability which is an increase of 10.8% w.r.t. the results of Section 4.6.3. Thus, we conclude that our modified scheme is successful since it achieves *perfect* availability for nearly *all* the queries over the six-week period.

Such a scheme with high availability may not be worthwhile to implement if clients are now redirected to servers which have much higher latencies than the servers of the original Akamai scheme. Low latencies to servers are very important in a content distribution scheme, especially since Akamai is used for delivering dynamic content like video and audio streaming [195]. We compute the latency inflation of this scheme by computing the difference between the average latency to the client and the prefixes in the prefix pair of the new scheme versus the original scheme. We estimate the latency of a node to an Akamai prefix, either by choosing the median value of the latency measured to IP addresses belonging to the prefix during our six-week long measurements, if available, or by choosing the median of a day long measurement of ping latencies to IP addresses within the prefix. We stress that these are only latency *estimates* to the prefixes computed to evaluate the effectiveness of our scheme with no impact on the scheme itself. We compute the statistics of the average latency inflation of our scheme across all queries and obtain a median value of 10.34 ms with median percentage latency inflation of 17.4%. This inflation is likely because we have randomly chosen a secondary prefix without any consideration of the latency from the

client to the prefix. We will determine if it is possible to design an informed scheme which reduces latency while increasing availability in Section 4.6.6.

4.6.6 Latency-aware scheme

We now assume that Akamai's content distribution scheme knows the approximate latency from a client to all possible Akamai prefixes. We discuss implications of this assumption below. We modify our content distribution scheme of Section 4.6.5 by choosing the secondary server's prefix to be the one from the inverse BGP molecule with the least latency from the client which initiated the query. The latency of the client to the prefixes is computed as the median of the measurements as in Section 4.6.5. If the inverse molecule of the primary server's prefix is empty, we retain the query results of the original measurements. From the latency perspective, this is the best variant of the content distribution scheme proposed in the previous subsection.

The performance of this scheme along with other schemes studied in this section is shown in Table 4.5. We evaluate the availability of this scheme as in Section 4.6.5 and obtain a mean availability of 99.9948%. We find that only 0.58% of our (CNAME, node) queries have less than five-nines availability, while 99.25% of the queries have perfect availability. Thus, the availability results are only marginally worse than the random prefix scheme of Section 4.6.5. We also evaluate the latency of this scheme w.r.t. Akamai's original content distribution scheme and find that the latency is *lower* by 12.31 ms and 39.83% in the median case. This shows that, given the latencies of a node to Akamai prefixes, one can improve both availability and latency.

While the assumption that Akamai knows the latency from a client to all its prefixes is unrealistic, Akamai tries to achieve this through active measurements [140]. Our results from Table 4.5 show that the performance of a scheme which knows the latencies is *significantly* better than the original Akamai scheme with latency savings of around 40% in the median case. Hence, we posit that Akamai does not need to know the latencies accurately; even an estimate of the latencies should lead to

Table 4.5
Results of all variants of content distribution schemes

Scheme Description	% queries below five-nines availability	% queries with 100% availability	Median latency difference w.r.t. Akamai's scheme	Median latency % difference w.r.t. Akamai's scheme
Akamai's scheme	9.68%	89.8%	-	-
Using random inverse molecule prefix	0.33%	99.53%	10.34 ms	17.37 %
Using least latency inverse molecule prefix	0.58%	99.25%	-12.31 ms	-39.83 %

a scheme which will perform better than the original Akamai scheme by increasing availability and likely reducing latency. A scheme which uses estimates of latency will have results somewhere in between the results of the second and third rows of Table 4.5, and will be closer to the third row if the estimate is closer to the real one. Hence, we conclude that BGP molecules are a very important tool in improving the availability of a content distribution scheme without any significant latency hit.

4.7 Chapter Summary

This chapter has focused on clustering prefixes using similarity in their failure tendency with the primary goal of predicting failures. We found the prefix characteristics of geographical location and AS paths to the prefix to be good indicators of the failure tendency of a prefix. We use these characteristics to construct a group of prefixes similar to a prefix of interest called a "BGP molecule," which can be used to predict failures of a prefix of interest. To the best of our knowledge, ours is the first

work to evaluate the similarity of prefixes in the Internet w.r.t. their failure tendency, and demonstrate applications to failure prediction and content distribution networks.

We evaluate four schemes to predict failures of a prefix of interest. We find that a scheme that naïvely observes past failures is impractical for online prediction. BGP atom-based prediction has low coverage so it cannot be used for most prefixes. AS path-based prediction achieves high predictability at a moderate coverage. A hybrid scheme based on AS paths and geographical location performs the best with about a 91% predictability and 99.3% coverage, and thus is useful for predicting prefix failure in the Internet. Success of our control plane failure prediction mechanism coupled with the results of [122,123] imply that our technique should also be fairly reliable at predicting data plane failures.

Our work does not aim at clustering the entire set of prefixes in the Internet – a computationally intractable task for an online algorithm. Rather, we focus on predicting failures of a few prefixes by tracking the failures of prefixes in their molecules. In most of our experiments, we have used a random set of prefixes from the molecule showing that once the molecule is constructed, we can change the set we track periodically or on-demand (e.g., if our failure prediction rate is low) and still achieve good performance. However, tracking failures based on RouteViews data is non-trivial. This is because RouteViews updates are published at 15 minute intervals, which is longer than a reasonable prediction time of a prefix failure. The routing updates of some prefixes in the molecule must be available through an a priori mechanism. This is why we prefer construction of BGP molecules using relatively static approaches such as AS paths in routing tables and geographic location, over tracking failures in real-time as in the Naïve prediction scheme. If such a setup exists, we can predict failures in about the time it takes for BGP convergence, which is on the order of a few minutes.

We investigate another application of BGP molecules in improving the availability of a content distribution network using Akamai as a case study. We perform a six-week measurement study querying eleven Akamai CNAMEs from 353 PlanetLab nodes,

recording the two content servers returned per query. We study the control-plane availability of the content distribution network using BGP data, and find that while the availability of most queries is high, there exist a significant number of cases with availability lower than five-nines. We then propose a new approach where the secondary server is replaced by one within a prefix which is in the inverse BGP molecule of the primary prefix, and hence is unlikely to fail with it. We find that a content distribution scheme which chooses secondary servers in a smart fashion can simultaneously increase the availability and reduce the latency with respect to Akamai's scheme.

5. DNS IN THE CLOUD

In this chapter, we consider the implications of the evolution of Domain Name System, which is one of the primary components of the Internet infrastructure (Section 2.1.2). We consider the evolution of DNS to the cloud, which is an organization potentially with multiple worldwide data centers (Section 2.4.2). The DNS service can be used for the organization's internal use for Global Traffic Management to redirect users to the appropriate server (Section 2.4.1), and we study an example of that in Akamai's CDN in Section 5.2. The DNS service can also be provided by the cloud to any Internet client as an external DNS system, and we conduct a measurement study of Google external DNS in Section 5.4, leveraging our technique for geolocating data centers of a cloud in Section 5.3. Finally, we consider how an Internet client can obtain good performance while retrieving content from a highly distributed CDN like Akamai through an external DNS like Google DNS (Section 5.5).

5.1 Motivation

With the emerging trend of cloud computing, external DNS services are being offered through the cloud, for example, Google public DNS [41] and OpenDNS [42]. They are external in the sense that they are being offered not as a part of Internet service provided to the client but as a standalone service provided by an Internet cloud. These provide DNS choices to a client as an alternative to the DNS provided by ISPs.

For example, Google has been offering public DNS services [41] since 2010. This enables clients anywhere in the Internet to convert Internet hostnames to IP addresses. Google provides two IP addresses that clients can use as their DNS servers, namely 8.8.8.8 and 8.8.4.4. Google's DNS has been rapidly gaining market share

since its launch [139], which makes it a representative candidate for studying external DNS, as we have done in this chapter. Google advertises several benefits of using its DNS servers, including added security and performance. Because of its extensive infrastructure and global presence, Google DNS can resist various forms of security attacks [154]. Google also claims client performance improvements [137] in using its DNS servers, because of adequately provisioned servers and prefetching of name resolutions to avoid cache misses. However, it does acknowledge the possibility of slow browsing for certain sites, because of websites (like Akamai) which return servers according to the resolver's IP address. We study this in Section 5.5 and suggest solutions to achieve better client performance while using external DNS.

Since its inception in the year 1998, Google has been steadily growing, starting from a search company into an organization with an increasing role in the networking community [196]. The search giant is recently pushing into providing networking services like broadband Internet through using its own optical fiber networks [197]. Google also operates its data centers at several locations around the world, aiming to provide low latencies to clients located anywhere [66].

Google traffic on the Internet is increasing, not only due to its search engine but also because of the acquisition of YouTube in 2006. Labovitz of Arbor Networks estimates that as of summer 2009, Google was carrying about 6-10% of Internet traffic globally [198]. According to Labovitz, only two big ISPs carry more traffic than Google, and a significant percentage of traffic is Google transit [199]. Google is the fastest growing Autonomous System Number (ASN) group in terms of traffic volume [199]. Thus, a study of Google's data center network conducted in this chapter along with a study of Google public DNS (Section 5.4) will shed light onto Internet traffic as a whole.

5.2 DNS of Content Distribution Networks

Content Distribution Networks (CDNs) need to operate their own DNS infrastructure to enable clients to be redirected to an appropriate server, determined by Global Traffic Management (Section 2.4.1). In this section, we study the two-level DNS infrastructure operated by Akamai, one of the largest CDNs. Other CDNs can certainly have different DNS architecture, *e.g.* Huang *et al.* found that the CDN Limelight uses a single level DNS infrastructure, with the DNS nameservers mapping to the same IP addresses when queried from various worldwide clients [200]. These nameserver IP addresses are announced using IP anycast [136]. As we shall see in this section, this is different than Akamai, whose DNS server names map to different IP addresses when resolved from different worldwide locations.

Akamai uses two levels of DNS servers to redirect clients to the closest content server [142]. We use the same example as in Section 4.6.1 and illustrate the steps involved in the resolution process. The client wishes to resolve the URL *http://www.buy.com/videoclip/top-apple/81074.html* which redirects to *http://videos.buy.com/videos/buytv/2011/264/264_Apple.mp4* so the client needs to resolve *videos.buy.com*. Figure 5.1 depicts the steps involved in the resolution of this URL by the client.

The client starts by querying its local DNS for *videos.buy.com*. We omit the details of this query in the figure for brevity, since it is not the final stage query. Either the local DNS knows the answer from its cache, or it queries top level and Akamai DNS servers and returns the canonical name (CNAME) *videos.buy.com.edgesuite.net*. The client then queries the local DNS for this CNAME and receives another CNAME *a1507.b.akamai.net* in a similar fashion as the first step.

We now use the command *dig +trace* [201] from the client to resolve this CNAME so that we can follow the referrals from the root servers to the Akamai DNS servers, eliminating caching at the local DNS server. However, the local DNS server is involved in the following queries to obtain IP addresses of the servers involved e.g.,

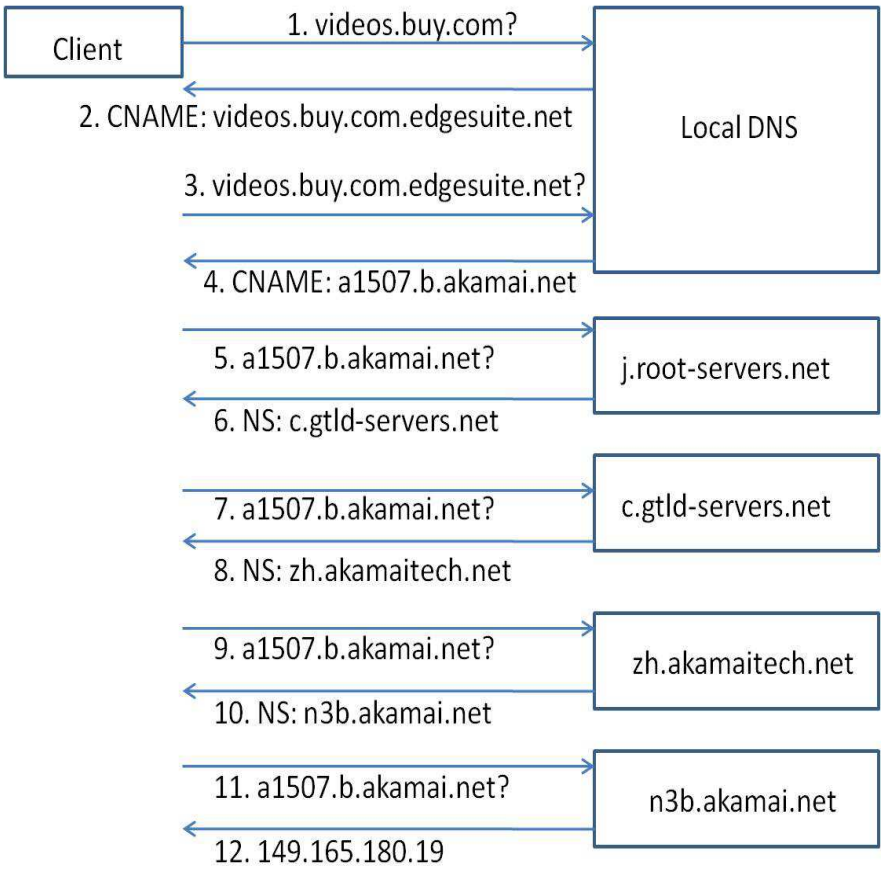


Fig. 5.1. Steps taken by a client in obtaining content server for an Akamai-hosted website

j.root-servers.net. The client queries the top level domain server *j.root-servers.net* for *a1507.b.akamai.net*, which returns a list of nameservers authoritative for *akamai.net* out of which the client chooses *c.gtld-servers.net* and queries it. This gives a list of nameservers controlled by Akamai which are its top level nameservers. The client chooses *zh.akamaitech.net* for the query in the next step. This server returns Akamai second level nameservers which are dependent upon the client's location (i.e., proximity-aware). Overall, there are nine second level nameservers for this CNAME, from *n0b.akamai.net* to *n8b.akamai.net*. The client then chooses *n3b.akamai.net*, querying it for *a1507.b.akamai.net* which returns the content server 149.165.180.19.

While Akamai usually returns two content servers for each query, we use the first one in this chapter.

The IP address of the second level nameserver *n3b.akamai.net* depends on the client's local DNS location and is different for different clients. This is because the Akamai nameservers, both first and second level, redirect the client to the best possible (based on location and network conditions) nameservers and content servers. Hence, the client achieves good DNS resolution performance and fast content retrieval.

The CNAME and its associated nameserver can have different forms based on the content. For example, Rush Radio 94.5 uses Akamai for audio streaming with CNAME *a23.vmg0.akastream.net*. However, the resolution of these different CNAMEs follows the pattern of Figure 5.1, with corresponding changes. For example, the second level nameservers of *a23.vmg0.akastream.net* are *n0vmg0.akamai.net* to *n6vmg0.akamai.net*.

5.3 Technique for Geolocating Cloud Data Centers

Extensive research exists on geolocating IP addresses in the Internet [202]. A detailed discussion on geolocation techniques is outside the scope of this dissertation. In this section, we discuss how to geolocate IP addresses of cloud providers, with the goal of discovering the data centers of the cloud. Throughout this chapter, we use the commercial geolocation tool GeoIP City provided by MaxMind [203] to geolocate IP addresses. We use the web-based lookup service MaxMind GeoIP City/ISP/Organization Web Service available at [204]. This geolocation is accurate up to 25 miles for most IP addresses [205].

Using this service, we can easily geolocate Akamai content servers and nameservers with reasonable accuracy, which gives us locations of Akamai's data centers. For example, in Figure 5.1, we can geolocate the end-server 149.165.180.19 to Bloomington, Indiana, which is found to be 85 miles away from our client IP's location (also

found using MaxMind). We compute the distance between two locations, given their latitudes and longitudes using Haversine Formula [190] for computing the great-circle distance. The RTT of the server from the client is found using ping to be 1.5 ms, which is reasonable given the expected Geo-RTT of 1 ms between the nodes. The Geo-RTT is computed based on the estimate that the bits travel through the Internet at $4/9^{th}$ the speed of light in vacuum [66].

However, there are cases when a simple geolocation of IP addresses will not give expected locations using a geolocation tool. For example, as we show in Section 5.4, most of Google's IP addresses resolve to its headquarters in Mountain View, California. The results seem inaccurate given Google's well-known global presence. Besides, there are cases when a single IP address is announced from all over the world through IP anycast, like Google's public DNS address 8.8.8.8. Maxmind's geolocation technique will be unsuccessful in locating data centers in that case as well. One of the solutions to this problem is presented in [139], who attempt to geolocate Google DNS data centers. The authors embed Javascript code in a popular website which causes Google DNS to reveal its IP to DNS servers under the authors' control, while resolving the website on behalf of clients visiting it. This requires an infrastructure setup and is passive in that it requires a period of measurements beyond the authors' control where clients using Google DNS visit the website from various locations around the world.

Rather than relying on extensive infrastructure and measurements over a significant period of time, we design a novel lightweight *active* technique for geolocating cloud data centers. We use PlanetLab [186] nodes spread all over the world to run traceroutes to the "target" IP address which we wish to geolocate, for example Google's public DNS IP address 8.8.8.8. We define VTIP, which is the *Virtual Target IP*, as the last hop right before the target IP in the traceroutes. We then geolocate these VTIPs using MaxMind [203] and determine unique locations for these VTIPs. To geolocate cloud data centers, we use hierarchical clustering algorithms [174] to cluster the unique latitudes and longitudes of VTIPs using Matlab [206]. We com-

pute the distance between two locations using Haversine Formula [190]. and cluster them using the agglomerative complete link clustering technique [174], where the distance between two clusters is determined by the distance between its farthest points, using 50 miles as the cutoff distance between clusters. Since the accuracy of MaxMind is 25 miles, two IPs at the same location can be no more than 50 miles apart. The clusters returned are the cloud data centers. We present the results of this technique in Sections 5.4 and 5.5 and find that our technique is fairly successful at locating appropriate number of cloud data centers as is publicly known. While this technique does suffer from measurement errors and its accuracy will depend on the number of Planetlab nodes used and the duration of the measurements, its lightweight active nature makes it a good choice for geolocating data centers of clouds, that seldom divulge its data center locations to the public.

5.4 Measurement Study of Google DNS

In this section, we describe our measurement study of various facets of Google's public DNS. One of the metrics we use to evaluate Google is whether it redirects clients to its closest data center. A key challenge in evaluating that metric is that the Google data center locations are unknown. To the best of our knowledge, Google does not make its data center locations public. There are a few blogs which record data center locations [207, 208], but the credibility of their information is unclear. Moreover, not only do we need the data center physical locations but also their IP addresses, in order to measure network characteristics to these addresses. In this section, we address the challenge of geolocating the data centers by using our technique from Section 5.3.

After we geolocate the data centers and compose a representative set of IP addresses for the data centers, we conduct a series of experiments to analyze the performance of public DNS. We investigate the performance of Google public DNS by looking up web sites through both native and Google DNS and comparing the results. We measure the round trip latency to the servers returned, and the Time-to-Live

(TTL) value of the DNS entries returned along with the DNS query time. We study lookup results for both highly popular and less popular web sites, including some search competitors of Google.

