MINIMIZING WIDE-AREA PERFORMANCE DISRUPTIONS IN INTER-DOMAIN ROUTING

YAPING ZHU

A DISSERTATION
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE
BY THE DEPARTMENT OF
COMPUTER SCIENCE

ADVISOR: PROFESSOR JENNIFER REXFORD

SEPTEMBER 2011

Abstract

The Internet is the platform for most of our communications needs today. The networks underlying the Internet undergo continual change – both planned changes (e.g., adding a new router) or unplanned failures. Unfortunately, these changes can lead to performance disruptions, which affect the user experience. Because of this, network operators have to quickly diagnose and fix any problems that arise. Diagnosing wide-area performance disruptions is challenging: first, each network has limited visibility into other networks, so network operators must collect and analyze measurements of routing and traffic data in order to infer the root cause of the disruption; second, there are so many potential factors which might lead to performance disruptions, and these factors are usually interdependent of each other; third, there are no formalized ways to define metrics and classify the performance disruption according to the causes, thus network diagnosis is usually done in an ad-hoc manner.

The thesis conducts two case studies to diagnose wide-area performance disruptions from the perspectives of a large tier-1 Internet Service Provider (ISP) and a large content distribution network (CDN): i) From the ISP's perspective, we designed and implemented a system that tracks inter-domain route changes at scale and in real time. Our system can be used as the building block for many diagnosis tools for the ISPs. ii) From the CDN's perspective, we focus on diagnosing wide-area network changes which resulted in latency increases to access the services in the CDN. We designed a method for automatically classifying large increases of latency, and evaluated our techniques on one month of measurement data to identify major sources of high latency for the CDN.

Stepping back, the difficulties in network diagnosis can be traced back to the inter-domain routing protocol itself. Based on the lessons learned from the case studies, we refactor the border gateway protocol (BGP), the main inter-domain routing protocol in two ways: first, since the network operator has visibility into its own network and some limited visibility in the neighboring networks, we propose to select a route only based on the next-hop AS (instead of the networks further away); second, the BGP protocol was designed as a way to exchange path availability information between independent networks, not with the operational challenges of performance, security, and traffic engineering. This has led many to propose additional BGP attributes that satisfy the operational needs. These proposals make the protocol and configuration more complicated, and thus more error-prone and more difficult for network operators to diagnose problems. Instead, we propose simplifying the protocol, and in effect enable addressing the operational challenges outside the protocol. Our proposal of next-hop BGP not only simplifies the protocol, but also has the benefits of fast convergence, incentive compatibility, and easier support for multi-path routing.

Acknowledgments

I am extremely fortunate to have Jennifer Rexford as my advisor. Jen is absolutely a great researcher in the field of network systems. Thanks to her solid expertise, and I have enjoyed inspirational discussions with her on various research ideas. She also guided me to think about the practical implications of my research projects, and helped me to improve my paper writing and presentation skills. In additional to a great research advisor, Jen is also a great mentor for any kind of advice. I am very grateful for her encouragement and support for me to do several internships during my graduate studies, which broadened my understandings of the industry. Jen is even supportive of my extracurricular activities (e.g., running the Graduate Engineering Council): she encourages me to improve my communications skills, and gives me the freedom to pursue my passion. Jen is undoubtedly a role model for her students, and she has amazed me with her genuine kindness, her patience, and also her optimism. With all these, I believe she is the best advisor that a graduate student could ask for.

I am also really lucky to have worked with Aman Shaikh. As my internship mentor at AT&T Research, Aman gave me invaluable advice and feedback on my research, which helped to shape the first chapter of this dissertation. He is an expert in the area of Internet routing, and he shared lots of his knowledge with me by helping me understand the technical details of the problems. As my paper co-author and thesis reader, he is rigorous on both the research and the writing, and he always takes one extra step to make sure that everything is presented correctly and clearly. His solid expertise and patience set an example for a great mentor. In addition, Aman has a very pleasant personality, which makes him a fun person to work with.

I would like to thank Vivek Pai, Michael Freedman, and Andrea LaPaugh for serving on my thesis committee and giving me valuable feedback.

I am really thankful to Subhabrata Sen. Subhabrata is a fantastic mentor I met at AT&T Research. He shared his passion for solving practical problems in industrial research with me. I am also fortunate to have met and worked with Aspi Siganporia, Benjamin Helsley, Sridhar Srinivasan, and Bo Fu at Google. I owe a special thank you to them for introducing me to the operational challenges of running a large CDN network, and sharing their data set for my research. I enjoyed working together with them on network troubleshooting. Benjamin and Bo also patiently helped me improve my programming style. I am also lucky to have worked with Michael Schapira. As my paper co-author, Michael is inspirational, and the collaboration with him was truly pleasant. Also, I would like to thank Dan Wendlandt, Murtaza Motiwala, Vytautas Valancius, and Mukarram Bin Tariq for their invaluable feedback on my research.

At Princeton, especially in the Cabernet research group, I am lucky to be surrounded by truly bright people. I benefit tremendously from the fruitful discussions with them. Wen Xu guided me in the first year of my graduate studies. Sharon Goldberg and Haakon Ringberg gave me great feedback on my project ideas, and helped me prepare for presentations. Special thanks to Haakon who gave me great guidance for the job search. Alex Fabrikant and Matthew Caesar helped answer many detailed technical questions. Jiayue He and Yi Wang taught me people skills. I also owe thanks to Robert Harrison and Eric Keller for proofreading parts of this dissertation. I would also like to thank Ioannis Avramopoulos, Wenjie Jiang, Changhoon Kim, Steven Ko, Srinivas Narayana, Martin Suchara, and Minlan Yu for their invaluable feedback on my research.

I thank my friends from the Computer Science Department (Rebecca Fiebrink, Jialu Huang, Sid Sen, Umar Syed, Qian Xi, Siyu Yang, and Yun Zhang) for their support over the years. I am grateful for my friends from the Princeton Graduate Engineering Council (Theresa Cho, Stephen Friedfeld, Josephine Kelly, Yi Ma, and Raghuveer Vinukollu) for their tremendous help make all the student events possible. I would like to thank Jieying Chen, Nancy Zheng, and Shuyi Chen for making my internship fun and enjoyable. Last but not the least, I am indebted to Karl Zaininger, Karen Schwartz and Elaine Kung for their mentorship and their invaluable support. Also, special thanks to Melissa Lawson for her help over the years of my graduate studies.