We conduct another set of experiments to investigate the claim that queries to Google public DNS reach the closest data center. A client reaching the closest data center may or may not experience the lowest latency due to a variety of reasons, such as queuing delay and interdomain routing issues [66]. To represent the Google data centers that correspond to public DNS, we run traceroutes to the public DNS IPs from clients spread around the world for a period of a few days, and note the IP address of the node right before the Google public DNS IP in the traceroute. We then collect the list of addresses and geolocate them to compose a representative list of addresses according to our technique of Section 5.3. We measure latencies to the representative IPs of Google public DNS around the world, and compare them to the latency observed to the Google public DNS IP.

Finally, we study the performance of the most famous Google application, Google search, performed on the server returned by native DNS and by Google public DNS. The reason for using Google public DNS for Google searches is to validate whether this DNS server returns an optimal Google server in terms of application performance. Since by using this DNS, all steps of a search proceed through Google's network, one can assume that this yields the best available server. We also record the packets exchanged during the TCP session of the Google search between the Google Front End (GFE) and the client, and observe the trends in the Google search time reported by Google, and the overall session time.

We utilize PlanetLab [186] in our experiments, in order to leverage clients geographically spread in the world. This is important since we aim to study Google's global network of data centers. We use all the PlanetLab nodes that were available to use at the time of starting our experiments (June 2010). In total, we have 688 nodes spanning 44 countries and 268 unique locations (as determined by the latitudes and

longitudes). The distribution of the continents to which the nodes belong to is shown in Table 5.1.

Table 5.1
Continent Distribution of the PlanetLab Nodes

Continent	Number of Clients
Asia	95
Europe	242
Africa	4
North America	315
South America	20
Australia	12
Total	688

5.4.1 Insight into Google's Network

We start with the Internet Routing Registry (IRR) [209] for obtaining information about Google's network. We look up the WHOIS of the five Internet Regional Internet Registries (RIRs), namely ARIN [210], RIPE [211], APNIC [212], AFRINIC [213] and LACNIC [214], and record the range of IP addresses and Autonomous System (AS) numbers allocated to Google. The Whois lookups result in the ASes 15169, 36039, 36040, 36384, 36385 definitely belonging to Google. ASes 8551 and 3552 likely belong to Google as reported by RIPE Whois; however they are also reported to be maintained by other organizations. There are also several IP address ranges reported by the Whois results which are not reported as belonging to any AS.

This technique of looking up IP ranges through Whois is, however, fraught with errors. There exists the possibility that the databases are incomplete. Moreover, it is not clear which IPs will be used by Google and for which purposes. Hence, we devise an experiment to discover the Google IPs actually used as described below.

Through each of the available 688 PlanetLab nodes, in a single iteration, we lookup the Google front page www.google.com, and record the IP address returned. We then run traceroutes to the IP address(es) returned. We use Paris traceroute because of its ability to yield a single consistent path in the presence of load balancers. We conduct 300 iterations, one every 15 minutes, thereby running cumulatively for a period of three days.

We now combine the IP addresses of Google Front Ends (GFEs) obtained during this experiment across all the PlanetLab nodes. This gives 232 unique IP addresses, i.e., about one unique IP per three PlanetLab nodes. We geolocated these IP addresses using MaxMind, and discovered that MaxMind shows the IP addresses belonging to only eight unique locations, as determined by latitude and longitude. Seven of the eight locations are located in the United States (US), and three of those are in the Mountain View, California, area. We verified these results through geolocation of these IP addresses using other freely available services such as IP2Location [215]. We conclude that the results are not accurate, since Google operates more than eight data centers in the US alone.

This experiment demonstrates the difficulty of geolocating Google's data centers. In fact, similar observations have been made in blogs, e.g., by [207], which claims that nearly all of Google's IP addresses resolve to Mountain View, CA, the headquarters of Google. Looking back at our 232 IP addresses of GFEs, 206 or about 89% appear to be in Mountain View, California. This percentage is uncharacteristically high and seems to support the secretive nature of Google about its data centers as mentioned in [207]. While we do not know the exact techniques used by Google for the IP addresses of its GFEs, we believe that one possibility can be that the IP addresses are anycast, thereby leading to the closest data center while geolocating primarily to Mountain View, CA.

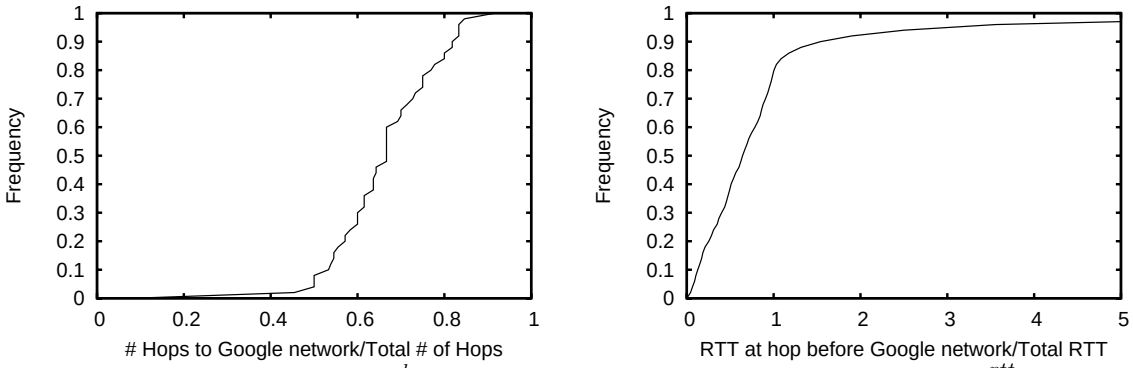
We now proceed to analyze traceroutes conducted from the PlanetLab nodes to the Google IPs we obtained. We seek an answer to the question: how many hops in a typical traceroute are traversed within Google's network and how many are traversed

outside it? To identify Google's network, we use RouteViews [9], which contains routing tables containing the prefixes advertised by various ASes using the Border Gateway Protocol (BGP). We use the routing tables from May and June 2010, and collect the prefixes advertised by the five confirmed Google's ASes at any time during this period. We consider these prefixes as the most accurate and latest indication of the IP addresses used by Google (more accurate than Whois databases which may contain inaccurate and outdated information).

This technique yields 892 prefixes belonging to Google, which include the two prefixes of the Google public DNS, namely 8.8.4.0/24 and 8.8.8.0/24. For each of the hops of the traceroutes to the GFEs, we determine if the IP address lies within the prefix of any of the 892 prefixes. If so, we declare the IP to be inside Google's network. We use about 438,000 traceroutes, combined across all PlanetLab nodes, for further processing in this section. Hence, if there are h_t total number of hops in the traceroute, and h_{bg} number of hops before entering Google's network, the hop at $h_{bg} + 1$ will be the first hop inside Google's network. We record these numbers for each of the traceroutes (across all PlanetLab nodes), and then compute the ratio $\frac{h_{bg}}{h_t}$, which we call the *Pre-Google Hop Ratio*. The CDF of this ratio is shown in Figure 5.2(a) with both mean and median being around 2/3. The plots show that on the average, traceroutes to a Google Front End traverse $1/3^{rd}$ of the hops inside Google's network. An average traceroute to a Google IP takes about 11.9 hops, so about 4 of those hops are traversed inside Google.

We also record the Round Trip Time (RTT) of each of the hops of the traceroute by averaging the RTT of the available values out of the three ICMP packets sent by Paris traceroute for each hop. Some packets may be dropped in the forward or return path leading to no RTT value; we ignore those packets for the RTT computation. We then compute the ratio of the RTTs of the hop right before Google's network rtt_{bg} and the last hop (Google Front End) rtt_t of the traceroute. The CDF of the RTT latency ratio is shown in Figure 5.2(b). The median value is around 0.64 while the mean is around 1, affected by the high values of the RTTs obtained in certain cases. We find

that about $1/3^{rd}$ of the latency is spent inside Google's network on a median basis, which is consistent with the number of hops inside the Google's network. There are a few high values of the ratio, even greater than 1. High RTTs are not uncommon in measurements [66] and queuing delays on the forward or the reverse path can easily cause the RTT of a hop to be higher than a subsequent one in a traceroute, since each hop is probed by different packets. However, the median values, not affected by outliers, exhibit consistency between the number of hops and the latency.



(a) Pre-Google Hop Ratio $\frac{h_{bg}}{h_t}$ (b) Pre-Google RTT Ratio $\frac{rtt_{bg}}{rtt_t}$

Fig. 5.2. CDFs of hops and delays from traceroutes to Google Front Ends

5.4.2 Geolocating Google Data Centers

As discussed above, geolocating the GFEs yields inaccurate locations for Google data centers. Hence, we use the technique of Section 5.3 to geolocate Google data centers. We parse all the traceroutes conducted to GFEs from PlanetLab nodes and call the last but one hop of the traceroute the *Virtual Google Front End* (VGFE), since it is not the actual front end, but the hop right before it in the traceroute towards the GFE. Clearly, the mapping between a VGFE and a GFE is many-to-many, since multiple VGFEs can exist for one GFE and vice-versa.

We obtain 790 unique VGFE IP addresses from the traceroutes collected from the PlanetLab nodes. This number is about 3 times the corresponding number of GFEs and is a little more than one per PlanetLab node, which implies that these IPs are unlikely to be anycast. We geolocate them using MaxMind [203] and obtain 106 unique locations (determined by latitude and longitude). Out of these, 45 unique locations are in the US. We the use hierarchical clustering algorithms [174] to cluster the 106 unique latitudes and longitudes of VGFEs as described in Section 5.3. This gives us 75 clusters out of the 106 unique locations. We also cluster the 45 unique locations in the US using the same technique and obtain 25 clusters. This number is consistent with the 19 US data center locations reported in [208], considering that this number was reported around 3 years ago. While our technique is certainly not perfect for geolocating Google data centers, in the absence of public information and due to the problem of geolocating GFEs, we consider these results of geolocating VGFEs promising.

We now compare the RTTs of the VGFEs with the RTTs of the GFEs in all 438,000 traceroutes. We compute the ratio of VGFE RTT to GFE RTT for each traceroute. The closer this ratio is 1, the closer the VGFE is to the GFE. The CDF of the VGFE RTT to GFE RTT ratio is shown in Figure 5.3. Interestingly, this ratio is greater than 1 most of the time, with median 1.16 and mean 1.43. Again, using the median to eliminate impact of outliers, we conclude that the RTTs of VGFEs and GFEs are typically within 20% of each other. While this seems like a high percentage, the average RTT of a GFE is around 31 ms, which means that the RTTs of a VGFE and a GFE will be typically within 5 ms of each other. We note this as the approximate error margin of RTT when using VGFEs.

It is also interesting to note that the VGFE RTT is almost always higher than the GFE RTT. This can be attributed to the fact that the GFEs are likely well-connected from anywhere in the world. The forward or the reverse path of a GFE is hence less congested than the corresponding path for a VGFE. While this means that we will be underestimating the connectivity of a GFE if we instead use VGFE in place of it,

geolocation of the Google data centers is worthwhile. We will keep the error margin in RTT mentioned above in mind when using VGFEs.

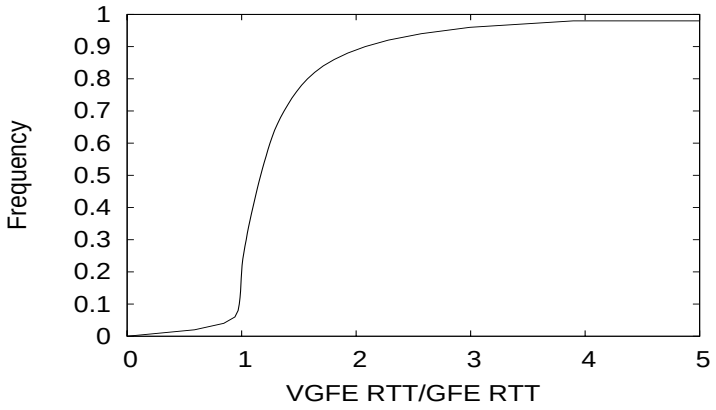


Fig. 5.3. CDF of the ratio VGFE RTT/GFE RTT

Having geolocated the VGFEs, we leverage similar techniques to geolocate the Google public DNS servers. We run traceroutes to the Google Public DNS IP 8.8.8.8 from various PlanetLab nodes over a period of a couple of days resulting in about 245,000 traceroutes. We again use the prefixes originated by Google's network to determine the hops inside Google's network in the traceroutes to the public DNS IP. Let h_{tDNS} be the total number of hops in the traceroute to DNS IP and h_{bgDNS} be the number of hops before entering Google's network. We compute the *Pre-Google DNS Hop Ratio* $\frac{h_{bgDNS}}{h_{tDNS}}$. The CDF of this ratio is about the same as Figure 5.2(a) with both the median and the mean of the Pre-Google DNS Hop Ratio being around $2/3^{rd}$. Similar to the traceroutes to Google IPs, we also compute the Pre-Google DNS RTT Ratio which is the ratio of the RTTs of the hop right before Google's network rtt_{bgDNS} and the last hop (Google DNS) rtt_{tDNS} of the traceroute. The median RTT ratio is about 0.63 which is around $2/3^{rd}$ just like the hop ratio, whereas the mean RTT ratio is 1.11, affected by outliers where the RTT to the hop before Google's network is high due to queuing delays and other network conditions.

We also define VGDNS, which is the *Virtual Google DNS IP*, as the last hop right before the Google DNS IP in the traceroutes. We collect all such VGDNS IPs across the traceroutes from PlanetLab nodes. We first use these VGDNS nodes to geolocate the Google data centers using techniques of Section 5.3 similar to those we used for VGFEs. We obtain 571 unique IP addresses for VGDNS which respond to pings. They are then geolocated using MaxMind [203] and we find 142 unique locations for those IP addresses. We use the same hierarchical clustering algorithm to cluster these IP addresses within a threshold of 50 miles. This leads to 98 unique clusters for the 142 IP addresses. If we consider the 67 unique locations located in US, they can be grouped into 37 clusters.

We now compare the RTTs of the VDNSs with the RTTs of the GDNSs in all 245,000 traceroutes, and compute the ratio of VGDNS RTT to GDNS RTT for each traceroute. The CDF of this ratio is depicted in Figure 5.4. This ratio is greater than 1 most of the time, with median 1.19 and mean 1.43. These figures are very similar to those obtained for VGFEs. Again, using the median to eliminate impact of outliers, we conclude that the RTTs of VGDNSs and GDNSs are typically within 20% of each other. While this seems like a high percentage, the average RTT of the DNS node is around 12 ms, which indicates that the RTTs of a VGDNS and DNS will be typically within 3 ms of each other. We note this as the approximate error margin of RTT when using VGDNS nodes. Since the RTTs of VGDNS nodes versus DNS nodes are closer, compared to VGFE and GFE nodes, we consider the VGDNS nodes to represent the Google data centers more closely.

5.4.3 Methodology

We now seek to study the performance of Google public DNS with the aim of comparing it with native DNS. For this purpose, we look up 14 web sites (listed in Table 5.2) using both native DNS and Google public DNS (IP 8.8.8.8) at periodic intervals. The intervals are chosen based on the TTL values returned using the "dig"

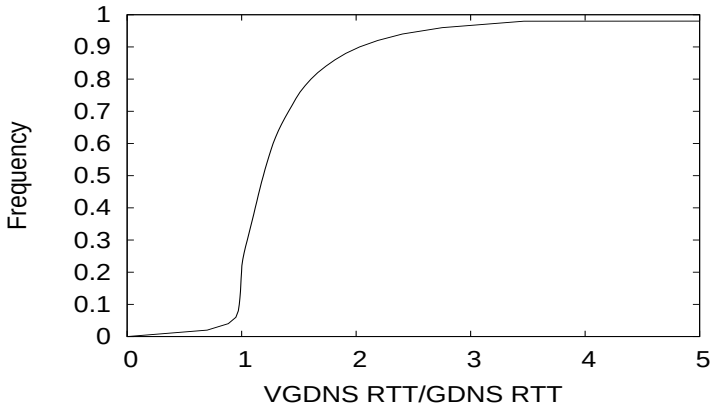


Fig. 5.4. CDF of the ratio VGDNS RTT/GDNS RTT

command. Since the server returned would be the same if probing was repeated during the TTL period, we set our interval to be the TTL value returned during the first iteration. If the TTL value returned using the native DNS and the public DNS are different, we select the higher of the two values as the probing interval. This ensures that both the public DNS and native DNS entries returned during each interval are not the cached ones. The experiments are repeated over a period of two days, with the iterations performed depending on the probing interval.

The 14 web sites we lookup exhibit different levels of popularity, e.g., www.google.com and www.facebook.com are highly popular whereas other web sites like www.cs.purdue.edu (the Computer Science Department at Purdue University), www.item.ntnu.no, and www.lowfatlinux.com are much less popular. One of the features of popular websites is that their TTL values are low so lookups are more frequent.

In the subsequent sections, we study three features of Google public DNS, namely, caching, DNS query time, and lookup results in terms of RTT to the server returned.

Table 5.2
Web sites resolved for comparing Google public DNS and native DNS

Web Site Number	Web Site
1	www.bing.com
2	www.google.com
3	www.ask.com
4	www.youtube.com
5	www.yahoo.com
6	www.facebook.com
7	www.search.yahoo.com
8	www.craigslist.org
9	www.caida.org
10	www.cs.purdue.edu
11	www.ripe.net
12	www.item.ntnu.no
13	www.ieee-infocom.org
14	www.lowfatlinux.com

5.4.4 DNS Caching

We record the TTL values for the web sites in Table 5.2, as reported by looking up each site from both Google public DNS (TTL_{gDNS}) and native DNS of the PlanetLab node (TTL_{nDNS}). The higher the TTL value, the longer the result will be cached. We compute the difference $TTL_{gDNS} - TTL_{nDNS}$ whose mean and median values are shown for each of the sites in Table 5.3 along with the mean values of TTL_{nDNS}.

The results demonstrate that for popular web sites, Google's public DNS and native DNS have about the same TTL values, indicating similar levels of caching. However, two popular web sites, namely www.google.com and www.youtube.com stand out since for them, the TTL values from Google public DNS (about a minute or so)

are higher than the native DNS values. This can be explained by the fact that since the servers to be returned are controlled by Google, the best servers for a reasonable TTL value can be returned easily. For relatively unpopular web sites, Google TTL values expire much sooner than those of native DNS. For instance, for www.ripe.net, the Google DNS caches for about 10 hours versus 28 hours on the average for native DNS. Google is using prefetching [216], where Google DNS prefetches name resolutions independently of whether users ask for them. This can explain why the TTL values are expected to be lower since during the probing interval, Google DNS could have prefetched the records, reducing the TTL values. Another possible reason for the lower TTL values can be that less popular web sites are accessed relatively frequently through Google DNS because of Google's popularity. This could lead to resolution of names during the probing period. However, we believe that prefetching is the most likely cause of lower TTL values. Overall, Google public DNS performs less caching than native DNS for less popular web sites, prefetching name resolutions, whereas for popular web sites, its caching behavior is about the same as native DNS.