My research work was supported by Princeton University, the National Science Foundation (NSF), the Air Force Office of Scientific Research (AFOSR), and Google Inc. Thanks to all for making this dissertation possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Today, the Internet is the de facto platform for most of our communications needs, including electronic mail, instant messaging, and telephony. In additional to these one-to-one communication services, there are also many services on the Internet, like web search, collaborative editing, video streaming, etc. Ensuring the performance of these services is crucial to their usability, and network operators go to great lengths to improve the network performance to support the demand of these services.

The Internet is composed of thousands of independent networks. These networks are mainly classified into three categories: the ISPs (Internet Service Providers) which provide connectivity to networks, by transiting IP packets between them; the CDNs (Content Distribution Networks) which use geographically distributed resources to provide ubiquitous services to users; and stub networks, which are connected to ISPs in order to access the rest of the Internet.

The servers and computers on the Internet are connected by routers. Routers implement the functionality of routing, which is the process of computing paths and delivering traffic to its destination. Between networks, adjacent routers in neighbor networks utilize an inter-domain routing protocol to exchange routes – with the Border Gateway Protcol (BGP) [1] being the main inter-domain routing protocol used in the Internet today.

Improving network performance is essential in ISPs' and CDNs' network operations. ISPs are motivated to provide good delivery of IP packets for their customers. Maximizing network performance is also crucial for CDNs hosting services like web search and video streaming. Better performance can help ISPs and CDNs to attract more customers and increase their revenue.

However, managing and operating networks comprising the Internet is a daunting task, especially since the original Internet and protocols underpinning it were not designed with management and performance challenges in mind. As a result, even answering basic questions like what path a given packet traversed, or why the end-to-end latency from a particular client to the server increased at a given time is a challenging task. However, determining the answers to these seemingly simple questions is crucial for performing a wide range of tasks in network operations.

In this dissertation, we looked at two case studies to examine the performance challenges that ISPs and CDNs face today. We propose techniques for a tier-1 ISP and a large CDN to diagnose performance disruptions for each. Based on these techniques, we design and implement systems for the ISP and CDN to use in network operations. Stepping back, observing that many of the performance problems are caused by today's BGP protocol, we propose a variant of the routing protocol, which is incrementally deployable and designed for better performance.

In this chapter, we first give a high-level overview of Internet routing and network management. We then discuss the challenges of minimizing performance disruptions in the wide area for ISPs and CDNs, and present the summary of our techniques to address these challenges. Lastly, we raise the performance problems caused by routing, and outline the design of a protocol variant for better performance.

1.1 Overview of Internet Routing and Network Diagnosis

1.1.1 Network Changes Affect User Experience

The Internet is divided into tens of thousands of networks called Autonomous Systems (ASes), each of which has network elements, such as routers and servers, usually operated by a single organization. Inside each AS, the routers run an Interior Gateway Protocol (IGP) to determine paths between routers, servers, and hosts in the same AS. Between networks, the Border Gateway Protocol (BGP) allows ASes to exchange information about how to reach external destinations.

If a router or link fails in the network, the routers will disseminate the changes of network reachability as routing updates, and recompute the route. The failure might cause parts of the network to become unreachable. If the destination is still reachable, routers will compute and update the new routes. During the routing convergence process, when routers are exchanging information and calculating a new path, different routers might get inconsistent routes for the same destination, and this process is called the *routing* *convergence*. Because of these inconsistencies, IP packets might circulate among the routers, or even get dropped. In this situation, users would experience performance degradation.

Besides equipment failures, network congestion is another cause of performance disruptions. Network congestion happens when too many IP packets are waiting to be transmitted over the same link. If the demand is more than the link capacity, some packets are dropped by the router. To cope with this, end hosts run a protocol with congestion control mechanism which reduces the rate of sending packets when congestion is detected.

Both equipment failures and network congestion cause network performance disruptions, where IP packets are dropped or retransmitted. Users might not be able to reach the destination, or experience higher network latency. Most routing convergence events last for minutes [2]. The network operators have to fix the failed routers or links, especially if some destinations become unreachable because of the failure. As for network congestion, if the problem persists (e.g. some links are always overloaded), then the network operators also have to react: the best approach is to upgrade the links with higher capacity, but a more short term fix is to explicitly redirect some traffic to alternate routes to avoid the congested links. The latter approach is called *traffic engineering*.

But how do the network operators know that the network changes are causing performance disruptions? And, how could they pinpoint the root cause of the problem? In the next subsection, we will introduce general steps for diagnosing performance problems, as well as the challenges.

1.1.2 Framework for Diagnosing Performance Problems

In this subsection, we give a high-level overview of the network diagnosis procedure inside a single network. We use an enterprise network (i.e., the network operated by a corporation) as an example to illustrate how performance disruptions are typically diagnosed. In order to better manage the network, the network operators usually setup monitoring systems for the routers and links. These monitoring systems trigger alarms that notify the network operator when, for example, equipment fails. If the customers are experiencing bad performance, the customers would complain, and the network operators also have to react in response.

The general procedure for performance diagnosis consists of three steps, as illustrated in Figure 1.1: measure, diagnose and fix. First, the operators have to measure the network status and changes. Based on the measurement results, they then investigate the problem (e.g., the problem is caused by network device

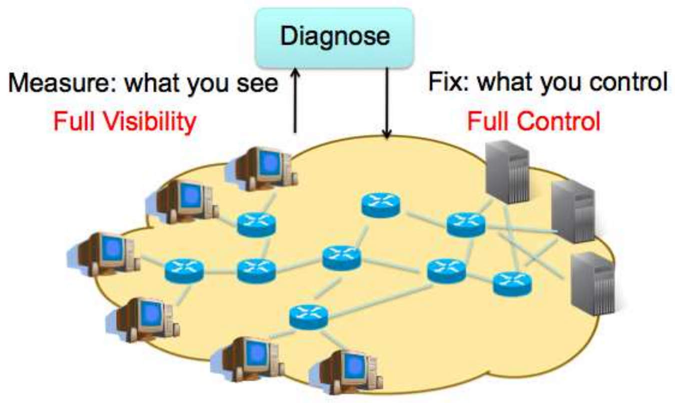


Figure 1.1: Diagnosis in the Enterprise Network

failure, network congestion, or problems at the server or clients), and finally fix it correspondingly.

For example, in an enterprise network, if the performance problem is caused by server overload (i.e., the capacity of the server could not meet the demand), then the network operators have to redirect some of the requests to alternative servers; if the performance disruption is caused by a router failure, then the network operators have to look into why the router failed (e.g., hardware failure, or software bugs). The advantage of diagnosing in an enterprise network is that the network operators have full visibility and full control of all the equipment in the network.

However, in practice, the Internet is composed of many networks and the problem is often in a network not under the control of the network operator. An IP packet has to traverse many networks from the source to reach the destination. If something goes wrong within one network, the client would complain about the performance degradation. However, the other networks might not have enough information to detect the source of the problem, or not enough control to make changes to alleviate the disruption. Thus, limited visibility and limited control makes the wide-area (not inside a specific network) diagnosis hard. In the next section, we present the challenges for minimizing performance disruptions in the wide area, and summarize contributions of our techniques.

1.2 Challenges in Minimizing Performance Disruptions in the Wide Area

Diagnosing performance disruptions in the wide area is challenging: the network operators have to collect and measure large volume and diverse kinds of data. The diagnosis procedure today is ad-hoc, and it usually takes a long time to get back to the customers. Therefore, the performance diagnosis does not scale. The goal of our work is to build systems for wide-area diagnosis. We formalize and automate the diagnosis process. We also analyze a large volume of measurement data.

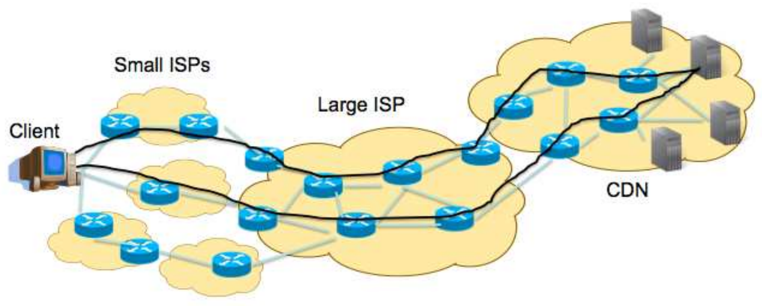
In this section, we present the challenges faced by ISPs and CDNs in minimizing performance disruptions in the wide area. Then, we summarize the problems with the inter-domain routing protocol in meeting the performance goals.

1.2.1 ISP's Challenge in Providing Good Transit for IP Packets

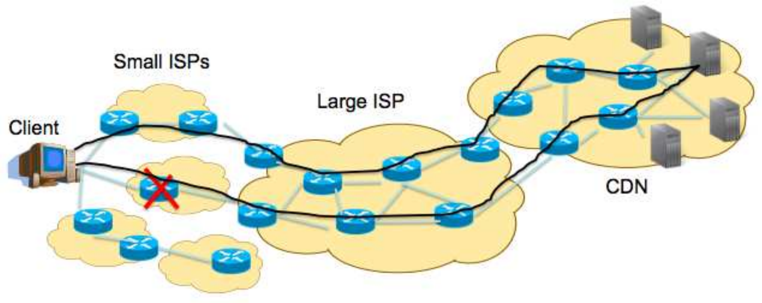
The example we use to illustrate the challenges of wide-area diagnosis is shown in Figure 1.2. In Figure 1.2(a), IP packets from the client traverse several networks (including a large ISP) to reach the server in the CDN network. The upper curved line shows the forwarding route from the client to the server, and the lower line shows the reverse route from the server back to the client. Then, a router on the reverse path fails, as illustrated by Figure 1.2(b). In Figure 1.2(c), the routers compute an alternative reverse path to deliver packets. However, this new reverse path is heavily congested, which result in performance degradation for the client.

In this situation, the client would call the local ISP to complain about the bad performance, while trying to reach the CDN server. The local ISP checks for the network conditions inside its network. Since its network is having good performance, the local ISP calls the large ISP to complain for its client. Note that the local ISP only has limited visibility, and does not know that the performance problem happens on the reverse path, which does not go through its network. Under such circumstances, the large ISP is called for diagnosing this problem, even if the disruption actually happened outside its network.

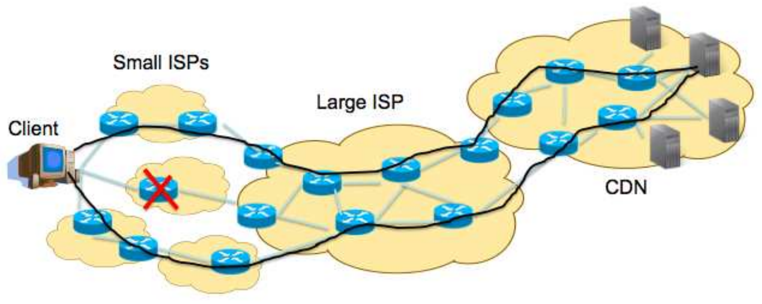
Given the IP addresses of both the client and the server, the network operators in the large ISP start diagnosing the problem by measuring the routes. By analyzing the routing updates from the routers, the network operators determine that a routing change happened on the reverse path. By measuring the latency on the reverse route to the client (this could be done by sending IP packets from the large ISP to the client



(a) Initial Routes: Forward and Reverse Paths



(b) Router Failure on the Reverse Path



(c) Reroute on the Reverse Path with Bad Performance

Figure 1.2: Performance Diagnosis Example

IP address), the network operators could tell that the performance disruption is caused by the downstream ASes on the reverse path. The network operators might choose to direct traffic along another route if available; otherwise, they have to call the downstream ASes to fix the problem.

This example shows the challenges of wide-area diagnosis, because of limited visibility and control: (i) the local ISP which gets the customer complaint does not have the visibility to measure the performance changes; and (ii) the large ISP measures the problem on the downstream path, but does not have direct ability to fix the congested link, unless it has alternate routes to alleviate the problem.

In network diagnosis, tracking routing changes is the first and the most fundamental operation in diagnosing performance problems. In fact, many network-management tasks in large backbone networks need the answer to the following question: where does a given IP packet, entering the network at a particular place and time, leave the network to continue its journey to the destination? Knowing the answer to this question could help the network operators identify which IP addresses were affected by the routing changes, and how the traffic was rerouted to reach the destinations. However, answering this question is challenging in practice, because of the scale of the routing data to analyze. Furthermore, network operators might need the answer to a large number of IP addresses immediately in response some large network events. In Chapter 2, we focus on answering this question.

1.2.2 CDN's challenge in Maximizing Performance for Services

From the CDN's perspective, the goal is to maximize network performance for the clients accessing its services. Network performance is critical to the CDNs because it affects the user experience. Good network performance could increase the number of requests, and bring more revenue to the CDN's business. Many large CDNs monitor the network latency (i.e. the round-trip time from the client to the server, and from the server back to the client) for their clients. In the example illustrated in Figure 1.2, the large CDN detects that all the clients from the same small ISP are experiencing latency increases. Although the network disruption might be outside the CDN's network, the CDN is motivated to troubleshoot the latency increases for its clients.