5.4.5 DNS Query Resolution Time

We now study how the DNS query performance of Google public DNS compares with that of native DNS. For each of the DNS queries performed over the course of two days, we record the DNS query response time of the web sites through Google public DNS $QTime_{gDNS}$ and that through native DNS $QTime_{nDNS}$. We compute the percentage difference of the public DNS query time from native DNS query time according to the equation:

$$\% \textit{ Difference in Query Times} = \frac{(QTime_{gDNS} - QTime_{nDNS}) \times 100}{QTime_{nDNS}}.$$

The statistics of this percentage difference for the 14 web sites are listed in Table 5.4. We observe that, except for few web sites, the query time of Google DNS is quite high compared to that of native DNS even for the median case. There are cases

Table 5.3
Statistics of diff=$TTL_{gDNS} - TTL_{nDNS}$

Web Site	diff Mean (s)	diff Median (s)	TTL_{nDNS} Mean (s)
www.bing.com	6.16	-4	15.00
www.google.com	62.16	59	178.89
www.ask.com	5.88	-4	15.56
www.youtube.com	52.86	42	196.56
www.yahoo.com	0.75	-7	41.25
www.facebook.com	4.53	4	70.05
www.search.yahoo.com	3.99	12	173.82
www.craigslist.org	-25.49	-35	214.68
www.caida.org	-125.86	-112	495.39
www.cs.purdue.edu	-8030.39	-3965	57420
www.ripe.net	-38418.9	-24387	104231
www.item.ntnu.no	-10773.3	-1141.5	75411.9
www.ieee-infocom.org	-516.05	-226.5	3133.56
www.lowfatlinux.com	-4096.34	-8075.5	65508.7

when Google public DNS performs better but on the average, native DNS outperforms Google public DNS. This can be a surprising result given Google's resources. However, this can be attributed to Google's DNS service being subjected to the problems of interdomain routing and delays due to queuing in the Internet. Clearly, from this perspective, having a local DNS is preferable.

5.4.6 DNS Lookup Results

For every DNS lookup conducted through Google public DNS and native DNS during our experiments, we record the RTT to the server returned as a result of the

Table 5.4
Statistics of % diff=$\frac{(QTime_{gDNS}-QTime_{nDNS})\times 100}{QTime_{nDNS}}$

Web site	% diff Mean (%)	% diff Median (%)
www.bing.com	1206.54	250
www.google.com	2793.82	800
www.ask.com	1121.2	200
www.youtube.com	2117.76	300
www.yahoo.com	1133.74	40
www.facebook.com	1364.29	252.17
www.search.yahoo.com	1151.4	50
www.craigslist.org	914.82	-58.57
www.caida.org	819.02	-80.27
www.cs.purdue.edu	1164.13	22.41
www.ripe.net	838.77	-3.59
www.item.ntnu.no	899.27	-80.47
www.ieee-infocom.org	843.81	-66.67
www.lowfatlinux.com	1137.06	-76.36

lookup. The RTT is recorded by averaging the RTTs of 10 ping packets. For each iteration, we record RTT_{gDNS}, which is the RTT to the server returned by resolving the web site through Google public DNS, and RTT_{nDNS}, the corresponding value through native DNS.

Although we trigger the ping probes immediately following the DNS lookup, the RTTs to the servers may change based on queuing delays and network conditions. Hence, we consider two cases, one where the servers returned by the two DNS services are the same, and the second where they are different. For each of the cases, we

compute the value of $RTT_{gDNS} - RTT_{nDNS}$, the statistics of which are shown in Table 5.5.

Table 5.5
Statistics of difference between RTT_{gDNS} and RTT_{nDNS} for same and different servers

Web Site (www.*)	Mean Same Server (ms)	Median Same Server (ms)	Mean Different Server (ms)	Median Different Server (ms)
bing.com	-11.38	0	25.02	15.11
google.com	-2.19	0	-20.97	-0.1
ask.com	-9.08	0	22.44	16.76
youtube.com	-1.93	0	-16.49	-0.069
yahoo.com	-1.81	0.002	16.02	0.305
facebook.com	-6.7	0	29.82	0.896
search.yahoo.com	-0.55	0	20.81	-8
craigslist.org	0.13	0	-27.77	-54
caida.org	0.092	0	-7.7e+06	-365
cs.purdue.edu	-0.095	-0.003	-4.51e+06	-55
ripe.net	1.23	-0.003	-28888.3	-56
item.ntnu.no	-0.48	0	-8.03e+06	-404.5
ieee-infocom.org	-0.17	0	-2338.44	-3385
lowfatlinux.com	1.03	-0.001	-1.43e+07	-57236

The results indicate that the RTTs to the same server are almost the same. This validates our methodology as despite that pings are triggered at slightly different times, the RTT values are not severely affected. When we compare the RTTs to different servers being returned by Google and native DNS, we find that for *unpopular* web sites, the servers returned by Google have extremely *high* performance. This can be attributed to the prefetching strategy of Google, whereby the best servers can be

picked before a resolution request comes in. For the Google and YouTube servers, the servers returned by Google public DNS are moderately better in latency than those given by native DNS.

For other popular web sites, the servers given by Google DNS fare worse in RTT than their native DNS counterparts. For example, for Google's search competitors www.bing.com and www.ask.com, the servers returned are worse in latency for both the median and mean cases by 15-25 ms. Since the RTT of servers returned for these popular sites is itself of that order of magnitude, the RTTs can sometimes be significantly higher. This also applies to other highly popular sites such as www.facebook.com. This is because popular web sites have their own extensive data center networks or clouds, and the optimal server chosen by the closest Google DNS site to a client may not be the optimal one chosen at the client itself. We discuss this poor performance of content retrieval of distributed websites while using Google DNS and ways to improve client performance in Section 5.5.

5.4.7 Redirection Performance of Google Public DNS

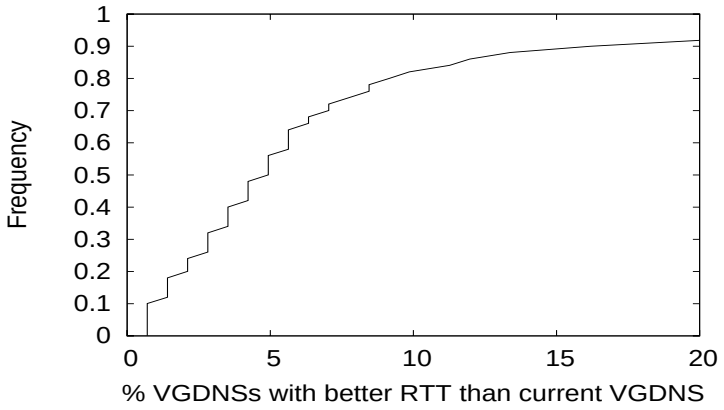
We now study whether a query to Google public DNS is indeed redirected to the "closest" data center location as claimed in the performance benefits of using the service. Our definition of closeness is in terms of latency since this is the main factor in the performance of the DNS application. As pointed out above in Section 5.4.5, a high query time is likely due to the time taken to reach the data center location.

To investigate this redirection performance, we use VGDNS IP addresses from Section 5.4.2 as an indication of the locations of public DNS servers. Even though we had clustered them earlier, we now use all the 142 VGDNS nodes as candidates for redirection to Google DNS. This is because while clustering helps us understand the geographical location of data centers, closeness in geographical distance does not necessarily mean closeness in network latency, which is the primary criteria for determining query performance.

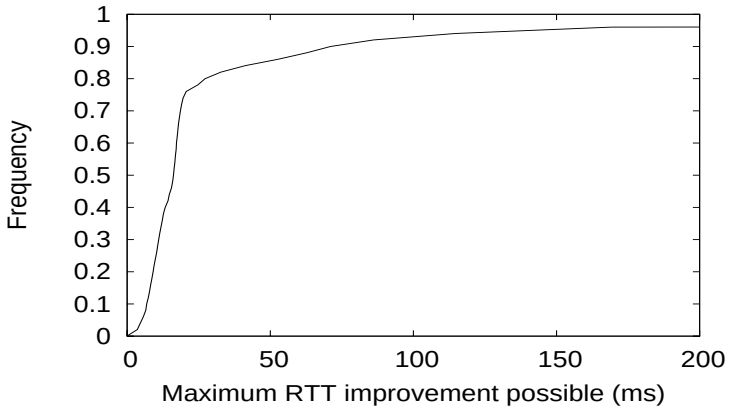
For a more in-depth investigation, we focus in this section on a subset of 20 PlanetLab nodes, well spread out geographically, with nodes from the US, Australia, Poland, Korea, Finland, Netherlands, Spain, Israel, and Denmark. For each of those nodes, in each iteration, we perform the following experiment. We measure the RTT (using ping) to each of the 142 VGDNS nodes and compare it to the RTT of the VGDNS node $VGDNS_{curr}$ obtained by finding the last hop in the traceroute to the Google public DNS IP. We compute, in each iteration, the number of VGDNS nodes with lower RTT than $VGDNS_{curr}$, and if there are any, the RTT difference between $VGDNS_{curr}$ and the VGDNS node with the lowest RTT. This estimates the maximum RTT savings that could be obtained by redirecting the query to the public DNS IP to a different location, indicating the best possible performance. It is important to note that this potential RTT saving is observed via measurements, so utilizing another VGDNS would have improved performance.

A total of 2200 iterations are completed across the twenty PlanetLab nodes. Out of these, for only 20.8% iterations, none of the 142 VGDNS nodes have a shorter RTT than the current VGDNS node $VGDNS_{curr}$. This seems to point to other factors being used to route to a Google public DNS node, since about 80% of the time, we can find a lower latency VGDNS node than the one returned. The CDF of the percentage of the 142 IPs which have lower RTT values in any given iteration is shown in Figure 5.5(a). The mean percentage is 7.8% and the median value is 4.9%, which implies that when VGDNS nodes with lower RTT are available, there are about 8% of the total VGDNS nodes available which are closer on average. While these numbers are low, even in the median case, there are about 7 VGDNS nodes which have lower latency than $VGDNS_{curr}$.

Of course, we need to consider the *magnitude* of the reduced latency when lower latency VGDNS nodes exist. A miniscule improvement, e.g., a couple of milliseconds, can easily be attributed to measurement errors and may not necessarily reflect a better available data center. The maximum possible reduction in RTT can be obtained by using the alternate VGDNS node with the lowest RTT of the 142 possible



(a) CDF of % of VGDNS node with lower RTT than current VGDNS



(b) CDF of maximum RTT improvement possible over current VGDNS by querying an alternate VGDNS

Fig. 5.5. CDFs of percentage of closer VGDNS nodes and the latency improvement possible by moving to the closest VGDNS node

VGDNS nodes. The CDF of this maximum improvement in RTT is plotted in Figure 5.5(b). The average improvement is about 135 ms, which is affected by some high improvement values. However, even the median improvement is 16 ms, which is non-negligible, given that the latency of the VGDNS node can differ from the latency of the DNS node by an average of 3 ms (Section 5.4.2).

We examine a few cases of improvement in RTTs to see if the geographical location of the VGDNS nodes is consistent with the RTT improvement offered. One of the PlanetLab nodes 147-179.surfsnel.dsl.internl.net located in Utrecht, Netherlands, reached the VGDNS node with IP address 209.85.255.126, which according to MaxMind is located in Mountain View, CA, with an RTT value of 62 ms. However, our list of VGDNS nodes includes a VGDNS node with IP address 162.97.116.65, located in Rochester, NY, which would have resulted in an RTT of only 38.6 ms, resulting in a 38% improvement in RTT. Geographically speaking, instead of being redirected to a data center in NY 3690 miles away, it was redirected to one which is 5484 miles away in CA. Using the observation that expected RTT through the Internet is that obtained when bits travel at $\frac{4}{9}^{th}$ the speed of light in vacuum [217], a distance of 1794 miles should result in savings of 21.7 ms, which is consistent with our result of a saving of 23.4 ms. This result suggests that latency-based redirection of clients may improve Google DNS lookup performance.

This high latency redirection performance is consistent with the findings of Google researchers in 2009 [66]. They study a currently running system on Google's network which diagnoses causes of high latencies, some of which are reported to be fixed [66]. Our results show that there is still room for improvement in client redirection in the Google public DNS service.

5.4.8 Performance of Google Search

We now study the performance of Google search – the most important Google application and the number one search engine in the world. For each of the 20 PlanetLab nodes considered in Section 5.4.7, we look up www.google.com through both native and Google public DNS, and find the RTT to these two GFEs returned through ping. We take the minimum of these two latencies to be $CurrRTT$, the latency of www.google.com. About 52% of the time, the GFE RTT from the public DNS server is longer than the GFE RTT from the native DNS server. We also find

the latency to the 232 GFEs recorded earlier in Section 5.4.1 and record if any of the GFEs have lower latency than $CurrRTT$. We find that for only about 15.3% of the iterations, none of the 232 GFEs have lower latency than the one returned from the lookup of www.google.com. However, when we compute the mean latency by which one of the GFEs is lower, we find that the median RTT improvement possible is only 3.7 ms while the mean RTT improvement possible is about 41.7 ms. The CDF of the RTT improvement is depicted in Figure 5.6.

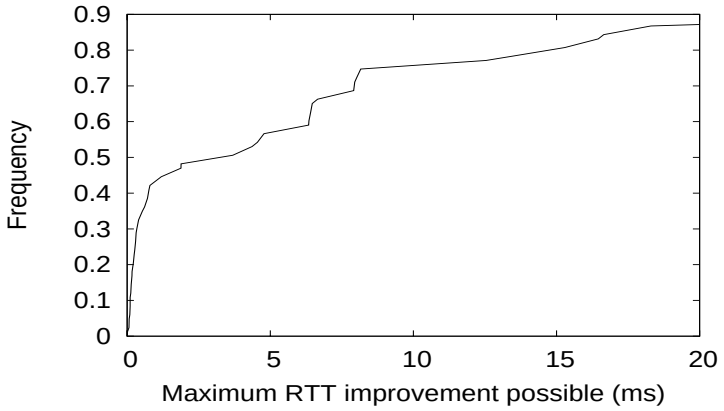


Fig. 5.6. CDF of maximum improvement possible using another GFE rather than the one returned

We also performed Google searches on a few popular search terms, such as *facebook*, *google*, and *youtube* (their popularity was determined by Google Insights [218]) on each of the GFEs, and recorded the entire packet exchange using `tcpdump`. The search result was parsed to yield the *Google Search Time* as reported by the search engine.

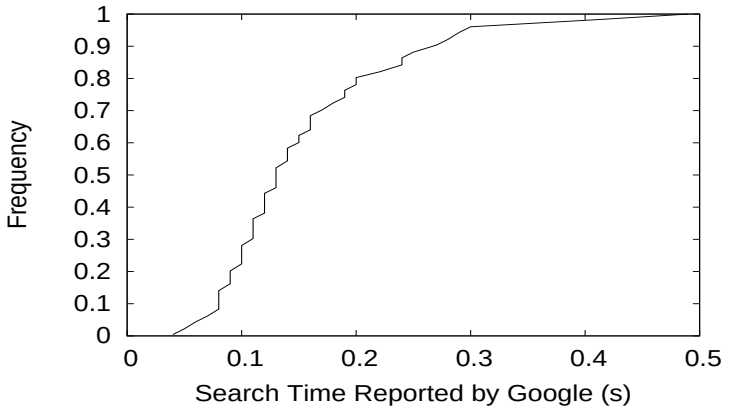
Using the `tcpdump` of the TCP connection exchange between the client and the google.com server, we compute the RTT of the communication as the time elapsed between the client sending the SYN and receiving the SYN-ACK. Since SYN packets are small, transmission delays are negligible. We only assume that the server responds immediately to the SYN. We also record the *Google Response Time* as the time

between the client sending the ACK to the SYN-ACK to the time when the GFE sends a FIN packet, terminating the search session. For each of the 232 GFEs, we record these three quantities: (i) Google search time, (ii) RTT, and (iii) Google Response Time, and we compare the GFEs returned by resolving www.google.com through native DNS with the public DNS case. A typical set of results for the search term "facebook" for all GFEs is shown in Figures 5.7 and 5.8 along with the metrics of the GFE redirected to through Google search.

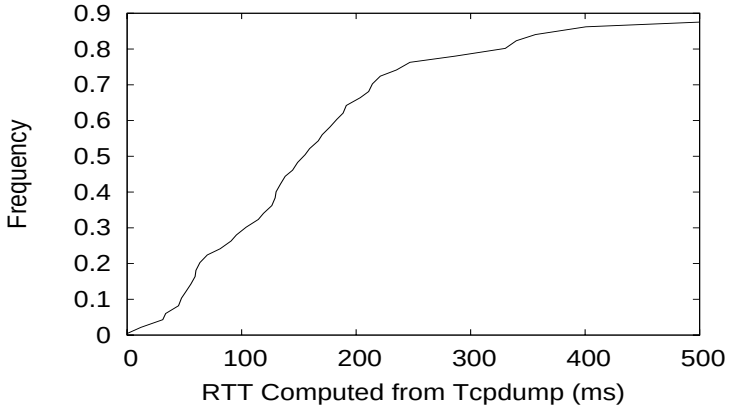
The results illustrate that the distribution of search statistics is quite diverse. Through both native and public DNS lookups, the performance of the resolved GFE exceeds that of the median GFE, showing that the GFE to which the client is redirected to is better than in the median case. The GFE given by the native DNS performs better in terms of RTT and overall Google response time. In contrast, the search time reported by public DNS-resolved GFEs is usually lower than the native DNS-resolved GFE. These results as well as our earlier results suggest that Google public DNS uses server load as a more important criterion for GFE selection, compared to latency between client and server. This is easily explained by the fact that the server load is known to Google and this information can be exploited by its DNS for better search performance. This comes at the expense of RTT and Google response time, due to unpredictable network conditions between the client and the Google DNS server. This result points to a potential avenue for research on incorporating network latency when selecting data centers or front ends.

5.5 Content Retrieval using Cloud-based DNS

Having obtained an insight into Google DNS through the measurement study in last section, we now focus on content retrieval by a client using cloud-based DNS with Google DNS as an example. As pointed out in the previous section, performance degradation with cloud-based DNS has been observed, due to lack of proximity between the server returned and the client. For sites which only have a few co-located



(a) CDF of Google search time as reported by all GFEs, Median=0.13 s; GFE Native DNS search time = 0.22 s ; GFE public DNS search time = 0.12 s



(b) CDF of RTT between client and all GFEs, Median=155 ms; GFE Native DNS RTT=11.98ms; GFE public DNS RTT=55.23ms

Fig. 5.7. CDFs of Google search times and RTT

DNS and content servers, e.g., Purdue University, proximity of the returned content server to the client is not a concern. Even for websites which have tens of data centers comparable in number to cloud-based DNS systems like Microsoft [132, 139], the clients should receive good performance using cloud DNS. However, for a highly dis-

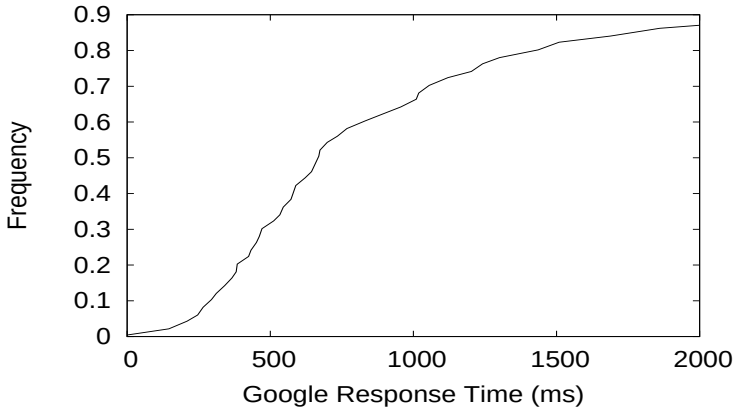


Fig. 5.8. CDF of Google Response Time as reported through all GFEs, Median=669 ms; GFE Native DNS Response Time=267ms; GFE public DNS Response Time=465.49ms

tributed CDN like Akamai [142], this lack of proximity-awareness becomes a severe problem. Huang *et al.* [139] estimate that the server latency increases by as much as 193 ms at the 95^{th} percentile when using cloud-based DNS systems, compared to local DNS. This is unacceptable especially since Akamai's network is used for dynamic content that is fetched over a long period of time, e.g., video streaming.