In the example in Figure 1.2, the CDN operators start by measuring the changes in the network: in the example, the CDN server used for serving the client requests did not change. The traffic from the client enters and leaves the CDN network at the same routers; however, the AS path from the CDN back to the client is changed, which is correlated with latency increase from the CDN to the client. In this way, the network operators could determine that performance problem happens on the downstream ASes.

Because of the limited visibility, the CDN could not figure out the exact root cause of the problem (which is caused by congestion in a small ISP). The CDN also has limited control, as this is the only available route to reach the client. In this case, the CDN could possibly alleviate the problem, by directing those clients to a server in a different location.

The example shows that minimizing user-perceived latency is challenging for CDNs. Although this is crucial, especially for CDNs hosting services like web search and video streaming, latency may increase for many reasons (e.g., CDN's server changes, or inter-domain routing changes). Moreover, the network operators need to collect multiple kinds of measurements related to performance, traffic, and routing. Joining and analyzing multiple kinds of measurement data is another challenge, especially when clients may use multiple servers or paths. In Chapter 3, we focus on automating the diagnosis for latency increases for CDN networks.

1.2.3 Rethink BGP Protocol Design

In the last two subsections, we focused on the challenges that ISPs and CDNs face today with the existing routing protocols. In this subsection, we rethink the design of the routing protocol to improve performance.

Many performance problems today are caused by the routing protocol: first, the BGP route selection (i.e., how the router chooses routes) is based on the length of the AS-level path, which is not necessarily a good indicator of performance. Second, routing changes cause lots of performance disruptions. Our study shows that 42.2% of the big latency increases in a large CDN are correlated with inter-domain routing changes. Moreover, today's inter-domain routing protocol does not support multiple paths to the same destination. Thus, the ISPs or CDNs have relatively few alternate paths, because each of its neighbors advertises at most one path.

In Chapter 4, we focus on designing the routing protocol for better performance: which satisfies the requirements of fast convergence, route selection based on performance, and scalable multi-path support.

1.3 Contributions

In this dissertation, we present two techniques: scalably tracking route changes for ISPs, and diagnosing wide-area latency increases for CDNs. We design and implement systems which are deployable for use in network management. We also design a BGP variant, and show its performance benefits: fast routing convergence, scalable multi-path support, and passive monitoring for route selection based on performance.

1.3.1 A Scalable Technique for Tracking Route Changes for ISPs

First, we propose a technique for tracking inter-domain routing changes for ISPs. We focus on answering this question from a given network's point of view: when traffic to a particular destination enters the network at some router, what inter-domain route will it use to exit the network and, beyond that, what downstream path will it take to reach the destination.

Realizing a system to track the inter-domain route changes is challenging for two reasons: first, routing updates are advertised at the granularity of prefix, which is a block of consecutive IP addresses. An IP prefix is represented in the format of an IP address, followed by the prefix length. The IP address in the prefix is the start IP address of this block of IP addresses. The prefix length is the number of shared initial bits shared by this group of IP addresses. A single IP address could be covered by multiple prefixes. The packet forwarding in such a case is governed by the longest matching prefix, which can change over time as routes are updated. Thus, in order to know the route for a particular IP address, we need to track the changes for its longest matching prefix. Second, in some network diagnosis scenarios, the operators need to know routes to hundreds of thousands of IP addresses at a given time. For such cases, our system needs to answer a large number of queries in real time.

This dissertation makes the following contributions to handle these challenges:

- To handle nested prefixes, we introduce the notion of an *address range*. An address range is a group of IP addresses that have the same set of matching prefixes. Address ranges facilitate rapid determination of route changes for an IP address. We propose an algorithm to efficiently determine changes to address ranges as well as their matching prefixes and associated routes as BGP updates arrive.

- We designed, implemented, and evaluated the tool that answers queries about routing changes on behalf of network management applications. To answer queries for a large number of IP addresses, we implement two optimizations to our system: (i) amortize the I/O cost by reading the address range records once for multiple queries; (ii) take advantage of multi-core processors prevalent today by distributing the workload across individual cores. Experiments with BGP measurement data from a large ISP backbone demonstrate that our system answers queries in real time and at scale.

1.3.2 A System for Diagnosing Wide-Area Latency Increases for CDNs

Next, we propose techniques for CDNs to classify the large latency increases, based on measurement data that can be readily and efficiently collected—measurements of traffic, routing, and performance to and from their own servers.

Finding the *root cause* of latency increases is difficult. Many factors can contribute to higher delays, and these factors are usually correlated. The *scale* of large CDNs also introduces challenges. A group of clients may send requests to *multiple* front-end servers, and the traffic may traverse *multiple* ingress and egress routers. Thus, in order to analyze the latency increases for groups of requests, we need to define the metrics to distinguish the changes from an individual router or server.

This dissertation makes the following contributions, in diagnosing wide-area latency increases:

- **Decision tree for separating cause from effect**: We identify the causal relationship among the various possible causes which lead to latency increases. Based on this causal relationship, we present a *decision tree* for separating the causes of latency changes from their effects. We use the measurement data to identify suitable thresholds to identify large latency changes, and to distinguish one possible cause from another.

- **Metrics to analyze routing and traffic data over *sets* of servers and routers:** We are motivated to analyze latency increases and traffic shifts over *sets* of servers and routers. For all potential causes of the latency increase in the decision tree, we propose metrics to quantify the proportion of the latency increases contributed by each possible cause. Furthermore, we define the metric to quantify the proportion of latency increases contributed by a single router or server, as well as a way to summarize the contributions across all routers and servers.

- **Latency characterization for Google's CDN:** We apply our methodology to traffic, performance, and routing data from Google's CDN. Our results show that about three quarters of these large increases in latency were explained (at least in part) by a large increase in latency to reach an existing front-end server; around one third of the large increases of latency involved a significant shift of client traffic to different front-end servers. We also present several events as case studies to highlight the challenges of managing wide-area performance for CDN networks.

1.3.3 A Routing Protocol Designed for Better Performance

Finally, we propose a variant of the BGP routing protocol called "next-hop BGP". The key idea is to move from today's *path-based routing* (where routing decisions depend on the entire AS path) to *next-hop routing*—a solution that selects and exports routes based only on the neighboring domain. This dissertation makes the following contributions, which improves the network performance of the inter-domain routing protocol:

- **A protocol with fast convergence to reduce transit disruptions:** We show, both analytically and experimentally, that next-hop routing leads to much fewer update messages. Our simulations show that next-hop routing is especially effective at preventing serious convergence problems.

- **Selection of the next-hop based on performance, not shortest AS path:** Our protocol offers an efficient way for selecting routes based on the performance. In addition, we present the design of passive monitoring schemes for ISPs, CDNs, and stub networks to measure the performance of routes through different next-hop ASes, so that they could select routes based on the performance.

- **Scalable multi-path support to avoid congestion and failure:** Next-hop BGP also lowers the barrier to multi-path routing. Hence, it brings us closer to achieving the benefits that come with multi-path routing (e.g., improved availability, better recovery from failures, load balancing, customized routes selection, and more).

- **Less complexity, ease of configuration and fewer errors:** We explain how next-hop routing can be implemented in a backwards-compatible manner. We also discuss how it can simplify router implementation and configuration, and thus minimize software complexity for vendors, require less training for network operators, and lead to fewer configuration errors.

Collectively, the contributions of this dissertation provide effective systems solutions for ISPs and CDNs to automatically diagnose network performance disruptions in the wide area. We also propose a transition from today's path-based routing to next-hop routing, and our results show that next-hop routing leads to significantly better network performance than path-based routing with BGP. Chapter 2 and 3 describes the case studies of network diagnosis for an ISP and a CDN in detail. Chapter 4 presents the improved design of BGP, followed by the conclusion in Chapter 5.

Chapter 2

Route Oracle: Tracking Route Changes Scalably for ISPs

2.1 Introduction

Many network-management problems in large backbone networks need answer to a seemingly simple question: from a given network's point of view, when traffic to a given destination enters the network at some router, what router it will use to exit the network and, beyond that, what downstream path it will take to reach the destination. Answering this question at scale and in real time is challenging. Yet, knowing the answer to this seemingly simple question is crucial for performing a wide range of network management tasks. We describe our system called *Route Oracle* that answers this question at scale and in real time.

The Internet is divided into thousands of ASes, and BGP allows ASes to exchange information about how to reach external destinations. A BGP route contains many attributes including these key ones: the egress router (i.e., the last router before packets leaves a given AS) and the AS path (i.e., the list of ASes on the downstream path to the destination. Knowing the egress router and AS path for an individual IP address, both currently and at a given time in the past, is crucial for several management tasks such as troubleshooting reachability problems in response to customer complaints. While knowing the egress router and AS path already provides vital information, joining this data with other information could form the basis for even more powerful applications. For example, joining the routing data from the intra-domain routing protocol could provide the path traversed by traffic from ingress to egress routers in an AS. Similarly, joining with the performance measurements for an application would allow network operators to correlate performance impairments with route changes in the network.

Realizing a system to track BGP route changes is challenging for two reasons:

- In BGP, routing updates are advertised in terms of prefixes, and prefixes can be nested. Thus a single IP address could be covered by multiple prefixes. The packet forwarding in such cases is determined by the longest matching prefix, which can change over time as routes are advertised and withdrawn. Thus, in order to know the route for a particular IP address, we need to track the changes to its longest matching prefix, and this needs to be done at the rate of BGP updates, especially for applications that need to know routes in real time.

- Some applications need to know routes to hundreds of thousands of IP addresses at a given time. An example is a CDN serving a large number of clients that wants to know the routing changes for all its clients to troubleshoot latency increases. For such applications, our system needs to answer a large number of queries in real time.

This chapter overcomes these challenges by making the following contributions:

- **Characterization of Prefix Nesting and Prefix-Match Changes:** We analyze the effects of BGP routing changes on the longest-matching prefix. We find that more than 30% of BGP updates do *not* simply switch an existing prefix from one route to another. In fact, 14.8% of BGP updates cause addresses to gain or lose reachability, and 13.2% of updates cause addresses to switch to a different longest-matching prefix. These prefix-match changes have a variety of causes, including route flapping, sub-prefix hijacking, and failover to backup routes.

- **The Concept of Address Range to Track Routes Scalably for IP Addresses:** We introduce the concept of an *address range*. An address range is a group of consecutive IP addresses that share the same group of matching prefixes. We design the algorithm to efficiently determine changes to address ranges as well as their matching prefixes and associated routes as BGP updates arrive. The resulting data structure facilitates rapid determination of route changes for an IP address – we just need to look for the appropriate address ranges instead of prefixes.

- **Route Oracle Tool with Performance Evaluation:** We present the design and implementation of the Route Oracle tool. To answer queries for a large number of IP addresses, we implement several optimizations to Route Oracle: (i) we archive the pre-processed address range records to speed up query processing, (ii) read address range records once for multiple queries to amortize the I/O cost, and (iii) take advantage of multi-core processors prevalent today by distributing the workload across individual cores. Performance evaluation with BGP measurement data from a large ISP backbone illustrate that our system answers queries in real time and at scale.

We have deployed Route Oracle in a large Tier-1 ISP where it is being used to troubleshoot performance impairments for various services. In fact, the optimizations described above were implemented after the initial trial which revealed that the users were interested in using the system for hundreds of thousands of IP addresses in real time.

The rest of the chapter is organized as follows. In Section 2.2, we further motivate Route Oracle by describing three applications of the tool. Next, we describe the challenges of building Route Oracle, by characterizing prefix nesting and the dynamics of prefix-match changes in Section 2.3. Then, we present the algorithm to efficiently track prefix and route changes for address ranges in Section 2.4, followed by the overall design and implementation of Route Oracle in Section 2.5. In Section 2.6, we evaluate the performance of Route Oracle. We present related work in Section 2.7, and conclude the chapter in Section 2.8.

2.2 Motivating Applications

In this section, we present three possible applications of Route Oracle. We show that determining the egress router and AS path to external destinations is an important building block for many network management tasks. For each application, we present the challenges of the application, and how Route Oracle could effectively handle it. We also explain why each application needs to track routing changes at scale and in real time.

2.2.1 Historical Traceroute

Traceroute is a popular active measurement tool, which lists the hops along the path to the destination. It is extensively used by the network operators to troubleshoot reachability or performance problems. Traceroute is implemented by sending IP packets in real time. Therefore, if traceroute result was not collected in the past, it would be impossible to roll back to a time in the past to see the path at that time.