Akamai is the dominant content provider, delivering between fifteen and thirty percent of all Web traffic, reaching more than 4 Terabits per second [219]. It is used in streaming popular content such as the recent UK royal wedding, and has experienced peak loads of up to 10 million views per minute [220]. This makes the problem of poor Akamai content servers returned by using cloud-based DNS systems an important one, which we study in this section.

We use our technique of geolocating cloud data centers from Section 5.3 and find that Google DNS servers are placed much sparsely around the world than Akamai's servers yielding poor perceived client performance while accessing Akamai's content using Google DNS. We then present and compare alternative solutions to handle this problem. We posit that cooperation among cloud providers, those which host content

and those which host DNS services, is the best solution to this problem. However, in the absence of such cooperation, clients need to be smarter when accessing CDN-based content using cloud-based DNS systems. We present the design of a hybrid client-cloud approach which uses our understanding of Akamai's DNS network to query specific nameservers whose identity (IP address) has been found using cloud-based DNS. We find that the servers returned by this hybrid approach are usually the same as those returned by local DNS, preserving the performance advantage of local DNS. Our results also shed light onto Akamai's network, demonstrating that Akamai's DNS servers do respond to queries even when asked out of turn, albeit after a potential delay caused by possible information sharing across DNS servers.

5.5.1 Preliminary Measurements

We begin with a preliminary study of Akamai and Google DNS infrastructure. Section 5.2 describes the operation of Akamai. Its CDN hosts different types of content with various CNAMEs, *e.g. a1507.b.akamai.net* (Figure 5.1). We use the eleven Akamai CNAME patterns from Section 4.6.2, which are listed in Table 5.6, where channel numbers 0 to 4094 within a pattern lead to valid CNAMEs. It is known from previous work that channels share edge servers for load balancing purposes [193]. We find that each of the 4095 channels within the same CNAME pattern map to edge servers IPs within the same Class C subnet or /24 prefix. Since there are at most 256 IPs in a Class C subnet, the average number of channels mapping to an edge server is about 16.

We now geolocate Google data centers by using our measurement technique involving traceroutes from Section 5.3. We run traceroutes to the Google Public DNS IP 8.8.8.8 from 575 PlanetLab [186] nodes spread all over the world. We define VGDNS, which is the *Virtual Google DNS IP*, as the last hop right before the Google DNS IP in the traceroutes. We verify that these IPs indeed belong to Google in a two-step process. We perform Whois lookups for Google which yield Autonomous

Table 5.6
Akamai CNAMEs studied in this section with their respective nameservers

CNAME pattern	Nameservers
x = 1 to 4094 , y = 0 to 8 for all rows unless specified otherwise	
a{x}.b.akamai.net	n{y}b.akamai.net
a{x}.c.akamai.net	n{y}c.akamai.net
a{x}.f.akamai.net	n{y}f.akamai.net
a{x}.h.akamai.net	n{y}h.akamai.net
a{x}.k.akamai.net	n{y}k.akamai.net
a{x}.l.akamai.net	n{y}l.akamai.net
a{x}.p.akamai.net	n{y}p.akamai.net
a{x}.vmg0.akastream.net	n{y}vmg0.akastream.net y = 0 to 6
a{x}.vmg2.akastream.net	n{y}vmg2.akastream.net y = 0 to 6
a{x}.uqg0.kamai.net	n{y}uqg0.kamai.net y = 0 to 6
a{x}.gi3.akamai.net	n{y}gi3.akamai.net

Systems (ASes) 15169, 36039, 36040, 36384, 36385 definitely belonging to Google (Section 5.4.1). We then collect routing tables from RouteViews [9], which contains routing tables containing the prefixes advertised by various ASes using the Border Gateway Protocol (BGP) and verify that all the VGDNS IPs are indeed advertised by Google ASes.

We collect all such VGDNS IPs across the traceroutes from PlanetLab nodes run for around 12 hours (1000 iterations), and obtain 1477 unique IP addresses for VGDNS. They are then geolocated using MaxMind [203] and we find 46 unique lo-

cations for those IP addresses. We use hierarchical clustering using 50 miles as the cutoff distance between clusters to obtain 40 clusters out of the 46 unique locations.

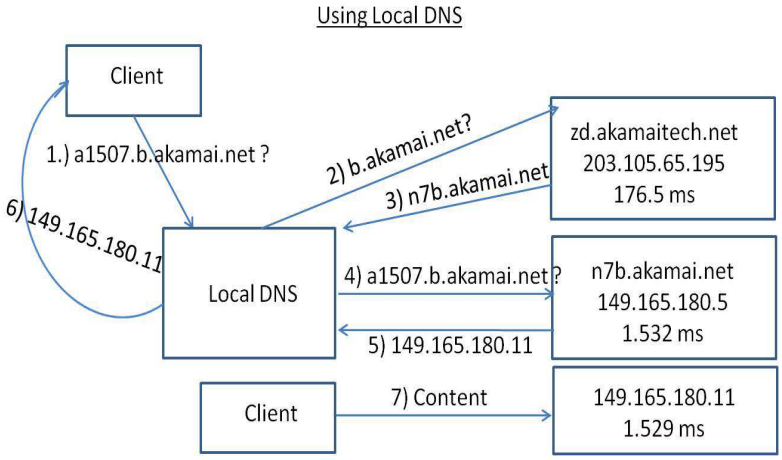
For locating Akamai data centers, we geolocate the content servers obtained by PlanetLab clients, as they resolve 11 random Akamai CNAMEs (one each from each row of Table 5.6) through local as well as cloud-based DNS (1000 iterations each). We aggregate the results across PlanetLab nodes and obtain 3223 unique IP addresses, which geolocate to 260 unique locations (latitudes and longitudes). Using the same clustering technique as in the previous paragraph, we obtain 123 clusters. While we by no means claim to discover all Akamai data centers, we point out that an experiment running for the same time from the same clients uncovers about *three times* as many Akamai data centers as Google data centers. This points to the more extensive presence of Akamai content servers, as opposed to Google data centers offering public DNS services.

5.5.2 Demonstrating the Problem

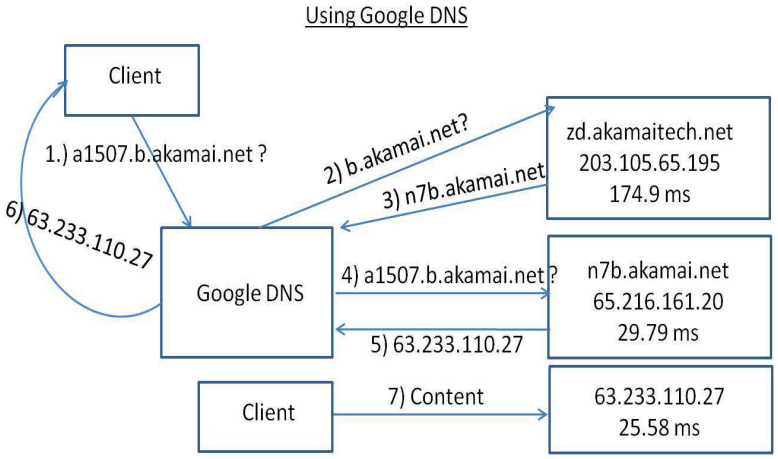
The problem we are investigating in this section is the high latency to the Akamai content servers, that a client is redirected to, when using cloud-based DNS systems. This problem stems from interactions between the cloud-based DNS systems and the Akamai DNS infrastructure. Akamai DNS returns the closest server to the querying node, which is the cloud-based DNS, and hence returns a server close to the DNS server and not necessarily the client [139, 155].

Figure 5.9 illustrates an example of the problem. We use the CNAME *a1507.b.akamai.net*, which is the CNAME of *videos.buy.com* (Section 5.2) and resolve it using local DNS and Google Public DNS. For clarity, we only show the resolution steps initiated by each of the DNS systems on behalf of the client involving Akamai nameservers. We choose a case where both resolutions seem to proceed exactly the same as far as the DNS server names are concerned. However, as Figure 5.9 shows, the actual server IP addresses and their latencies from the client are different, with the

Google DNS suffering because Akamai returns the IP addresses of the nameserver and content server which are close to the Google data center. This problem has been documented in [139, 155].



(a) Resolution through local DNS, indicating IPs and the RTTs from client



(b) Resolution through Google DNS, indicating IPs and the RTTs from client

Fig. 5.9. Comparison of DNS lookup of *a1507.b.akamai.net* through local DNS and Google Public DNS

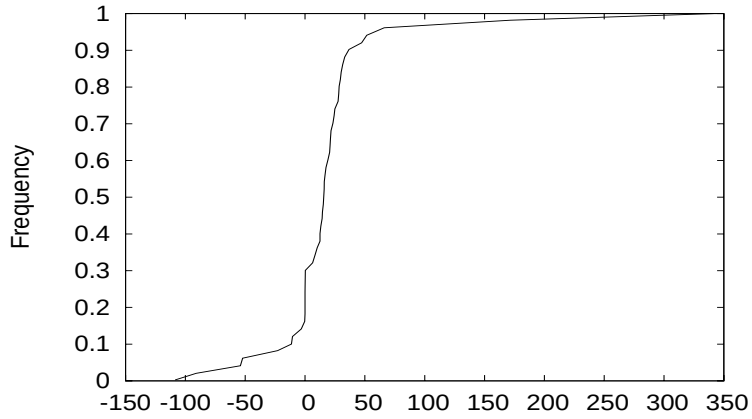
We now quantitatively demonstrate the existence of the high latency Akamai servers to the client when cloud-based DNS is used. We use 575 distinct PlanetLab

nodes spread over the world for our experiments. Each PlanetLab node serves as a client interested in obtaining Akamai content using Google DNS. We are interested in obtaining the Akamai-hosted content in each of the eleven CNAME patterns in Table 5.6 from as many content servers as possible. Since each of the 4095 CNAMEs within a pattern map to 256 content servers within the same /24 prefix, we randomly select n CNAMEs from each pattern such that we expect to see all 256 edge servers, with n to be determined. This problem is equivalent to selection of white balls from a box full of white balls one at a time, painting them red and putting it back in the box. In this case, n would be the expected number of draws required to see all the balls in the box in which case they would all be red. This problem has been solved in [221] and, using their result in our context, we find that we need $n = 1568$. Adding in the cases with known CNAMEs, (e.g. *a1507.b.akamai.net* for *videos.buy.com*), we obtain 1571 CNAMEs per pattern of the form $a\{x\}.\{z\}.akamai.net$ or $a\{x\}.\{z\}.akastream.net$ or $a\{x\}.\{z\}.kamai.net$ with x as the random number between 0 and 4094 and z chosen as appropriate from Table 5.6. We use these 1571 CNAMEs per pattern in all further experiments.

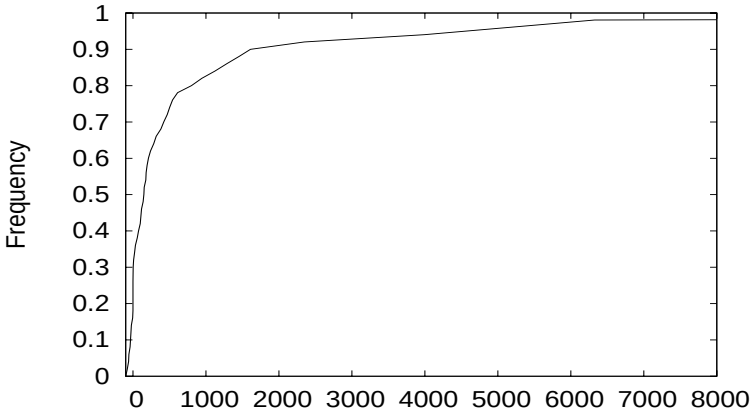
In the first set of experiments, we probe the CNAMEs using the local DNS of each PlanetLab node and then using Google DNS. We measure the quality of servers returned by pinging the servers with three ICMP echo request packets and noting the minimum RTT, which reduces RTT inflation due to network congestion to a certain extent. We use this technique for latency measurement throughout this work.

For each pattern of CNAMEs, we compute the mean difference in latency between the client and the server resolved through cloud-based DNS and that resolved through local DNS. We ignore the cases when the server returned by both DNS services is the same. We then average this mean latency inflation across all CNAME patterns and then across all nodes. Our results show that the average latency inflation is 14.15 ms for Google DNS, which is 720.5% in percentage terms. While the absolute latency inflation numbers do not seem extremely large, they are significant for video

streaming and dynamic content applications and the percentage inflation shows the poor cloud-based DNS performance.



(a) CDF of latency inflation when using Google DNS as observed by a client



(b) CDF of percentage latency inflation when using Google DNS as observed by a client

Fig. 5.10. Quantifying performance degradation using cloud-based DNS w.r.t. local DNS for CNAME $a\{x\}.c.akamai.net$

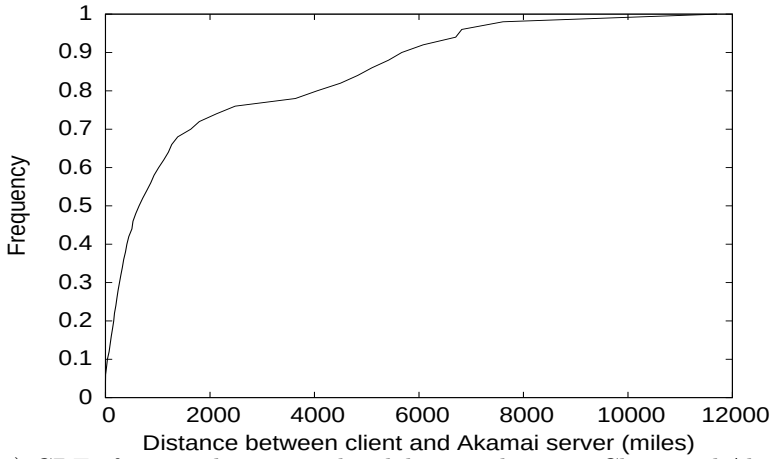
We also plot the CDF of latency and percentage latency inflation for a typical CNAME in Figure 5.10. The CDF is computed with one data point per PlanetLab node (client) and the plots for other CNAMEs are similar. One can see that there are a few cases for which the inflation is negative, i.e., the local DNS does give a server

which is farther from the client than that obtained by cloud-based DNS. However, such cases are infrequent and are likely caused by large distances between the client and local DNS [139]. The results also show that the latency inflation has a heavy tail. While the *average inflation is around 15 ms*, around 17% of the clients experience inflation of more than 1000%. The results are consistent with those given in [139] and stress the need to work around this problem.

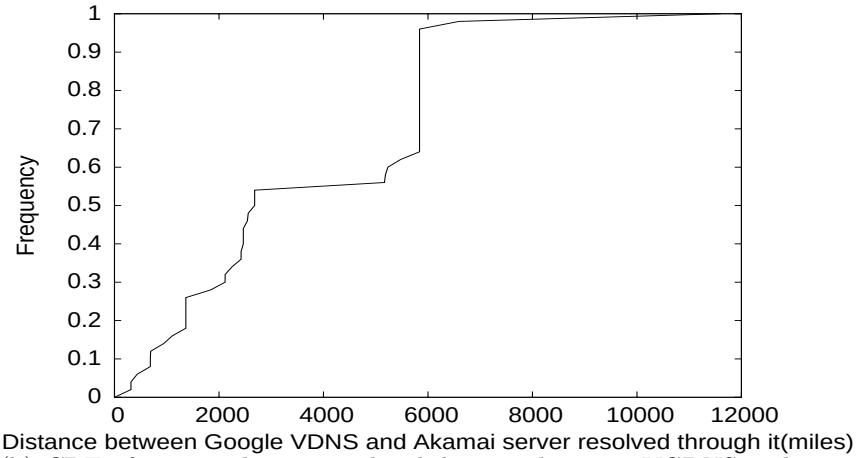
5.5.3 Causes

We now delve deeper into the results of the last subsection to understand the causes and characteristics of the latency inflation. For each of the 1000 iterations run from PlanetLab nodes, we record the node IP C, VGDNS IP G (Section 5.4.2), the Akamai server IP corresponding to CNAME *a1507.b.akamai.net*, obtained through local DNS (server A) and through Google DNS (server A'). We then geolocate these four IP addresses and compute the geographical distance between the client C and the Akamai server it is redirected to A, g_{C-A}. We also compute the distance between the VGDNS IP G and the Akamai server it is redirected to A', $g_{G-A'}$. We compute these distances and combine the results across iterations and across nodes. We find that the median g_{C-A} is 643 miles, whereas the median $g_{G-A'}$ is 2683 miles, substantially higher than g_{C-A}. We plot the CDF of these two distances in Figure 5.11.

We observe jumps at discrete distances in Figure 5.11(b), because of the small number of data center locations, which will cause some iterations to be grouped together. The plots show that Google DNS sees an Akamai server which is much farther away from it than a client seeing a corresponding Akamai server. The performance is particularly poor for a large percentage of clients which should see smaller distances (e.g., due to their presence in the US amid dense server infrastructure). Both plots in Figure 5.11 have a long tail, showing that some clients are indeed redirected to servers across the globe.



(a) CDF of g_{C-A}, the geographical distance between Client and Akamai server resolved through local DNS



(b) CDF of $g_{G-A'}$, the geographical distance between VGDNS and Akamai server resolved through Google DNS

Fig. 5.11. Comparing distances of Akamai content servers from the resolution node for client and Google DNS

We also compute, for each iteration, the percentage difference of $g_{G-A'}$ w.r.t. g_{C-A} and find that the median difference is 101%, which implies that $g_{G-A'}$ *is twice as much as* g_{C-A} in the median case. While some error can be introduced by our geolocation technique and VGDNS for representing Google data centers, the difference is still

substantial. This result is interesting assuming Akamai does not discriminate among clients, and shows that Google's DNS is substantially less effective in identifying a good Akamai server for itself than a client identifying a server for itself. Even if the client was colocated with the Google DNS server, it would still attain lower performance than an average Internet client. We contend that this is due to two reasons. First, Google performs prefetching of name resolutions [137], which does not work well for Akamai-hosted dynamic content. Akamai changes name resolutions in a matter of seconds [142] and the dynamic content precludes caching. Second, Google as a cloud is spread out over significant distances and may share its DNS resolutions among its data centers. As a result, it may not necessarily query Akamai's server from the DNS server which resolves client requests.

In our experiments, we compute g_{C-G}, which is the distance between the client and the VGDNS IP address. The median value of g_{C-G} computed across all iterations and all nodes is *5374 miles* and its CDF is depicted in Figure 5.12. We compute the percentage difference of g_{C-G} w.r.t. g_{C-A} for each iteration and find this to be 88% in the median case, showing that *Akamai servers are usually located closer to the client than Google DNS servers*. This further indicates that Google's DNS presence is sparse in the world, as shown by results of Section 5.4.2 and [139]. Coupled with the sub-optimal Akamai servers seen by Google nodes, this leads to significantly poorer performance of clients in accessing Akamai content through Google DNS.

5.5.4 Solutions Overview

Having studied the poor client performance problem, we now explore the solution space of how a client can best use cloud-based DNS to access content hosted by Akamai and Akamai-like distributed content providers. We briefly compare the solutions presented in this work in Table 5.7. The subsequent sections explain these solutions in detail.

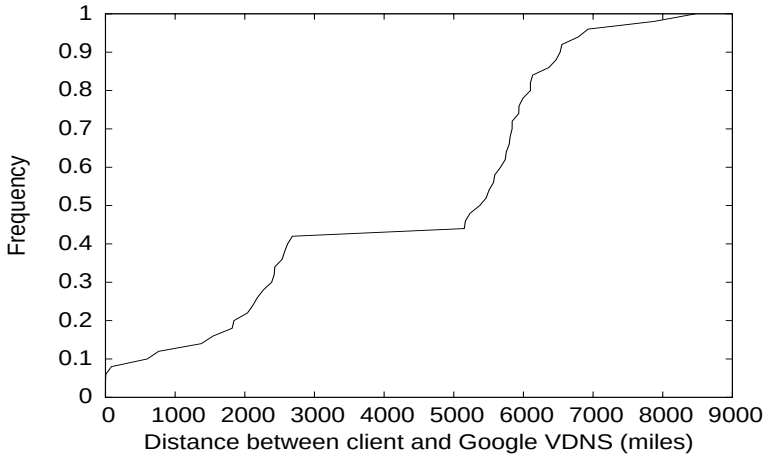


Fig. 5.12. CDF of g_{C-G}, the distance between client and VGDNS

5.5.5 Solution 1: Changes to DNS

A possible solution to the content location problem is based on a proposal initiated by Google researchers, which is currently an IETF draft [222]. This proposal requires changes to the generation of DNS requests and replies by allowing recursive DNS resolvers to expose the first three octets of the client IP address to a CDN network like Akamai [223]. Upon receipt of such a request, the CDN's DNS system *may* use that information to choose the server to be returned, which will be optimized for the client. While this scheme will certainly solve the problem, the primary drawback of this approach is that it requires changes to the DNS protocol which may not be universally adopted. One can certainly imagine some CDNs ignoring the option in the DNS request messages. As of this writing, it seems like some progress is being made on adoption of this proposal. Two cloud DNS providers, Google public DNS and Open DNS, along with CDNs of Bitgravity, Cloudflare, Comodo, CDNetworks and Edgecast have implemented this proposal [224]. However, the two biggest CDNs, Akamai and Limelight, have not participated yet and it is unclear if they will participate in the future.

Table 5.7
Solutions for obtaining good client performance when accessing Akamai-like content using cloud-based DNS

Solution	**Pros**	**Cons**
Changes to DNS by revealing client IP to Akamai	Correct Solution	Changes to DNS are expected to face adoption barriers
Cooperation among clouds	Best solution with varying degrees of cooperation possible	Agreements and trust setup
Increasing DNS centers	Some performance improvement expected	Infrastructure spending and no guarantee of improved performance
Hybrid client-cloud approach	Good resolved server performance	Requires client to potentially wait for resolution. The technique based on reverse-engineering Akamai is temporary as it depends on Akamai implementation.

5.5.6 Solution 2: Cooperation among Clouds

We posit that the best solution is to have cloud-based DNS providers like Google cooperate with CDNs like Akamai, similar to a peering arrangement between Autonomous Systems. It is in Akamai's interest to provide the best content servers for each client for higher revenue and customer loyalty, irrespective of the DNS provider that they use. Similarly, Google DNS wishes to deliver the best possible servers to the client for increased adoption of its DNS services. Such a solution is similar in fla-

vor to DONAR [225], where a content provider outsources its server replica selection mechanism to a third party provider. Various degrees of cooperation are possible, from where Google will have the responsibility of selecting an Akamai replica based on the client request (DNS server chooses replica, similar to DONAR [225]) to where Google DNS forwards requests for Akamai-hosted content to Akamai servers (content provider chooses replica, similar to [222]). The primary drawback of this technique is that it requires agreements between cloud providers, which may be difficult in the real world because of business reasons. Further, security issues may require trust between multiple organizations, which may be difficult to establish.

5.5.7 Solution 3: Increasing DNS Data Centers

Yet another solution can be for cloud-based DNS providers like Google to employ many satellite data centers [132], penetrating deep within ISPs so that its DNS servers become comparable in number to Akamai content servers. This implies that anycast routing for Google DNS will redirect a client to a closer DNS server which perhaps will see an Akamai server close enough to the client. However, this solution involves a significant investment from DNS providers like Google which they may not be eager to incur. Moreover, as our results from Section 5.5.3 indicate, sparse Google servers are only a part of the problem – Google returning farther Akamai servers than a normal client possibly because of prefetching still needs to be solved. In this scenario, cooperation among clouds as presented in the preceding paragraph would be a better solution.

5.5.8 Solution 4: Hybrid Approach

The solutions presented above are not deployed in today's Internet, and a client who wishes to access Akamai content today using cloud-based DNS will be directed to distant content servers. We now present a *hybrid client-cloud approach* that a client

can use to identify low-latency Akamai content servers while preserving the security and outsourcing benefits of cloud-based DNS.

In the hybrid approach, the client queries the Akamai second-level nameserver directly, which will cause a closeby content server to be returned. Of course, the client will need to know the IP address of the appropriate Akamai nameserver, and for that purpose it uses cloud-based DNS. Figure 5.13 shows the same example as Figure 5.9 but using this hybrid approach. The client queries Google DNS for obtaining the IP address of *n7b.akamai.net*. Once the IP is returned, the client queries the IP for the CNAME and obtains the content server. We also observe that the content server returned in this case is the *same* as that returned by local DNS in Figure 5.9(a). This is a hybrid approach because it involves the use of DNS services in the cloud to resolve the nameserver IP and a local approach to query the IP directly to obtain content servers.

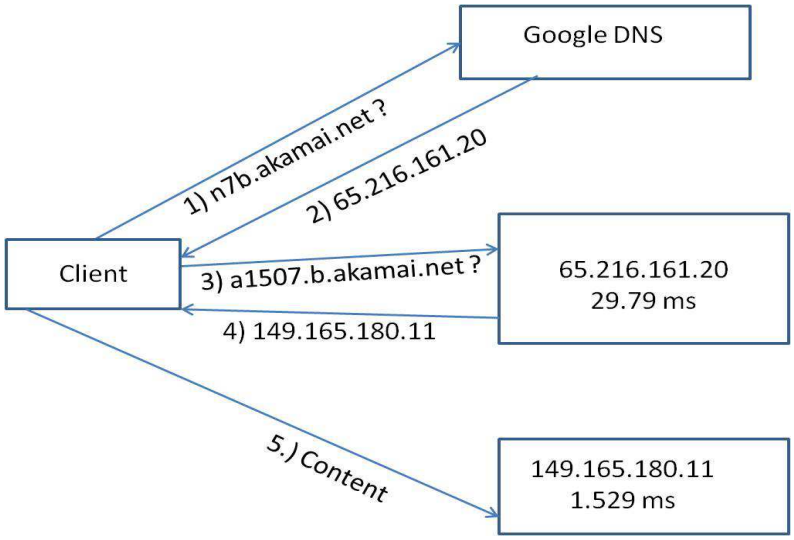


Fig. 5.13. Example of a hybrid approach for looking up Akamai content servers using Google DNS, showing IPs and the RTTs from client

A key aspect of this approach is that the client needs to know the name of the Akamai second-level nameserver, e.g., *n7b.akamai.net*. This can be built into the client-

side DNS software, since Akamai uses predictable nameserver names. For example, a CNAME of $a\{x\}.\{z\}.akamai.net$ will have the nameserver name $n\{y\}\{z\}.akamai.net$ with y ranging from 0 to 6 (Table 5.6). A nameserver with any value of y will work and one can even choose y randomly for load balancing purposes. CNAMES with different endings are handled similarly. An alternate way to find the name of the nameserver is through the authority section of a *dig* [201], or to do a *dig +trace* for the CNAME using cloud-based DNS as the default DNS (assuming the client wishes to take advantage of its security features). This reveals the name of the nameserver.

One of the key assumptions in this approach is that a client can successfully query an Akamai nameserver IP which has been provided to it by a potentially distant cloud-based DNS data center. From our experiments, we found that on querying an Akamai nameserver out of turn, it may or may not return a content server IP address. In case it does not, it returns a CNAME like *a1.b.akamai.net.0.1.cn.akamaitech.net* whose resolution causes the client to go through Akamai's DNS infrastructure and query a closeby nameserver. However, if the client retries the query after some time, it is usually successful and receives an IP address which is the *same* as the one it would have received had it queried using local DNS. This result indicates an important feature of Akamai's network. The content server returned to a client is dependent on the client's location and is *independent* of the Akamai nameserver queried. This is what makes this hybrid approach successful.

It is also important to note that there may be a slight delay before an arbitrary Akamai nameserver resolves a CNAME. This delay is most likely caused by background information sharing among various Akamai nameservers, presumably with the nameservers close to the client's location. We find that in nearly all cases, the delay is 15 seconds (which was the period of our retries), i.e., the content server is obtained sometime during that period. However, for CNAMEs in pattern $a\{x\}.k.akamai.net$, the resolution does not succeed even after 5 minutes, which is when we stop our retries. A few seconds delay in the most frequent case is an acceptable penalty since a typical client accessing Akamai dynamic content cares about the quality of the con-

tent (usually audio or video) and maintains a long-lived session for which a setup delay is acceptable.

We now conduct a measurement study to investigate the effectiveness of the hybrid approach. A content server is obtained using the approach and the latency to the server is measured and compared to the latency to the server returned using cloud-based DNS. We take the median latency difference to prevent interference from outliers, and average across all CNAME patterns and PlanetLab nodes. We find that the hybrid approach *reduces this median latency by around 7.5 ms*. If we consider the *mean* latency saving, we get 12.7 ms savings. These numbers are within 1 ms of the actual latency inflation caused by using cloud-based DNS as opposed to local DNS (Section 5.5.2). The numbers do not match exactly due to the variability in the ping latencies caused by variable network conditions.

We also find that the hybrid approach returns around 4-6% of the same content servers as those returned by using Google DNS. This may be because of the fact that there are cases for which a Google DNS data center is located close to the client, reducing the advantage of the hybrid approach. Nevertheless, in both the mean and the median case, our technique results in closeby servers being returned to the client.

We now compare the servers returned by the hybrid approach to those returned using local DNS. We find that they return the same server in 45.1% of the cases. This is expected since Akamai returns two content servers and we choose the first one as the content server returned. The second one is usually equally good and will randomly occur around 50% of the time, hence the exact similarity of content servers occurs in around 50% of the cases. To evaluate the cases where the servers are not the same, we measure the latency difference between the server returned by the hybrid technique and the local DNS and find that this difference is less than a hundredth of a millisecond on the average. This shows that the hybrid approach indeed performs per expectations, returning essentially the same servers as the local DNS systems. Hence, we have shown that the hybrid approach is indeed feasible and works well for most cases, with a delay of the order of a few seconds.

5.6 Chapter Summary

In this chapter, we have studied the evolved Domain Name System (DNS), a key component of the Internet infrastructure. This evolution is brought upon by the move to cloud computing, which necessitates DNS in the cloud. We have studied both DNS internal to a cloud, which is used to redirect clients to the best server, with the Akamai CDN as an example, and external DNS which provides DNS service to any Internet client, through a case study of Google DNS. We have developed a novel technique based on active measurements to geolocate cloud data centers, which cannot be readily located using commercial geolocation tools.

Our measurement study of Google's public DNS has shown that Google data centers cannot be easily geolocated via geolocation tools such as MaxMind. Using the hop before the last hop returned by traceroute as the geolocation technique yields more accurate results. Our study also reveals that Google DNS outperforms native DNS in terms of proximity of servers returned by resolving less popular web sites. Native DNS typically performs better in that respect for highly popular sites. Overall, Google DNS performs less caching. We find that lookups using the Google DNS service are not always directed to the closest server in terms of latency. The Google DNS servers to which clients are redirected to appear to be chosen based on their load as a primary criterion.

Our study on Akamai-hosted content retrieval by clients using cloud-based Google DNS shows that clients experience poor performance due to redirection to far off content servers. We have analyzed the reasons for this performance degradation and found that sparse placement of Google DNS servers along with prefetching are likely to blame for sub-optimal content servers returned by Google DNS. We have discussed several solutions to this problem, and posited that cooperation among clouds is the best solution. However, since no such solution is deployed today, we have presented a hybrid client-cloud approach that involves querying both cloud-based DNS systems and Akamai nameservers directly, thereby identifying content servers which are close

to the client. Our results present a marked improvement over the current performance of content servers returned by cloud-based DNS.

Our work raises important questions about the future cloud-based Internet, specifically the cooperation required among clouds and which services should be migrated into the cloud. We study these issues in Section 6.2, in our visions of the future and discuss research work to be done in this field.

6. CONCLUSIONS AND OPEN ISSUES

In this concluding chapter, we present the key results of our dissertation in Section 6.1, which validate our hypotheses of Section 1.5. Our visions of the future Internet along with its challenges are then discussed in Section 6.2. Finally, Section 6.3 presents future work.

6.1 Key Results

In this dissertation, we have studied Internet routing and Domain Name System (DNS), two key components of the Internet infrastructure. The Internet faces several challenges in ensuring high availability because of its heterogeneity which leads to interactions between various Autonomous Systems. Not only is Internet availability poor, predicting it is a difficult task because of various unpredictable factors affecting it (Section 1.3). Meanwhile, continuous Internet evolution is leading to rapid changes in Internet infrastructure, changing the fundamental structure of the Internet. In this dissertation, we have shed light on the predictability of the Internet infrastructure and how its evolution has affected its performance. The key results of our work are summarized below.

- We have provided insight on the predictability of Internet inter-domain routing in Chapters 3 and 4. Specificially, we have predicted long term availability of prefixes by observing its routing characteristics for a short period of time and using prediction models learnt from other Internet prefixes. We have studied and compared various prediction models and shown their applicability given various learning and prediction durations (Section 3.5). Our results have shown that one can tradeoff prediction performance and prediction duration, depending on

the prediction goal. Based on this work, we conclude that Internet availability is indeed predictable validating our hypothesis of Section 1.5.

- We have also identified metrics to compare prefixes in terms of their propensity to fail and develop a new prefix grouping named BGP molecules (Chapter 4). Different prefix attributes have been considered to identify which ones correlate with its failure tendency (Sections 4.3 and 4.4). We have found that AS paths to a prefix followed by its geographical location are good prefix failure indicators. We have used BGP molecules to predict prefix failures and the results show that a hybrid scheme achieves 91% predictability of failures with 99.3% coverage of prefixes in the Internet (Section 4.5). Again, this validates our hypothesis that while the causes of Internet prefix failures are unpredictable, it is indeed possible to predict the failures themselves. We have also developed a novel application of BGP molecules in improving the availability of Content Distribution Networks, without significant performance degradation (Section 4.6).

- We have studied the cloud DNS system caused by the advent of cloud computing in Chapter 5. Both internal DNS of a cloud and external DNS systems offered by cloud have been studied with case studies of Akamai (Section 5.2) and Google DNS (Section 5.4) respectively. We have investigated the interactions between these two types of DNS systems by quantifying the poor client performance while retrieving Akamai-hosted content through Google DNS (Section 5.5). The reason of this poor performance is found to be the disparity in the number of data centers operated by the two clouds, coupled with Google seeing sub-optimal Akamai servers, possibly due to prefetching and information sharing between data centers. This evaluation has used our active, lightweight technique for geolocating cloud data centers from Section 5.3. We have suggested various solutions to this problem of poor client performance, including a hybrid client-cloud approach which can be used in the current Internet to obtain Akamai

content servers using Google DNS, which are comparable in performance to those obtained by native DNS (Section 5.5).

6.2 Visions of the Future Internet and Open Issues

In this section, we extend our discussion of cloud computing from Section 2.4.2 by using our visions of the future to develop a model of the future Internet. We also present the open issues and challenges in the future Internet using insights from this dissertation.

6.2.1 Future Internet Model

In developing our Internet model, we assume that the current trends of cloud computing and Internet flattening [38] continue to the point where the Internet is composed of end-users and several clouds. We define a *cloud* as an organization in the Internet, which provides any Internet service. Different-sized clouds will exist in the future providing a wide variety of Internet services with some clouds providing the same service and hence competing with each other. The services provided by the cloud can include static and dynamic content delivery, other real-world applications such as e-commerce, access to software *e.g.* for document processing and photo editing and ability to host one's own content.

We envision clouds to provide access to hardware – not only virtual hard disks but also undersea optical fibers to carry data. "Startup clouds", which are newly formed, are expected to lease resources like data centers from clouds that provide these resources as a service. Hence, a cloud can *overlap* with existing clouds to varying extents until it becomes large enough to exist as a separate entity. Cloud overlap can also occur in the services realm, *i.e.,* a cloud can outsource some services to other clouds under appropriate business agreements. For example, Facebook uses Akamai's content delivery network for hosting pictures [226]. Akamai is a major CDN with global presence and its fault tolerance and performance properties make it an

attractive content host. In the future Internet, we expect content distribution to become a key service due to the insatiable appetite of users for content.

Apart from these services, we reckon that Internet access will also be a service provided by the cloud. Indeed, Google as a cloud is in the process of rolling out Internet service [197]. We expect other organizations to do the same or merge with existing ISPs. This is because Internet access service is bundled with DNS services and DNS enables a cloud provider to provide best access to other services it offers. DNS can use several inputs like the client's location, current network conditions inside the service provider's network and server load to redirect the client to the best possible server for all services provided by the cloud. The resulting better user experience will lead to increased revenue that can more than compensate for any investment in providing ISP services [125]. Other business reasons for a cloud providing its own Internet service can include decreased reliance on other networks to carry its traffic [125].

6.2.2 Will Multiple Clouds Co-Exist?

One of the most important questions that we need to answer about the future Internet is whether multiple clouds will exist. The current Internet traffic consolidation trend coupled with mergers and acquisitions (*e.g.,* Microsoft taking over Skype [227], Google taking over YouTube [228]), is leading to bigger and bigger clouds. Moreover, with Internet access being provided as a service, a cloud or very few clouds can dominate the future forming a monopoly or an oligopoly in the Internet.

However, we posit that multiple clouds are here to stay in the foreseeable future. This is due to the following reasons:

- Internet users representing the world population are fundamentally diverse in nature. Different websites are popular in different countries [229]. Even within the same country, users like to use multiple services on a daily basis, and even use multiple service providers for the *same* service. For example, Google is

the leader in search, yet other search providers like Microsoft and Yahoo have existed for many years with significant market share [230]. Similar trends are observed in other technologies like mobile phone usage, Internet browsers, and operating systems, and even extends beyond the Internet, for example into shopping.

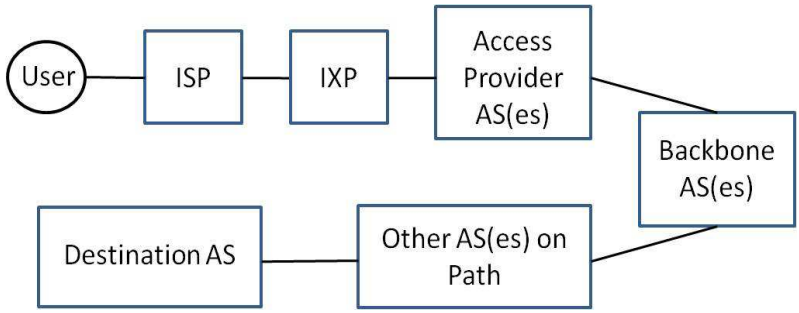
- Competition among technology players in the world fosters innovation and is an important driver for new and better technology. The innate nature of consumers is to use a variety of services – trying out services from competitors furthers multiple providers. For example, Facebook has been the leading social networking site since June 2008 [231] with an estimated 800 million users as of September 2011 [232]. However, Google Plus [233], a new social networking initiative by Google already has 20 million users within three weeks of its launch in a limited trial phase [234].