However, having a "historical traceroute" that provides paths from any vantage point to a given destination at any time in the past would be valuable for troubleshooting events. The straightforward approach to implement such historical traceroute is to collect and archive the traceroute results. However, collecting traceroute data from many vantage points to all the destination IP addresses is too expensive due to large probing and storage requirements. Instead, we propose to leverage the passively collected routing updates: from the BGP updates, we could determine the egress router and AS path the packets traverse; and from the routing messages of the intra-domain routing protocol like OSPF, we could determine the hop-by-hop path (from the vantage point to the egress router) inside the ISP. Our Route Oracle tool provides the former capability.

2.2.2 Analyzing External Routing Disruptions

One of the most important tasks of network operations is to react quickly to large network disruptions. Disruptions like an under-sea cable cut could result in loss of reachability or serious congestion in a large portion of a geographical region. Characterization of these events could help network operators understand the impact on their networks: what percent of the IP addresses in the affected geographical region became unreachable? How was the traffic rerouted to reach the destination, and did it cause any traffic shifts inside the network? Did the packets actually reach the destination, and were there any performance degradations? Knowing answers to these questions would help the network operators make decisions on how to respond to the event, including redirecting the traffic through a better route, or performing traffic engineering to direct traffic away from congested links.

Answering these questions requires determining the routes or route changes of all the IP addresses in the affected geographical regions. Route changes for these IP addresses should not only be processed at scale, but also in real time, in order to get the latest status of the event. Based on these results, aggregated statistics like the percent of routable IP addresses could indicate the overall impact on reachability. Aggregating the routes across egress routers and nexthop ASes could tell how the traffic shifted inside the network. Joint analysis with the traffic and performance data could help operators monitor performance along each rerouted path.

2.2.3 Service-Level Performance Management

With the deployment of many network applications, ISPs are facing the challenge of *service quality management*. Performance monitoring at the application layer alone is not enough to troubleshoot performance degradation of network services. Instead, network operators need to correlate the performance problems with network-layer changes (such as routing changes, or network congestion) for root-cause analysis.

As an example, consider an ISP hosting a Content Distribution Network (CDN). Such an ISP would be interested in monitoring the performance of its CDN, by measuring the round-trip time (RTT) to the clients. A simple approach is to monitor the performance of content download for each client. However, the RTT of individual clients is likely to vary significantly. Instead, if we could aggregate the RTTs over clients using the same paths (e.g., clients leaving the network at the same egress router), then large changes in the average RTT would show the performance changes caused by rerouting or congestion inside the network. In this case, we need the capability to determine the routes for all the clients' IP addresses at scale and in real time. Furthermore, this real-time performance information could be used to guide the selection of the server that should serve each client. [1]

To summarize the requirements from these applications, our system needs to track the changes of the egress router and AS path: (i) at scale: for thousands of IP address (e.g., all the IP addresses from the same country, or ISP), (ii) and in interactive mode so that the network operators can get the answer quickly and take actions based on the results.

2.3 Prefix Nesting and Prefix-Match Changes

In this section, we present the background of prefix nesting, with static analysis on the extent of prefix nesting. We also characterize the dynamics of the prefix-match changes, and correlate with the traffic data to understand its impact on the IP reachability. The goal of this section is to motivate the data structure and algorithm we use in the Route Oracle tool.

In the routing table, an IP address could be covered by multiple prefixes. For example, IP address 128.112.0.0 could be covered by both 128.112.0.0/16 and 128.112.0.0/24. When the IP packets destined to 128.112.0.0 are forwarded, routers perform the longest prefix match (LPM), and use the route of prefix

[1]Note that for the CDN case study in the next chapter, the data we get has already been aggregated at the prefix level. Thus, we do not have to solve the problem on determing routes for individual IP addresses and grouping the client IPs according to the routes (e.g., by egress router or AS path).

128.112.0.0/24 to deliver the packets to the destination. We analyzed a BGP routing table collected from a router in a large ISP on February 1, 2009. The result showed that 24.2% of the IP addresses were covered by multiple prefixes.

Nesting of prefixes is quite common for a variety of reasons. First, regional Internet registries allocate large address blocks to Internet Service Providers (ISPs), who in turn allocate smaller blocks to their customers. Second, customers that connect to the Internet at multiple locations may further sub-divide these address blocks to exert fine-grained control over load balancing and backup routes. Third, ISPs may also announce multiple blocks to protect themselves from route hijacking—for example, AT&T announces prefixes 12.0.0.0/9 and 12.128.0.0/9, in addition to the 12.0.0.0/8 supernet, to prevent other ASes from accidentally hijacking traffic intended for destinations in 12.0.0.0/8.

As a router receives more and more BGP updates, the longest matching prefix for an IP address may change over time. In [3], we used BGP update messages collected for the month of February 2009 from the router in the tier-1 ISP backbone, and determined the frequency of BGP updates that affected the longest-matching prefix for IP addresses. Our analysis revealed that 13% of the BGP updates caused some IP addresses to change their longest-matching prefix. Because of frequent changes in longest prefix match, tracking the prefix match changes efficiently is necessary.

In this section, we begin by characterizing the phenomena of prefix nesting, and the dynamics of how the BGP updates affect the longest-matching prefix for IP addresses. Then, we study four main categories of prefix-match changes, based on the origin ASes (i.e., the AS that first announces the prefix into BGP) of the two prefixes and how often the more-specific prefix is available.

2.3.1 Static Analysis of Prefix Nesting

To understand the nesting of prefixes, we analyze a BGP routing table collected from a router in a large ISP on February 1, 2009. We ignore small prefixes (with mask longer than /24) corresponding to the ISP's own routers and links, as they are not externally visible. We characterize prefix nesting from two perspectives: (i) how many prefixes cover each IP address? and (ii) what fraction of addresses covered by a prefix use that prefix for packet forwarding?