- Antitrust laws around the world prevent a technology company from monopolizing the market. For example, AT&T's antitrust suit led to its breakup [235], and Microsoft unbundled Internet Explorer from Windows as a result of a suit [236].

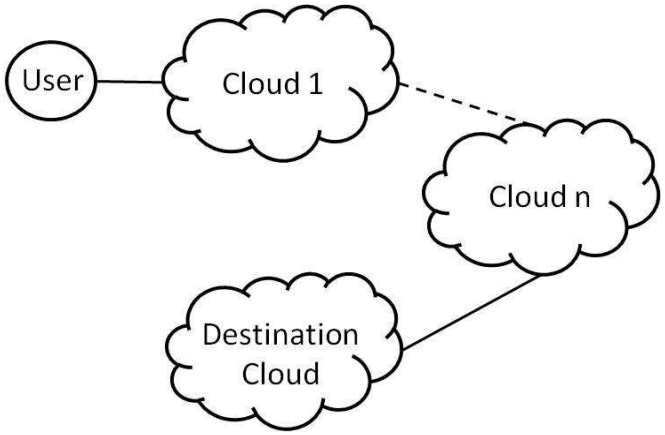
6.2.3 Evolution to a Cloud-Centric Internet

The current Internet requires a user to go through an ISP to the Internet backbone to reach a destination (Figure 6.1(a)) [38]. In our view, the future Internet will be composed of only clients and clouds as shown in Figure 6.1(b). The user uses Cloud 1 to connect to the Internet, and can then use services provided by other clouds.

One of the major changes is that ISPs will disappear and Internet service will be provided by clouds. This implies that traditional services provided by ISPs like DNS will have to morph into another form in the new Internet. We have studied DNS evolution in Chapter 5. The evolution of ISP services has already begun with cloud based DNS being provided by external DNS providers (Section 5.1). ISPs also provide content filtering which prevents malicious content from the Internet from reaching the



(a) The current Internet model



(b) The Internet cloud model

Fig. 6.1. Comparison of the current and future Internet models

client. We expect these services along with others like online backup to be provided as an Internet service suite to the client by the cloud. The evolution from the current Internet to the cloud-based Internet is expected to be smooth with mergers and acquisitions, and some users still continuing to use traditional ISP services. During this transitional period, backward compatibility will be implemented. For example, CloudIDs (Section 6.2.5) used in routing will be interpreted as AS numbers, and new versions of BGP can be incrementally deployed.

The key participant in the future Internet is the user who desires to use services from multiple clouds. The goal of a user is to get the best possible Quality of Service (QoS) for every service he/she uses. Storage is a good example of a service that a

user currently uses on a local machine. Storing an object in multiple clouds requires that the state of the object be kept consistent across clouds, and inter-cloud communication may be required to achieve this (Section 6.2.7). This may require significant traffic flow among clouds which may not be paying to transit traffic from each other (Section 6.2.5). It could lead to clouds passing storage costs to the consumer who pays low and ever-declining costs for the alternative of local storage hardware. However, the benefits of unlimited, easily accessible, reliable, and dynamically changing storage in the cloud cannot always be obtained by the user on his/her local machine. Hence, a clear tradeoff exists between the cost incurred for cloud storage and its benefits. We posit that this decision ultimately rests with each user and can go either way.

The future Internet should be user-centric and not cloud-centric, even though clouds will emerge as strong entities. While many services are provided by the cloud, outsourcing the services which a user can use on his/her own machine to the cloud may not always be efficient and cost-effective for the user.

6.2.4 Lightning among the Clouds

We now discuss the *lightning* or interactions among clouds in the future Internet. The most obvious interaction is competition among clouds providing the same service. Each cloud has to make its services as attractive as possible to the end-users – a challenging problem given diverse consumer preferences and business needs. This is certainly not a unique problem to the cloud-based Internet; it exists in today's Internet as well, for example between various providers of email services. The implications of this competition, however, are significant in the cloud context. Clouds are expected to make revenue from offering services to users at a high performance. This necessitates running a DNS service in each cloud to locate the best servers for each user and reaching a critical number of users to generate profit. The following sections delve deeper into various facets of cloud interactions.

6.2.5 Cloud Connectivity and Routing

We posit that the traditional AS relationships in the Internet [26] will converge to mostly peer-peer relationships among clouds with no customer-provider relationships. The situation is similar to richly connected Tier-1 ASes in today's Internet which are peers thereby carrying traffic for the other without any financial settlement. Peering agreements are usually established when both ASes can extract reciprocal benefits, *i.e.,* the benefit to any party in the agreement is not significantly greater than the benefit to the other party [237]. Usually that is true when both the ASes are of roughly similar geographic spread, number of customers, and incoming to outgoing traffic ratio [237].

In the cloud-based Internet, we posit that peer-peer relationships will exist even if the clouds are not of similar characteristics. This is because large clouds (equivalent to Tier-1 ASes) likely will provide Internet service and have to allow their users to connect to the entire Internet and use services of other clouds, especially for neutrality reasons (Section 6.2.6). Thus, they need to establish peering relationships with certain clouds. There can be a cloud A which does not provide Internet service, but that needs to establish peering with another cloud(s) B which provides Internet service, to reach Internet users. Cloud A can refuse to transit traffic from other clouds; however, that will not affect Internet connectivity. Hence, a cloud, whether it provides Internet service or not, *needs* to establish relationships with other clouds and this leads to peering relationships.

Addressing is needed for clouds to talk to each other, and it is likely that each cloud will be assigned a globally unique CloudId by IANA [43]. A cloud can have multiple ASes of the current Internet, due to geographic spread and possible mergers and acquisitions with other organizations. User addressing can stay the same as in the current Internet, being represented by IP addresses, perhaps IPv6 or a mixture of IPv4 and IPv6. IP addresses are grouped into prefixes, whose reachability is announced through an inter-cloud routing protocol, which is expected to be similar

to Border Gateway Protocol (BGP) [47]. The implications of the flatter cloud-based Internet with fewer clouds than present-day ASes will aid in adopting BGP variants which tackle problems with BGP [238], making routing more scalable and secure. Fewer oscillations are likely to happen as geographically widespread clouds should have fairly stable cloud routes to prefixes. Routing becomes more scalable and easier to manage due to CloudIDs, which are expected to be fewer in number than AS numbers in the current Internet.

Intra-domain routing also becomes important since clouds are expected to cover significant geographical distances, perhaps continents. Hence, a cloud could peer with another at multiple locations. In this case, a cloud can perform hot potato routing [239] and handover the traffic to the neighboring cloud as quickly as possible. This can lead to suboptimal routing where a packet can traverse intercontinental links many times. The routing mechanism should ensure that only a short distance is traversed by any packet. Possible solutions include source routing [71], which is feasible because of the small number of clouds. Another alternative is for a cloud to send the packet through the peering link closer to the destination by using latitude and longitude of the source and the destination IP address.

6.2.6 Cloud Neutrality

We define cloud neutrality as the behavior by a cloud that upholds a user's right to access any content in the Internet. This is similar to net neutrality in today's Internet. Conflicts of interest between clouds providing the same service can lead to throttling of competitor services by clouds that provide Internet services.

While some solutions to this issue can involve legal regulations like the Open Internet Order by the Federal Communications Commission (FCC) in 2010 [240], we propose the existence of third-party clouds, which can provide *neutrality-verification* services to ensure that a cloud is not violating cloud neutrality. This is analogous to certificate authorities which issue certificates for authentication of websites.

A service provider cloud can certainly filter traffic (with limited false positives) based on suspected security threats but should not degrade legitimate competitor traffic or discriminate among traffic classes using similar resources [241]. *e.g.* degrade file transfers over video streaming using the same bandwidth. We believe neutrality authorities will emerge which transparently reveal various standardized neutrality parameters of a service-providing cloud to end-users in the form of "certificates". The net neutrality issue is far from resolved in the present Internet [241] and any solutions proposed can be extrapolated into the cloud-based Internet.

6.2.7 Data Transfer among the Clouds

Cloud-computing as it exists today provides client-server communication: an end-user communicates with the cloud and uses its services. However, clouds may need to communicate with each other because of usage patterns and user diversity. A client can store an object on a cloud A and then want another cloud B to access it and use its features on the object. For example, a client can create a presentation using Google Docs [150] and then use Microsoft PowerPoint to open the file and add features. This will require transfer of the file from Google to Microsoft. Another use case can be when a user U shares an object with another user V who uses the same service from a different cloud. In that case, user V's cloud needs to get the object from user U's cloud, convert formats if necessary and work on it. This necessitates *server-server* communication and poses new challenges [242].

CloudIDs enable clouds to identify each other (Section 6.2.5). However, one also needs a way to uniquely identify an object within a cloud [242]. This can be achieved using URIs much like in the current Internet. However, the implications are that each cloud should run its own authoritative DNS server. This can be bundled with DNS services which need to be provided by every cloud for best user-perceived performance. A cloud can also have a hierarchical DNS structure like Akamai (Section 5.2) for better load balancing and fault tolerance.

Even if the objects in each cloud are uniquely identified using URIs and DNS servers in a cloud, the semantics of inter-cloud communication need to be agreed upon [242]. Transfer can be performed using FTP, but clouds, peering at multiple points, should coordinate to ensure a speedy transfer especially of very large objects over long geographic distances (Section 6.2.5). The incentive for a cloud to transfer an object to another cloud is minimal, and it could very well service the transfer request with inferior nodes. One can solve this using watchdog clouds (Section 6.2.6) as this is a cloud neutrality issue, but it is clear that a standard for inter-cloud communication is important.

6.2.8 Cloud Security and Privacy

Security is critical in any Internet-based infrastructure due to the existence of multiple entities not controlled by a single authority. Access control will gain importance, both within and between clouds [242]. Guidelines and semantics should be in place to specify which users (and clouds) are able to access a certain cloud and the services within. Clouds need to maintain firewalls to filter out harmful content. Inter-cloud peering agreements need to take security of content being exchanged into account.

Moving user data to the cloud creates unique user privacy issues [40]. Replicating user data on data centers spread across geographic boundaries may create regulatory problems due to different privacy laws in different countries. Cloud providers need to provide options that enable a user to control where his/her data is stored. For example, Amazon Simple Storage Service (S3) [243] provides options for storing objects in particular regions of the world.

Privacy concerns also arise in inter-cloud interactions. A user may have granted access of its data to a particular cloud under a privacy agreement. However, inter-cloud communication (Section 6.2.7) may cause the object to be transferred to another

cloud which may have completely different privacy norms. Semantics need to ensure that the owner of an object controls its use.

Just like in today's Internet, an attacker can launch a Distributed Denial of Service (DDoS) attack by compromising hosts over the Internet. We are optimistic that the flatter Internet will make launching such attacks more difficult. Since most services will run on the cloud, which should have an extensive security infrastructure, exploiting service vulnerabilities will not be easy. For example, exploits in file editing software will be unsuccessful at breaching a client since the software will be run on the cloud.

6.3 Future Work

While the previous section has described avenues for future work in cloud computing, we discuss potential work in Internet infrastructure in this section.

Our work on predicting Internet availability can be expanded to include other properties like integrity, scalability and stability. We can also extend our availability prediction framework of Chapter 3 to predict availability of an arbitrary end point as viewed by an arbitrary vantage point by using techniques similar to those used in [120]. Availability of a prefix can be measured from multiple peers, as opposed to considering a single (peer, prefix) combination at a time. Our work can also be expanded to study availability across prefixes which are subprefixes of other prefixes, modifying the long-term availability metric to incorporate the time varying nature of announcement of these prefixes. Other prediction techniques from data mining literature [188] can also be investigated for potential performance improvements over our availability prediction schemes.

Our work on grouping prefixes based on their failure tendencies of Chapter 4 can be extended to investigate whether other features of an Internet prefix like its connectivity to the Internet correlate with its failure tendency. One can also develop efficient ways of constructing BGP molecules so that for any given prefix of interest,

one can search the entire Internet to find prefixes with the closest failure tendencies to the prefix of interest. This can lead to lesser frequency of empty or insufficient sized BGP molecules. One can also use BGP molecules with prefixes ranked in terms of their failure propensity to predict failures with potentially higher performance than that obtained in Section 4.5, which selects prefixes randomly from a molecule. Studying the inherent causes of predictability of prefixes is another topic of future work, since it gives a deeper insight into why some prefixes are more predictable than others. The applications of BGP molecules can be expanded in other realms such as online games and peer-to-peer networks. Also, a study on control plane availability is not complete without studying the data plane. The correlation between the control and data planes is an important topic for future work.

Our studies of cloud-based DNS of Chapter 5 can be extended to study how the performance of Google's data center network and DNS changes at various instants of time, possibly to discover diurnal or weekly patterns. Google's data center network seems to redirect clients based on server load while ignoring latency to data centers for a significant number of cases (Section 5.4). This points to future work on in-depth investigation of server selection techniques, leveraging our geolocation technique of Section 5.3 and methods for prediction of network latency, possibly based on learned trends. We can also use our techniques from this chapter and apply it to other cloud-based DNS systems and content distribution networks to study how our findings change. Given the recent reported partnership of certain DNS systems with cloud-based content providers [224], it would be interesting to study content retrieval through cooperating DNS providers to compare the performance with results obtained in the chapter.

Our work on studying Internet evolution in the realm of cloud computing can be extended to study other ways in which the Internet is evolving, for example due to the use of mobile devices. The access to the Internet and content on the move by devices which are not as powerful as a traditional end-host and over networks not as robust as the wired Internet is a challenging topic. The Internet routing infrastructure needs

to interface with the wireless portion to ensure optimum quality of service to the client. The use of multiple devices by a user accessing content from multiple clouds also gives rise to the issue of content synchronization across devices and across clouds. Investigation of these evolution trends is left as a topic of future research.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Internet World Stats, "Internet Usage Statistics." `http://www.internetworldstats.com/stats.htm`, Retrieved September 2011.

[2] K. P. Birman, *Reliable Distributed Systems: Technologies, Web Services, and Applications*. Springer, first ed., 2005.

[3] GigaOM, "Mary Meeker: Mobile Internet Will Soon Overtake Fixed Internet." `http://gigaom.com/2010/04/12/mary-meeker-mobile-internet-will-soon-overtake-fixed-internet/`, April 2010.

[4] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, pp. 11–33, January 2004.

[5] D. G. Andersen, *Improving End-to-End Availability Using Overlay Networks*. PhD thesis, Massachusetts Institute of Technology, 2005.

[6] C. Labovitz, A. Ahuja, and F. Jahanian, "Experimental Study of Internet Stability and Backbone Failures," in *Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing (FTCS)*, pp. 278–285, 1999.

[7] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet Routing Instability," *IEEE/ACM Transactions on Networking*, vol. 6, pp. 515–528, October 1998.

[8] R. Khosla, S. Fahmy, and Y. C. Hu, "On the Impact of Filters on Analyzing Prefix Reachability in the Internet," in *Proceedings of 18th International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–8, 2009.

[9] University of Oregon, "Route Views Project." `http://www.routeviews.org/`, Retrieved September 2011.

[10] J. Li, M. Guidero, Z. Wu, E. Purpus, and T. Ehrenkranz, "BGP Routing Dynamics Revisited," *SIGCOMM Computer Communications Review*, vol. 37, pp. 5–16, March 2007.

[11] R. Barrett and S. Haar and R. Whitestone, "Routing Snafu Causes Internet Outage." Interactive Week, April 1997.

[12] Stephen A. Farrar, "C&W Routing Instability." NANOG Mail Archives. `http://www.merit.edu/mail.archives/nanog/2001-04/msg00209.html`, April 2001.

[13] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S. F. Wu, and L. Zhang, "Observation and Analysis of BGP behavior Under Stress," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment (IMW)*, pp. 183–195, 2002.

[14] William F. Slater, III, "The Internet Outage and Attacks of October 2002." `http://www.isocchicago.org/internetoutage.pdf`, Retrieved September 2011.

[15] E. Zmijewski, "Threats to Internet Routing and Global Connectivity," in *Proceedings of 20th Annual FIRST Conference*, 2008.

[16] Akamai, "Mideast Outage." `http://www.akamai.com/mideast-outage`, January 2008.

[17] Network World, "Mideast Cable Cuts Tripled Web Latency on Some Routes, Akamai Reports." `http://www.networkworld.com/news/2008/021408-cable-cuts-tripled-web-latency.html`, February 2008.

[18] X. Zhao, D. Massey, S. F. Wu, M. Lad, D. Pei, L. Wang, and L. Zhang, "Understanding BGP Behavior Through a Study of DoD prefixes," *DARPA Information Survivability Conference and Exposition*, April 2003.

[19] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, "Inferring Internet Denial-of-Service Activity," *ACM Transactions on Computer Systems*, vol. 24, pp. 115–139, May 2006.

[20] Arbor Networks, "Arbor Networks' Sixth Annual Worldwide Infrastructure Security Report." `http://www.arbornetworks.com/arbor-networks'-sixth-annual-worldwide-infrastructure-security-report.html`, February 2011.

[21] D. R. Kuhn, "Sources of failure in the public switched telephone network," *Computer*, vol. 30, pp. 31–36, April 1997.

[22] G. Zorpette, "Keeping the phone lines open," *IEEE Spectrum*, vol. 26, pp. 32–36, June 1989.

[23] IATA, "Aircraft Accident Rate is Lowest in History." `http://www.iata.org/pressroom/pr/pages/2011-02-23-01.aspx`, February 2011.

[24] V. Paxson, "End-to-End Routing Behavior in the Internet," *SIGCOMM Computer Communication Review*, vol. 36, pp. 41–56, October 2006.

[25] M. Caesar and J. Rexford, "BGP Routing Policies in ISP networks," *IEEE Network Magazine*, vol. 19, pp. 5–11, November 2005.

[26] L. Gao, "On Inferring Autonomous System Relationships in the Internet," *IEEE/ACM Transactions on Networking*, vol. 9, pp. 733–745, December 2001.

[27] J. Wu, Y. Zhang, Z. M. Mao, and K. G. Shin, "Internet Routing Resilience to Failures: Analysis and Implications," in *Proceedings of the 2007 ACM CoNEXT conference*, pp. 1–12, 2007.

[28] "CIDR Report." `http://www.cidr-report.org/as2.0/`, Retrieved September 2011.

[29] G. Iannaccone, C. nee Chuah, R. Mortier, S. Bhattacharyya, and C. Diot, "Analysis of Link Failures in an IP backbone," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement (IMW)*, pp. 237–242, 2002.

[30] Bellcore, "Automatic Protection Switching for SONET," Tech Report No. SR-NWT-001756, Bellcore, October 1990.

[31] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed Internet Routing Convergence," *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 293–306, 2001.

[32] R. Oliveira, B. Zhang, D. Pei, and L. Zhang, "Quantifying Path Exploration in the Internet," *IEEE/ACM Transactions on Networking*, vol. 17, pp. 445–458, April 2009.