The light bars in Figure 2.1 plot the distribution of the number of prefixes covering each IP address, with a logarithmic scale on the y-axis. While 75.8% of IP addresses are covered by a single prefix, 19.7% are covered by two prefixes, and 4.0% by three prefixes; some addresses are covered by as many as *seven*

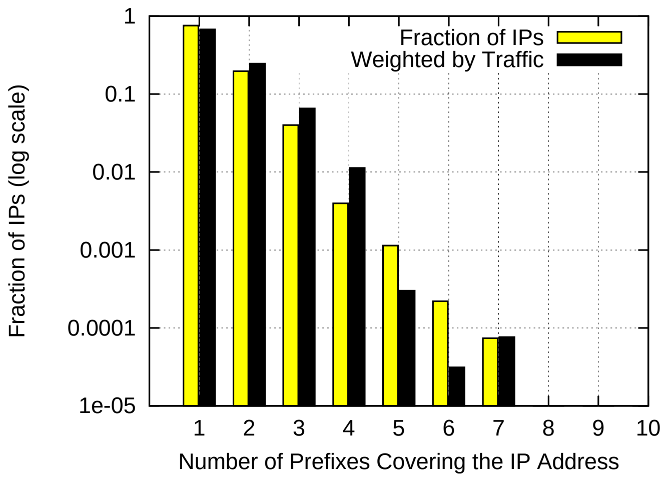


Figure 2.1: Distribution of number of matching prefixes (from a BGP routing table on Feb 01, 2009 00:00:00 GMT)

prefixes. In addition, destination addresses that match multiple prefixes are responsible for a higher fraction of the traffic, relative to other destinations, as seen by the dark bars in Figure 2.1. This bars plots the distribution weighted by the volume of traffic, as computed from the Netflow traces. While 61.6% of the traffic is destined to addresses matching a single prefix, 31.3% of the traffic corresponds to two prefixes, and 6.0% to three prefixes. We see similar trends for both histograms across a variety of routers and time periods for data collected in the same ISP.

We also explore what fraction of the IP addresses covered by a prefix use that prefix for packet forwarding. We use the same routing table snapshot for this analysis, which was taken on Feb 01, 2009. Table 2.1 shows the results for five sets of prefixes, grouped by mask length. Interestingly, 17% of the /8 prefixes are not the longest-matching prefix for *any* of the addresses they cover; the 12.0.0.0/8 prefix mentioned in Section 4.1 is one example. In fact, 39% of the /8 prefixes handle forwarding for less than half of their addresses, as seen by summing the first three rows of the "/8" column in Table 2.1. For smaller prefixes (with larger mask lengths), the prefixes are responsible for a larger fraction of the IP addresses they contain. Because we filtered the prefixes with mask length larger than 24 for this analysis, the /24 prefixes are the longest-matching prefix for all of their IP addresses. We saw similar results when analyzing a routing-table snapshot taken on Mar 01, 2009.

The nesting of prefixes suggests that BGP update messages may change which prefix is used to forward traffic to particular destination addresses. In the following sections, we track the evolution of the longest-

Fraction of	**Prefix Mask Lengths**				
IP Addresses	**/8**	**/12**	**/16**	**/20**	**/24**
0	0.17	0.16	0.09	0.04	0.00
(0, 0.25]	0.13	0.14	0.02	0.02	0.00
(0.25, 0.5]	0.09	0.06	0.03	0.03	0.00
(0.5, 0.75]	0.09	0.06	0.03	0.04	0.00
(0.75, 0.9]	0.13	0.05	0.03	0.07	0.00
(0.9, 1]	0.39	0.53	0.81	0.80	1.00

Table 2.1: Prefix coverage for different mask lengths (from a BGP routing table on Feb 01, 2009 00:00:00 GMT)

matching prefix to understand when and how BGP routing changes affect the forwarding of IP packets.

2.3.2 Frequency of Prefix-Match Changes

The BGP update messages from a top-level route reflector give us a view of BGP routing changes seen at a large Point-of-Presence (PoP) in the ISP backbone. We start by reading a BGP table snapshot taken on Feb 01 2009, followed by the stream of BGP update messages from Feb 01 to Feb 28, 2009. We filter duplicate update messages, including those sent after resets of our monitoring session [4] to the route reflector. We find four main categories of BGP update messages, as summarized in Table 2.2:

Updating a route for an existing prefix: Just under 70% of the update messages are announcements that merely change the route for an existing IP prefix, as indicated by the first row of the table. These update messages do not affect the longest-matching prefix used for forwarding data packets.

Gaining or losing reachability: Another 14.8% of messages either add or remove the only prefix that covers some range of IP addresses. Half are withdrawal messages that leave these addresses with *no* matching prefix, and the other half are announcements that allow these addresses to go back to having a matching prefix.

Changing the longest-matching prefix: Another 13.2% of messages cause some addresses to change to a different longest-matching prefix. Half are withdrawal messages that force these addresses to match a less-specific prefix, and the other half are announcements that allow these addresses to match a more-specific prefix.

Affecting a prefix that is not used for forwarding: The remaining 2.5% of update messages either add or remove a prefix that is not the longest-matching prefix for *any* IP addresses[2]. These prefixes are

[2]In this category, we see more announcements than withdrawals—a seemingly odd phenomenon we intend to investigate further.

Category	% Updates
Same prefix, route change	69.4%
Gain reachability	7.4%
Lose reachability	7.4%
More-specific prefix	6.6%
Less-specific prefix	6.6%
No impact announcements	2.3%
No impact withdrawals	0.2%

Table 2.2: Classification of BGP update messages

supernets like 12.0.0.0/8 that corresponded to address space that is completely covered by more-specific prefixes like 12.0.0.0/9 and 12.128.0.0/9.

Analysis of BGP update messages for a different time period (namely, March 2009) leads to very similar results. In the rest of this section, we focus on the 13.2% of BGP update messages that cause prefix-match changes.

2.3.3 Characterization of Prefix Match Changes

To analyze the prefix-match changes, we first account for the effects of route flapping, where a prefix is repeatedly announced and withdrawn for a long period of time. As in previous work [5], we group update messages for the same prefix that occur with an inter-arrival time of less than 70 seconds, assuming these updates are part of the same BGP convergence event. Since most convergence events last less than 600 seconds (five minutes) [5], we assume longer events correspond to persistent flapping, and remove these unstable prefixes from further analysis. This step filtered 117 prefixes and about 2.7% of the prefix-match changes. This leaves us with 1,259,532 prefix-match changes for the month of February 2009 for further analysis.

Then, we begin to characterize the prefix match changes. Recall that an address range is a group of consecutive IP addresses that share the same group of matching prefixes. Looking at the pre-processed measurement data, we notice that most address range have a single prefix that serves as the longest-matching prefix the vast majority of the time. In fact, 95.2% of the address ranges have a prefix they use more than 90% of the time, and 98.7% have a prefix they use more than 60% of the time. We apply a threshold of 60% to identify the *dominant* prefix for each address range, and analyze the prefix-match changes that cause an

We suspect that, over time, some ASes introduce additional supernet routes as part of configuring backup routes.