[33] B. Premore, "An Experimental Analysis of BGP Convergence Time," in *Proceedings of the Ninth International Conference on Network Protocols*, 2001.

[34] C. Villamizar and R. Chandra and R. Govindan, "BGP Route Flap Damping, RFC 2439." `https://www.tools.ietf.org/html/rfc2439`, November 1998.

[35] Z. M. Mao, R. Govindan, G. Varghese, and R. H. Katz, "Route Flap Damping Exacerbates Internet Routing Convergence," in *SIGCOMM '02: Proceedings of the 2002 conference on Applications, Technologies, Architectures and Protocols for computer communications*, pp. 221–233, 2002.

[36] T. G. Griffin and G. Wilfong, "An Analysis of BGP Convergence Properties," in *SIGCOMM '99: Proceedings of the 1999 conference on Applications, Technologies, Architectures and Protocols for computer communication*, pp. 277–288, 1999.

[37] The New York Times, "Demand at Target for Fashion Line Crashes Web Site." `http://www.nytimes.com/2011/09/14/business/demand-at-target-for-fashion-line-crashes-web-site.html?_r=1`, September 2011.

[38] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, "Internet Inter-Domain Traffic," in *SIGCOMM '10: Proceedings of the 2010 conference on Applications, Technologies, Architectures and Protocols for Computer Communications*, pp. 75–86, 2010.

[39] InternetNews, "Peer Dispute Leaves Some 'Net Users in the Dark." `http://www.internetnews.com/infra/article.php/3554476`, October 2005.

[40] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," Tech Report No. UCB/EECS-2009-28, Electrical Engineering and Computer Sciences, University of California at Berkeley, 2009. Available: `http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html`.

[41] Google, "Google Public DNS." `http://code.google.com/speed/public-dns/`, Retrieved September 2011.

[42] OpenDNS, "OpenDNS." http://www.opendns.com/, Retrieved September 2011.

[43] IANA, "Internet Assigned Numbers Authority." http://www.iana.org/, Retrieved September 2011.

[44] J. Moy, "OSPF Version2, RFC 2328." http://tools.ietf.org/html/rfc2328, April 1998.

[45] D. Oran, "OSI IS-IS Intra-domain Routing Protocol, RFC 1142." http://tools.ietf.org/html/rfc1142, February 1990.

[46] G. Malkin, "RIP Version 2, RFC 2453." http://tools.ietf.org/html/rfc2453, November 1998.

[47] Y. Rekhter and T. Li and S. Hares, "A Border Gateway Protocol 4 (BGP-4), RFC 4271." http://tools.ietf.org/html/rfc4271, January 2006.

[48] P. V. Mockapetris and K. J. Dunlap, "Development of the domain name system," *SIGCOMM Computer Communication Review*, vol. 25, pp. 112–122, January 1995.

[49] R. Elz and R. Bush and S. Bradner and M. Patton, "Selection and Operation of Secondary DNS Servers, RFC 2182." http://tools.ietf.org/html/rfc2182, July 1997.

[50] D. Dolev, S. Jamin, O. Mokryn, and Y. Shavitt, "Internet Resiliency to Attacks and Failures under BGP Policy Routing," *Computer Networks*, vol. 50, no. 16, pp. 3183–3196, 2006.

[51] Y. Shavitt and Y. Singer, "Beyond Centrality - Classifying Topological Significance using Backup Efficiency and Alternative Paths," *New Journal of Physics, Focus Issue: Complex Networked Systems: Theory and Applications*, 2007.

[52] Y. Singer, "Dynamic Measure of Network Robustness," *IEEE 24th Convention of Electrical and Electronics Engineers in Israel*, pp. 366–370, Nov. 2006.

[53] S. Park, A. Khrabrov, D. Pennock, S. Lawrence, C. Giles, and L. Ungar, "Static and Dynamic Analysis of the Internet's Susceptibility to Faults and Attacks," in *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications (INFOCOM)*, vol. 3, pp. 2144 – 2154, March 2003.

[54] M. Dahlin, B. B. V. Chandra, L. Gao, and A. Nayate, "End-to-End WAN Service Availability," *IEEE/ACM Transactions on Networking*, vol. 11, pp. 300–313, April 2003.

[55] M. Lad, X. Zhao, B. Zhang, D. Massey, and L. Zhang, "Analysis of BGP Update Surge during Slammer Worm Attack," *Distributed Computing (IWDC)*, vol. 2918, pp. 833–835, 2003.

[56] Ricardo V. Oliveira and Rafit Izhak-Ratzin and Beichuan Zhang and Lixia Zhang, "Measurement of Highly Active Prefixes in BGP," in *IEEE GLOBECOM*, 2005.

[57] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang, "BGP Routing Stability of Popular Destinations," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement (IMW)*, pp. 197–202, 2002.

[58] J. Wu, Z. M. Mao, J. Rexford, and J. Wang, "Finding a Needle in a Haystack: Pinpointing Significant BGP Routing Changes in an IP Network," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation (NSDI)*, pp. 1–14, 2005.

[59] S. Agarwal, C. Chuah, S. Bhattacharyya, and C. Diot, "The Impact of BGP Dynamics on Intra-domain Traffic," in *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pp. 319–330, 2004.

[60] F. Wang, Z. M. Mao, J. Wang, L. Gao, and R. Bush, "A Measurement Study on the Impact of Routing Events on End-to-End Internet Path Performance," *SIGCOMM Computer Communication Review*, vol. 36, pp. 375–386, August 2006.

[61] J. Li, Y. Bush, Z. Mao, T. Griffin, M. Roughan, D. Stutzbach, and E. Purpus, "Watching Data Streams Toward a Multi-Homed Sink Under Routing Changes Introduced by a BGP Beacon," in *Passive and Active Measurement Workshop (PAM)*, 2006.

[62] M. Caesar, L. Subramanian, and R. H. Katz, "The Case for an Internet Health Monitoring System," in *Proceedings of the First conference on Hot topics in System Dependability (HotDep)*, pp. 12–12, 2005.

[63] A. Feldmann, O. Maennel, Z. M. Mao, A. Berger, and B. Maggs, "Locating Internet Routing Instabilities," *SIGCOMM Computer Communication Review*, vol. 34, pp. 205–218, August 2004.

[64] R. Teixeira and J. Rexford, "A Measurement Framework for Pin-pointing Routing Changes," in *Proceedings of the ACM SIGCOMM workshop on Network Troubleshooting (NetT)*, pp. 313–318, 2004.

[65] K. Xu, J. Chandrashekar, and Z.-L. Zhang, "A First Step toward Understanding Inter-domain Routing Dynamics," in *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data (MineNet)*, pp. 207–212, 2005.

[66] R. Krishnan, H. V. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, and J. Gao, "Moving Beyond End-to-End Path Information to Optimize CDN Performance," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement (IMC)*, pp. 190–201, 2009.

[67] M. Saxena, U. Sharan, and S. Fahmy, "Analyzing Video Services in Web 2.0: A Global Perspective," in *Proceedings of the 18th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, May 2008.

[68] E. Katz-Bassett, H. V. Madhyastha, J. P. John, A. Krishnamurthy, D. Wetherall, and T. Anderson, "Studying Black Holes in the Internet with Hubble," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 247–262, 2008.

[69] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson, "The End-to-End Effects of Internet Path Selection," *SIGCOMM Computer Communication Review*, vol. 29, pp. 289–299, August 1999.

[70] A. Yip, "NATRON: Overlay Routing to Oblivious Destinations," Master's thesis, Massachusetts Institute of Technology, 2002.

[71] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall, "Improving the Reliability of Internet Paths with One-hop Source Routing," in *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation*, pp. 1–13, 2004.

[72] A. Akella, B. Maggs, S. Seshan, A. Shaikh, and R. Sitaraman, "A Measurement-based Analysis of Multihoming," in *SIGCOMM '03: Proceedings of the 2003 conference on Applications, Technologies, Architectures and Protocols for computer communications*, pp. 353–364, 2003.

[73] D. K. Goldenberg, L. Qiuy, H. Xie, Y. R. Yang, and Y. Zhang, "Optimizing Cost and Performance for Multihoming," in *SIGCOMM '04: Proceedings of the 2004 conference on Applications, Technologies, Architectures and Protocols for computer communications*, pp. 79–92, 2004.

[74] F. Guo, J. Chen, W. Li, and T. cker Chiueh, "Experiences in Building a Multihoming Load Balancing System," in *Proceedings of IEEE INFOCOM*, 2004.

[75] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A Hierarchical Internet Object Cache," in *USENIX Technical Conference*, pp. 153–163, 1995.

[76] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, pp. 281–293, June 2000.

[77] C. Hu, K. Chen, Y. Chen, and B. Liu, "Evaluating Potential Routing Diversity for Internet Failure Recovery," in *Proceedings of the 29th conference on Information communications (INFOCOM)*, pp. 321–325, 2010.

[78] H. Wang, Y. R. Yang, P. H. Liu, J. Wang, A. Gerber, and A. Greenberg, "Reliability as an Interdomain Service," in *SIGCOMM '07: Proceedings of the 2007 conference on Applications, Technologies, Architectures and Protocols for computer communications*, pp. 229–240, 2007.

[79] D. Meyer, L. Zhang, and K. Fall, "Report from the IAB Workshop on Routing and Addressing." `http://tools.ietf.org/id/draft-iab-raws-report-02.txt`, April 2007.

[80] T. Bu, L. Gao, and D. Towsley, "On Characterizing BGP Routing Table Growth," *The International Journal of Computer and Telecommunications Networking*, vol. 45, pp. 45–54, May 2004.

[81] A. Afanasyev, N. Tilley, B. Longstaff, and L. Zhang, "BGP Routing Table: Trends and Challenges," in *Proceedings of the 12th Youth Technological Conference High Technologies and Intellectual Systems*, April 2010.

[82] S. Bellovin, Y. Bush, T. G. Griffin, and J. Rexford, "Slowing Routing Table Growth by Filtering based on Address Allocation Policies." `https://www.cs.princeton.edu/~jrex/papers/filter.pdf`, 2001.

[83] E. Karpilovsky and J. Rexford, "Using Forgetful Routing to Control BGP Table Size," in *Proceedings of the 2006 ACM CoNEXT conference*, pp. 2:1–2:12, 2006.

[84] R. Oliveira, M. Lad, B. Zhang, and L. Zhang, "Geographically Informed Inter-Domain Routing," *IEEE International Conference on Network Protocols*, pp. 103–112, 2007.

[85] D. Krioukov, K. Claffy, K. Fall, and A. Brady, "On Compact Routing for the Internet," *SIGCOMM Computer Communication Review*, vol. 37, pp. 41–52, July 2007.

[86] G. Huston and G. Armitage, "Projecting Future IPv4 Router Requirements from Trends in Dynamic BGP Behaviour," in *Australian Telecommunication Networks and Applications Conference (ATNAC)*, 2006.

[87] A. Elmokashfi, A. Kvalbein, and C. Dovrolis, "On the Scalability of BGP: The Roles of Topology Growth and Update Rate-limiting," in *Proceedings of the 2008 ACM CoNEXT Conference*, pp. 8:1–8:12, 2008.

[88] A. Elmokashfi, A. Kvalbein, and C. Dovrolis, "BGP Churn Evolution: A Perspective from the Core," in *Proceedings of the 29th conference on Information communications (INFOCOM)*, pp. 1208–1216, 2010.

[89] D. Dean, M. Franklin, and A. Stubblefield, "An Algebraic Approach to IP Traceback," in *ACM Transactions on Information and System Security*, pp. 3–12, 2001.

[90] J. Ioannidis and S. M. Bellovin, "Implementing Pushback: Router-Based Defense Against DDoS Attacks," in *Proceedings of Network and Distributed System Security Symposium*, 2002.

[91] K. Park and H. Lee, "On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, Technologies, Architectures and Protocols for computer communications*, pp. 15–26, 2001.

[92] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Network Support for IP traceback," *IEEE/ACM Transactions on Networking*, vol. 9, pp. 226–237, June 2001.

[93] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, B. Schwartz, S. T. Kent, and W. T. Strayer, "Single-packet IP traceback," *IEEE/ACM Transactions on Networking*, vol. 10, pp. 721–734, December 2002.

[94] K. Butler, T. Farley, P. Mcdaniel, and J. Rexford, "A Survey of BGP Security Issues and Solutions," tech. rep., AT&T Labs - Research, Florham Park, NJ, 2004.

[95] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS Performance and the Effectiveness of Caching," *IEEE/ACM Transactions on Networking*, vol. 10, pp. 589–603, October 2002.

[96] J. Pang, J. Hendricks, A. Akella, R. D. Prisco, B. Maggs, and S. Seshan, "Availability, usage, and deployment characteristics of the domain name system," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement (IMC)*, pp. 1–14, 2004.

[97] R. Liston, S. Srinivasan, and E. Zegura, "Diversity in DNS Performance Measures," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement (IMW)*, pp. 19–31, 2002.

[98] V. Pappas, D. Wessels, D. Massey, S. Lu, A. Terzis, and L. Zhang, "Impact of Configuration Errors on DNS Robustness," *IEEE Journal on Selected Areas in Communications*, vol. 27, pp. 275–290, April 2009.

[99] Secure64, "DNS Security News from Secure64." `http://www.secure64.com/dns_security_news`, Retrieved September 2011.

[100] K. Park, V. S. Pai, L. Peterson, and Z. Wang, "CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups," in *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Volume 6*, pp. 14–14, 2004.

[101] G. Kambourakis, T. Moschos, D. Geneiatakis, and S. Gritzalis, "A Fair Solution to DNS Amplification Attacks," in *Proceedings of the Second International Workshop on Digital Forensics and Incident Analysis*, pp. 38–47, 2007.

[102] R. Chandramouli and S. Rose, "An Integrity Verification Scheme for DNS Zone file based on Security Impact Analysis," in *Proceedings of the 21st Annual Computer Security Applications Conference*, pp. 312–321, 2005.

[103] F. Guo, J. Chen, and T. cker Chiueh, "Spoof Detection for Preventing DoS Attacks against DNS Servers," in *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pp. 37–, 2006.

[104] D. Atkins and R. Austein, "Threat Analysis of the Domain Name System (DNS), RFC 3833." `http://tools.ietf.org/html/rfc3833`, August 2004.

[105] R. Arends and R. Austein and M. Larson and D. Massey and S. Rose, "DNS Security Introduction and Requirements, RFC 4033." `http://tools.ietf.org/html/rfc4033`, March 2005.

[106] A. Chakrabarti and G. Manimaran, "Internet Infrastructure Security: A Taxonomy," *IEEE Network*, vol. 16, pp. 13 – 21, December 2002.

[107] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker, "On the Constancy of Internet Path Properties," in *Proceedings of the Internet Measurement Workshop*, pp. 197–211, 2001.

[108] P. Francis, S. Jamin, C. Jin, Y. Jin, V. Paxson, D. Raz, Y. Shavitt, and L. Zhang, "IDMaps: A Global Internet Host Distance Estimation Service," in *Proceedings of IEEE INFOCOM*, pp. 210–217, 2000.

[109] T. S. E. Ng and H. Zhang, "Predicting Internet Network Distance with Coordinates-Based Approaches," in *Proceedings of IEEE INFOCOM*, pp. 170–179, 2001.

[110] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A Decentralized Network Coordinate System," in *SIGCOMM '04: Proceedings of the 2004 conference on Applications, Technologies, Architectures and Protocols for computer communications*, pp. 15–26, 2004.

[111] B. Wong, A. Slivkins, and E. G. Sirer, "Meridian: A Lightweight Network Location Service without Virtual Coordinates," in *SIGCOMM '05: Proceedings of the 2005 conference on Applications, Technologies, Architectures and Protocols for computer communications*, pp. 85–96, 2005.

[112] M. Zhang, C. Zhang, V. Pai, L. Peterson, and Y. Wang, "PlanetSeer: Internet Path Failure Monitoring and Characterization in Wide-Area Services," in *Proceedings of OSDI*, pp. 167–182, 2004.

[113] Y. Shavitt and T. Tankel, "On the Curvature of the Internet and its Usage for Overlay Construction and Distance Estimation," in *Proceedings of INFOCOM*, pp. 1–11, March 2004.

[114] Y. Mao and L. K. Saul, "Modeling Distances in Large-scale Networks by Matrix Factorization," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement (IMC)*, pp. 278–287, 2004.

[115] H. V. Madhyastha, T. Anderson, A. Krishnamurthy, N. Spring, and A. Venkataramani, "A Structural Approach to Latency Prediction," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (IMC)*, pp. 99–104, 2006.

[116] Q. He, C. Dovrolis, and M. Ammar, "On the Predictability of Large Transfer TCP Throughput," in *SIGCOMM '05: Proceedings of the 2005 conference on Applications, Technologies, Architectures and Protocols for computer communications*, pp. 145–156, 2005.

[117] Y. Qiao, J. Skicewicz, and P. Dinda, "An Empirical Study of the Multiscale Predictability of Network Traffic," in *IEEE Proceedings of HPDC*, pp. 66–76, 2003.

[118] M. Swany and R. Wolski, "Multivariate Resource Performance Forecasting in the Network Weather Service," in *ACM/IEEE conference on Supercomputing*, 2002.

[119] S. Vazhkudai, J. M. Schopf, and I. T. Foster, "Predicting the Performance of Wide Area Data Transfers," in *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 270–, 2002.

[120] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iPlane: An Information Plane for Distributed Services," in *Proceedings of OSDI*, pp. 367–380, November 2006.

[121] H. V. Madhyastha, E. Katz-Bassett, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iPlane Nano: Path Prediction for Peer-to-Peer Applications," in *Proceedings of NSDI*, pp. 137–152, 2009.

[122] N. Feamster, D. G. Andersen, H. Balakrishnan, and M. F. Kaashoek, "Measuring the Effects of Internet Path Faults on Reactive Routing," in *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp. 126–137, 2003.

[123] Y. Zhang, Z. M. Mao, and J. Wang, "A Framework for Measuring and Predicting the Impact of Routing Changes," in *Proceedings of INFOCOM*, pp. 339–347, 2007.

[124] A. Dhamdhere and C. Dovrolis, "Ten Years in the Evolution of the Internet Ecosystem," in *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement (IMC)*, pp. 183–196, 2008.

[125] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "The Flattening Internet Topology: Natural Evolution, Unsightly Barnacles or Contrived Collapse?," in *Proceedings of the 9th international conference on Passive and active network measurement (PAM)*, pp. 1–10, 2008.

[126] Om Malik, "Wholesale Internet Bandwidth Prices Keep Falling." `http://gigaom.com/2008/10/07/wholesale-internet-bandwidth-prices-keep-falling/`, October 2008.

[127] G. Goth, "New Internet Economics Might Not Make It to the Edge," *IEEE Internet Computing*, vol. 14, pp. 7–9, January 2010.

[128] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The Nature of Data Center Traffic: Measurements & Analysis," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference (IMC)*, pp. 202–208, 2009.

[129] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding Data Center Traffic Characteristics," *SIGCOMM Computer Communication Review*, vol. 40, pp. 92–99, January 2010.

[130] V. K. Adhikari, S. Jain, and Z.-L. Zhang, "YouTube Traffic Dynamics and its Interplay with a Tier-1 ISP: an ISP perspective," in *Proceedings of the 10th annual conference on Internet measurement (IMC)*, pp. 431–443, 2010.

[131] Y. Chen, S. Jain, V. Adhikari, Z.-L. Zhang, and K. Xu, "A First Look at Inter-Data Center Traffic Characteristics via Yahoo! datasets," in *Proceedings of IEEE INFOCOM*, pp. 1620 –1628, April 2011.

[132] Y. A. Wang, C. Huang, J. Li, and K. Ross, "Estimating the Performance of Hypothetical Cloud Service Deployments: A Measurement-based Approach," in *Proceedings of IEEE INFOCOM*, pp. 2372 –2380, April 2011.

[133] C. Huang, N. Holt, Y. A. Wang, A. Greenberg, J. Li, and K. W. Ross, "A DNS Reflection Method for Global Traffic Management," in *Proceedings of the USENIX Annual Technical Conference (USENIXATC)*, pp. 1–6, 2010.

[134] J. S. Gwertzman and M. Seltzer, "The Case for Geographical Push-caching," in *Fifth Workshop on Hot Topics in Operating Systems (HotOS-V)*, pp. 51 –55, May 1995.

[135] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi, "Web Caching with Consistent Hashing," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 31, pp. 1203–1213, May 1999.

[136] C. Partridge and T. Mendez and W. Milliken, "Host Anycasting Service, RFC 1546." http://tools.ietf.org/html/rfc1546, November 1993.

[137] Google, "Google Public DNS Performance Benefits." http://code.google.com/speed/public-dns/docs/performance.html, Retrieved September 2011.

[138] H. Ballani, P. Francis, and S. Ratnasamy, "A Measurement-based Deployment Proposal for IP Anycast," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (IMC)*, pp. 231–244, 2006.

[139] C. Huang, D. Maltz, J. Li, and A. Greenberg, "Public DNS system and Global Traffic Management," in *Proceedings of IEEE INFOCOM*, pp. 2615 –2623, April 2011.

[140] A. Su, D. Choffnes, A. Kuzmanovic, and F. Bustamante, "Drafting Behind Akamai: Inferring Network Conditions Based on CDN Redirections," *IEEE/ACM Transactions on Networking*, vol. 17, pp. 1752 –1765, December 2009.

[141] M. Andrews, B. Shepherd, A. Srinivasan, P. Winkler, and F. Zane, "Clustering and Server Selection using Passive Monitoring," in *Proceedings of Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 3, pp. 1717 – 1725, 2002.

[142] J. Pan, Y. T. Hou, and B. Li, "An Overview of DNS-based Server Selections in Content Distribution Networks," *Computer Networks*, vol. 43, no. 6, pp. 695 – 711, 2003.

[143] American Registry for Internet numbers, "ARIN Whois." http://whois.arin.net/ui/query.do, Retrieved September 2011.

[144] N. Leavitt, "Is Cloud Computing Really Ready for Prime Time?," *Computer*, vol. 42, pp. 15–20, January 2009.

[145] B. P. Rimal, E. Choi, and I. Lumb, "A Taxonomy and Survey of Cloud Computing Systems," in *NCM '09: Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC*, pp. 44–51, 2009.

[146] Amazon, "Amazon Web Services." http://aws.amazon.com, Retrieved September 2011.

[147] IBM, "IBM Smart Cloud." http://www.ibm.com/cloud-computing/us/en/, Retrieved September 2011.

[148] Facebook, "Facebook." http://www.facebook.com/, Retrieved September 2011.

[149] Google, "Google App Engine." http://code.google.com/appengine/, Retrieved September 2011.

[150] Google, "Google Docs." http://docs.google.com, Retrieved September 2011.

[151] Oracle, "Oracle Software as a Service." http://www.oracle.com/us/products/ondemand/saas-068569.html, Retrieved September 2011.

[152] Microsoft, "Microsoft Cloud Computing Solutions." http://www.microsoft.com/en-us/cloud/default.aspx, Retrieved September 2011.

[153] Apple, "Apple iCloud." http://www.apple.com/icloud/, Retrieved September 2011.

[154] Google, "Google Public DNS Security Benefits." http://code.google.com/speed/public-dns/docs/security.html, Retrieved September 2011.

[155] B. Ager, W. Muehlbauer, G. Smaragdakis, and S. Uhlig, "Comparing DNS Resolvers in the Wild," in *Proceedings of Internet Measurement Conference (IMC)*, pp. 15–21, November 2010.

[156] D.-F. Chang, R. Govindan, and J. Heidemann, "The Temporal and Topological Characteristics of BGP Path Changes," in *Proceedings of IEEE ICNP*, pp. 190–199, 2003.

[157] R. Beverly, K. Sollins, and A. Berger, "SVM Learning of IP Address Structure for Latency Prediction," in *Proceedings of the 2006 SIGCOMM workshop on mining network data (MineNet)*, pp. 299–304, 2006.

[158] R. Beverly and M. Afergan, "Machine learning for Efficient Neighbor Selection in Unstructured P2P networks," in *Proceedings of the 2nd USENIX workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysML)*, pp. 1:1–1:6, 2007.

[159] M. Mirza, J. Sommers, P. Barford, and X. Zhu, "A Machine Learning Approach to TCP Throughput Prediction," *IEEE/ACM Transactions on Networking*, vol. 18, pp. 1026–1039, August 2010.

[160] R. Beverly, "A Robust Classifier for Passive TCP/IP Fingerprinting," in *Proceedings of the 5th Passive and Active Measurement Workshop (PAM)*, 2004.

[161] R. Beverly and K. Sollins, "An Internet Protocol Address Clustering Algorithm," in *Proceedings of the Third conference on Tackling Computer Systems Problems with Machine Learning Techniques (SysML)*, pp. 1–5, 2008.

[162] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based Change Detection: Methods, Evaluation, and Applications," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement (IMC)*, pp. 234–247, 2003.

[163] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," in *Internet Mathematics*, pp. 636–646, 2002.

[164] AT&T, "AT&T High Speed Internet Business Edition Service Level Agreements." http://www.att.com/gen/general?pid=6622, Retrieved September 2011.

[165] Sprint, "Sprint Service Level Agreements." http://www.sprintworldwide.com/english/solutions/sla/, Retrieved September 2011.

[166] P. Pongpaibool and H. S. Kim, "Providing end-to-End Service Level Agreements across Multiple ISP Networks," *Computer Networks*, vol. 46, no. 1, pp. 3–18, 2004.

[167] R. Keralapura, C. N. Chuah, G. Iannaccone, and S. Bhattacharyya, "Service Availability: A New approach to Characterize IP Backbone Topologies," *Twelfth IEEE International Workshop on Quality of Service (IWQOS)*, pp. 232–241, June 2004.

[168] R. Bush, O. Maennel, M. Roughan, and S. Uhlig, "Internet Optometry: Assessing the Broken Glasses in Internet Reachability," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference (IMC)*, pp. 242–253, 2009.

[169] B. Cohen, "Incentives Build Robustness in BitTorrent." `http://www.bittorrent.org/bittorrentecon.pdf`, 2003.

[170] RIPE, "RIPE Network Coordination Centre." `http://www.ris.ripe.net/source/`, Retrieved September 2011.

[171] B. Zhang, V. Kambhampati, M. Lad, D. Massey, and L. Zhang, "Identifying BGP Routing Table Transfers," in *Proceedings of ACM MineNet workshop*, 2005.

[172] E. Chen and J. Stewart, "A Framework for Inter-Domain Route Aggregation, RFC 2519." `http://tools.ietf.org/html/rfc2519`, February 1999.

[173] A. Broido, E. Nemeth, and K. Claffy, "Internet Expansion, Refinement and Churn," in *European Transactions on Telecommunications*, 2002.

[174] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2nd ed., 2005.

[175] John Shepler, "The Holy Grail of Five-nines reliability." `http://searchnetworking.techtarget.com/generic/0,295582,sid7_gci1064318,00.html`, 2005.

[176] E. L. Lehmann and J. P. Romano, *Testing Statistical Hypotheses*. Springer, New York, 3rd ed., 2005.

[177] S. S. Sawilowsky, "Fermat, Schubert, Einstein, and Behrens-Fisher: The Probable Difference Between Two Means When $\sigma_1^2 \neq \sigma_2^2$," *Journal of Modern Applied Statistical Methods*, vol. 1, no. 2, 2002.

[178] L. Hamel, "Model Assessment with ROC Curves," in *The Encyclopedia of Data Warehousing and Mining*, Idea Group Publishers, 2nd ed., 2008.

[179] T. Fawcett, "ROC Graphs: Notes and Practical Considerations for Researchers," Tech Report HPL-2003-4, HP Laboratories, 2003. Available: `http://home.comcast.net/~tom.fawcett/public_html/papers/ROC101.pdf`.

[180] R. Webster West, "T Distribution Calculator." `http://www.stat.tamu.edu/~west/applets/tdemo.html`, Retrieved September 2011.

[181] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[182] J. R. Quinlan, P. J. Compton, K. A. Horn, and L. Lazarus, "Inductive Knowledge Acquisition: A Case Study," in *Proceedings of the Second Australian Conference on Applications of expert systems*, pp. 137–156, 1987.

[183] Second PacNOG Meeting, Conference and Educational Workshop, "BGP Attributes and Policy Control." http://www.pacnog.org/pacnog2/track2/routing/b1-1up.pdf, February 1999.

[184] M. Caesar, L. Subramanian, and R. H. Katz, "Towards Localizing Root Causes of BGP Dynamics," Tech. Rep. UCB/CSD-03-1292, EECS Department, University of California, Berkeley, 2003.

[185] A. Broido and K. Claffy, "Analysis of RouteViews BGP data: Policy Atoms," in *Network-Related Data Management (NRDM) workshop*, 2001.

[186] PlanetLab, "PlanetLab: An Open Platform for Developing, Deploying, and Accessing Planetary-scale Services." http://www.planet-lab.org/, Retrieved September 2011.

[187] X. Zhao, D. Pei, L. Wang, D. Massey, A. Mankin, S. F. Wu, and L. Zhang, "An Analysis of BGP Multiple Origin AS (MOAS) Conflicts," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement (IMW)*, pp. 31–35, 2001.

[188] D. J. Hand, H. Mannila, and P. Smyth, *Principles of Data Mining*. The MIT Press, 1st ed., 2001.

[189] MaxMind, "GeoLite City." http://www.maxmind.com/app/geolitecity, Retrieved September 2011.

[190] R. Sinnott, "Virtues of the Haversine," *Sky and Telescope*, vol. 68, no. 2, p. 159, 1984.

[191] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 2nd ed., 2001.

[192] S. Burkle, *BGP Convergence Analysis*. PhD thesis, Universitat des Saarlandes, 2003.

[193] Ao-Jan Su and A. Kuzmanovic, "Thinning Akamai," in *Proceedings of the 8th ACM SIGCOMM conference on Internet Measurement (IMC)*, pp. 29–42, 2008.

[194] Cymru, "IP to ASN Mapping." http://www.team-cymru.org/Services/ip-to-asn.html, Retrieved September 2011.

[195] Akamai, "Akamai Solutions." http://www.akamai.com/html/solutions/index.html, Retrieved September 2011.

[196] Google, "Google Milestones." http://www.google.com/corporate/history.html, Retrieved September 2011.

[197] Google, "Think Big with a Gig: Our Experimental Fiber Network." http://googleblog.blogspot.com/2010/02/think-big-with-gig-our-experimental.html, February 2010.

[198] C. Labovitz, "How Big is Google?." http://asert.arbornetworks.com/2010/03/how-big-is-google/, March 2010.

[199] C. Labovitz, "Internet Traffic and Content Consolidation," in *Proceedings of the seventy seventh Internet Engineering Task Force meeting*, March 2010.

[200] C. Huang, A. Wang, J. Li, and K. W. Ross, "Measuring and Evaluating Large-scale CDNs (Paper Withdrawn at Microsoft's request)," in *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement (IMC)*, pp. 15–29, 2008.

[201] die.net, "dig(1) - Linux Man Page." http://linux.die.net/man/1/dig, Retrieved September 2011.

[202] J. A. Muir and P. C. V. Oorschot, "Internet Geolocation: Evasion and Counterevasion," *ACM Computing Surveys*, vol. 42, pp. 4:1–4:23, December 2009.

[203] MaxMind, "MaxMind GeoIP City Database." http://www.maxmind.com/app/city, Retrieved September 2011.

[204] MaxMind, "MaxMind Web Services." http://www.maxmind.com/app/web_services, Retrieved September 2011.

[205] MaxMind, "MaxMind GeoIP City Accuracy." http://www.maxmind.com/app/city_accuracy, Retrieved September 2011.

[206] MathWorks, "Matlab for Technical Computing." http://www.mathworks.com, Retrieved September 2011.

[207] Data Center Knowledge, "Google Data Center FAQ." http://www.datacenterknowledge.com/archives/2008/03/27/google-data-center-faq/, March 2008.

[208] Royal Pingdom, "Map of all Google Data Center Locations." http://royal.pingdom.com/2008/04/11/map-of-all-google-data-center-locations/, April 2008.

[209] IRR, "Internet Routing Registry." http://www.irr.net/, Retrieved September 2011.

[210] ARIN, "ARIN WHOIS Database Search." https://ws.arin.net/whois/, Retrieved September 2011.

[211] RIPE, "RIPE Routing Information Service (RIS)." http://www.ripe.net/np/ris/, Retrieved September 2011.

[212] APNIC, "APNIC Whois." http://wq.apnic.net/apnic-bin/whois.pl, Retrieved September 2011.

[213] AFRINIC, "AFRINIC Whois." http://www.afrinic.net/cgi-bin/whois, Retrieved September 2011.

[214] LACNIC, "LACNIC Whois." http://lacnic.net/cgi-bin/lacnic/whois, Retrieved September 2011.

[215] IP2Location, "IP GeoLocator." http://www.ip2location.com, Retrieved September 2011.

[216] Google, "Introduction to Google Public DNS." http://code.google.com/speed/public-dns/index.html, Retrieved September 2011.

[217] E. Katz-Bassett, J. P. John, A. Krishnamurthy, D. Wetherall, T. Anderson, and Y. Chawathe, "Towards IP Geolocation using Delay and Topology Measurements," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (IMC)*, pp. 71–84, 2006.

[218] Google, "Google Insights." `http://www.google.com/insights/search/`, July 2010.

[219] Akamai, "Akamai Customer Stories." `http://www.akamai.com/html/customers/index.html`, Retrieved September 2011.

[220] Akamai, "Akamai Net Usage Index." `http://www.akamai.com/html/technology/nui/news/index.html`, Retrieved September 2011.

[221] T. M. Sellke, "How Many IID Samples Does it Take to See all the Balls in a Box?," *The Annals of Applied Probability*, vol. 5, pp. 294–309, February 1995.

[222] C. Contavalli and W. van der Gaast and S. Leach and D. Rodden, "Client IP information in DNS requests." IETF Internet Draft draft-vandergaastedns-client-ip-00.txt, January 2010.

[223] A. F. Internet, "A Faster Internet - The Global Internet Speedup." `http://afasterinternet.com/`, Retrieved September 2011.

[224] CNET News, "Google, OpenDNS Add Geo Speed Boost to Net." `http://news.cnet.com/8301-30685_3-20098994-264/google-opendns-add-geo-speed-boost-to-net/`, August 2011.

[225] P. Wendell and J. W. Jiang and M. J. Freedman and Jennifer Rexford, "DONAR: Decentralized Server Selection for Cloud Services," in *SIGCOMM '10: Proceedings of the 2010 conference on Applications, Technologies, Architectures and Protocols for computer communications*, pp. 231–242, 2010.

[226] M. Marcon, B. Viswanath, M. Cha, and K. Gummadi, "Sharing Social Content from Home: A Measurement-driven Feasibility Study," in *Proceedings of NOSSDAV*, pp. 45–50, 2011.

[227] M. N. Center, "Microsoft to Acquire Skype." `http://www.microsoft.com/presspass/press/2011/may11/05-10corpnewspr.mspx`, Retrieved September 2011.

[228] Google, "Google To Acquire YouTube for $1.65 Billion in Stock." `http://www.google.com/intl/en/press/pressrel/google_youtube.html`, Retrieved September 2011.

[229] Alexa, "Top Sites by Country." `http://www.alexa.com/topsites/countries`, Retrieved September 2011.

[230] comScore, "comScore Releases May 2011 U.S. Search Engine Rankings." `http://www.comscore.com/Press_Events/Press_Releases/2011/6/comScore_Releases_May_2011_U.S._Search_Engine_Rankings`, Retrieved September 2011.

[231] comScore, "Social Networking Explodes Worldwide." `http://www.comscore.com/Press_Events/Press_Releases/2008/08/Social_Networking_World_Wide`, Retrieved September 2011.

[232] Facebook, "Statistics." https://www.facebook.com/press/info.php?statistics, Retrieved September 2011.

[233] Google, "Google Plus." https://plus.google.com, Retrieved September 2011.

[234] T. W. S. Journal, "Google+ Pulls In 20 Million in 3 Weeks." http://online.wsj.com/article/SB10001424053111904233404576460394032418286.html, Retrieved September 2011.

[235] Wikipedia, "United States v. AT&T." http://en.wikipedia.org/wiki/United_States_v._AT%26T, Retrieved September 2011.

[236] Wikipedia, "United States v. Microsoft." http://en.wikipedia.org/wiki/United_States_v._Microsoft, Retrieved Septe,ber 2011.

[237] H. Chang, S. Jamin, and W. Willinger, "To Peer or Not to Peer: Modeling the Evolution of the Internet's AS-Level Topology," in *Proceedings of INFOCOM*, 2006.

[238] N. Feamster, H. Balakrishnan, and J. Rexford, "Some Foundational Problems in Interdomain Routing," in *HotNets*, November 2004.

[239] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford, "Dynamics of Hot-Potato Routing in IP Networks," in *SIGMETRICS Performance*, pp. 307–319, 2004.

[240] F. C. Commission, "The Open Internet." http://www.fcc.gov/cgb/consumerfacts/openinternet.pdf, Retrieved September 2011.

[241] J. M. Peha, "The Benefits and Risks of Mandating Network Neutrality, and the Quest for a Balanced Policy," in *34th Telecommunications Policy Research Conference*, 2006.

[242] V. Cerf, "Re-thinking the Internet." http://www.youtube.com/watch?v=VjGuQ1GJkYc, Retrieved September 2011.

[243] Amazon, "Amazon Simple Storage Service." http://aws.amazon.com/s3, Retrieved September 2011.

VITA

VITA

Ravish Khosla received his Bachelors of Technology (Honours) from Indian Institute of Technology (IIT) Kharagpur, India in 2004 with a major in Electrical Engineering and a minor in Electronics and Electrical Communication Engineering. He received a silver medal from IIT Kharagpur for having the highest GPA in his department's graduating class. Ravish then began his graduate studies at Purdue University and graduated with a Masters of Science in Electrical and Computer Engineering in 2006. His Masters thesis was on reliable data dissemination in energy constrained sensor networks under the guidance of Dr. Saurabh Bagchi. Ravish completed his PhD in Electrical and Computer Engineering in December 2011 from Purdue University under the guidance of Dr. Sonia Fahmy and Dr. Y. Charlie Hu. He has worked as a Teaching and Research Assistant at Purdue University and has received Magoon Award for Teaching Excellence and Graduate Teacher Certificate from Purdue.

Ravish's research interests are in both wired and in wireless networks, especially in routing protocols. He is also interested in cloud computing, data centers, machine learning and security.