Origin ASes	Prefix Match	#Events	Possible Explanations
Same	More-specific	87,996 (13.0%)	Route leak
Same	Less-specific	395,755 (58.5%)	Load balancing, failover to backup route
Different	Less-specific	172,260 (25.5%)	Customer failure
Different	More-specific	20,186 (3.0%)	Sub-prefix hijacking, announcement of new customer route

Table 2.3: Four classes of prefix-match events and their possible causes

address range to *stop* using its dominant prefix. This leaves us with 676,197 prefix-match changes to analyze. For some address ranges, these events involve the brief announcement (and subsequent withdrawal) of a more-specific prefix; for others, these events involve the brief withdrawal of the dominant prefix and the temporary use of a less-specific route. As such, we classify prefix-match changes in terms of whether the dominant route is more-specific or less-specific than the other (briefly used) prefix. To understand the possible reasons for the prefix-match changes, we also compare the origin ASes of the old and new prefixes. This leaves us with four cases, as summarized in Table 2.3. Note that the more-specific and less-specific prefix match mentioned in the table are the briefly used prefixes.

Same origin AS, more-specific prefix: About 13.0% of the prefix-match changes involve brief announcement of a more-specific prefix with the same origin AS as the dominant prefix. We suspect these prefix-match changes are caused by temporary route leaks, where the more-specific prefix is announced inadvertently due to a configuration mistake that is fixed relatively quickly (e.g., within a few hours or at most a day or two).

Same origin AS, less-specific prefix: About 58.5% of the prefix-match changes involve brief withdrawal of the dominant prefix that leads to the temporary use of a less-specific route with the same origin AS. We suspect that these prefix-match changes are caused by multi-homed ASes that announce both prefixes for fine-grain control over load balancing and backup routes. For example, a multi-homed stub AS connected to two providers may announce 15.0.0.0/17 to one provider and 15.0.128.0/17 to the other, and the supernet 15.0.0.0/16 to both. The more-specific 15.0.0.0/17 prefix would be withdrawn whenever the link to the first provider fails, and the less-specific 15.0.0.0/16 would remain because the route is also announced via the second provider.

Different origin ASes, less-specific prefix: About 25.5% of the prefix-match changes involve brief withdrawal of the dominant prefix that leads to the temporary use of a less-specific route with a different origin AS. We suspect that these prefix-match changes occur when a customer AS fails, but its provider does not. For example, suppose a provider that announces 12.0.0.0/8 has allocated 12.1.1.0/24 to one of its customers. If the customer fails, the customer's route for 12.1.1.0/24 is withdrawn, while the provider's 12.0.0.0/8 route remains.

Different origin ASes, more-specific prefix: Only 3.0% of the prefix-match changes involve the brief announcement of a more-specific prefix from a different origin AS. We suspect some of these announcements correspond to "sub-prefix hijacking" caused by a configuration mistake or a malicious attack. For example, during the infamous hijacking of YouTube in February 2008 [6], Pakistan Telecom mistakenly announced 208.65.153.0/24, a subnet of YouTube's 208.65.152.0/22 address block. Another cause could be an ISP that inadvertently misconfigures a route filter that is supposed to block small address blocks announced by one of its customer ASes.

2.4 Tracking Prefix Match Changes

In this section, we present how to determine routes (more specifically egress router and AS path) to destinations given IP addresses, ingress router (where the IP packets enter the network), and time. We use the BGP routes from the specified vantage point and time period as inputs for the algorithm. Since an IP address could be covered by multiple prefixes, the key problem is to track the longest prefix match and the associated routes for the IP addresses of interest as BGP updates are received.

We begin by exploring the most obvious design option of using the forwarding table of IP routers to track longest-prefix match (LPM) changes. Then, we point out the scaling challenge of this approach, and introduce the notion of an *address range* which is a group of IP addresses that have the same set of matching prefixes, to track routing changes efficiently. Last, we present the algorithm to track prefix match changes using address ranges, and demonstrate its benefits by run-time analysis. We also point out that the using address ranges facilitates parallelization of the algorithm.

2.4.1 Tracking Prefix Match Changes by the Forwarding Table

The most obvious way to implement the function of tracking LPM changes is to use the forwarding table of the IP routers. This could be done by running the router software to compute the routes and update the forwarding table. As the queries come in, the forwarding table would be able to output the results based on the longest prefix match.

Assume we have n prefixes initially in the routing table at time $t1$, and m updates from time $t1$ to $t2$. If the query is for *one* particular IP address from time $t1$ to $t2$, then the time complexity to get the answer is $O(n + m)$. This is because we need to process all the initial routes and updates. The space complexity is p, where p stands for the number of prefixes covering the query IP address. This is because the LPM might change, and we need to keep track of all the prefixes covering this IP address. Since there are no more than 32 prefixes covering one IP address, the space complexity is actually $O(1)$.

Suppose the queries are for *many* (say k) IP addresses from time $t1$ to $t2$. Then, instead of tracking the related prefixes, we need to keep track of all the prefixes in the forwarding table. The space complexity is $O(n)$, which includes all the routes. The forwarding table of routers uses the data structure of the Patricia Trie, which is similar to the tree, so that it takes $log(n)$ time to access and update each elements. As for the time complexity, it takes $O(n * log(n))$ to initialize the data structure. For each route, we also need to check if it affects the query IP address, so it takes $O(n * k)$ totally for k queries. Similarly for each of the updates, it takes $O(log(n))$ to update the data structure, and it takes $O(k)$ to check for the queries. Therefore, the total time complexity (including initialization and processing of the updates) is $O(n*(log(n)+k))+O(m*(log(n)+k))$, which is $O((n+m)*(log(n)+k))$. In practice, the number of routes in the routing table (which is n in the formula) is on the order of 300,000, the number of updates (m) is on the order of millions, and the number of queries could be hundreds of thousands. Thus, the query processing time could be large.

Since our goal is to reduce the query time, and this could be done by trading off more space for less time. We could achieve this by pre-processing the routes and updates, and storing the pre-processed results. Suppose we have one IP address in the query, and it takes $O(n + m)$ to pre-compute the LPM for this particular IP address. The storage space for the results is $O(s)$, where s represents the number of routes and updates which affects the LPM of the query IP address. In this way, the actual query processing time is reduced to $O(s)$, by simply outputting the results. However, this approach would not scale if we need to store the results for all IP address, since the total number of IP addresses is 2^{32}. In the next subsection, we introduce the idea of address range to track the LPM changes scalably.

2.4.2 Address Ranges and the Associated Data Structure

In this section, we present a method to track prefix match changes for a group of IP addresses. Because of the nesting of prefixes, an IP address could match several prefixes with different mask lengths over time. In

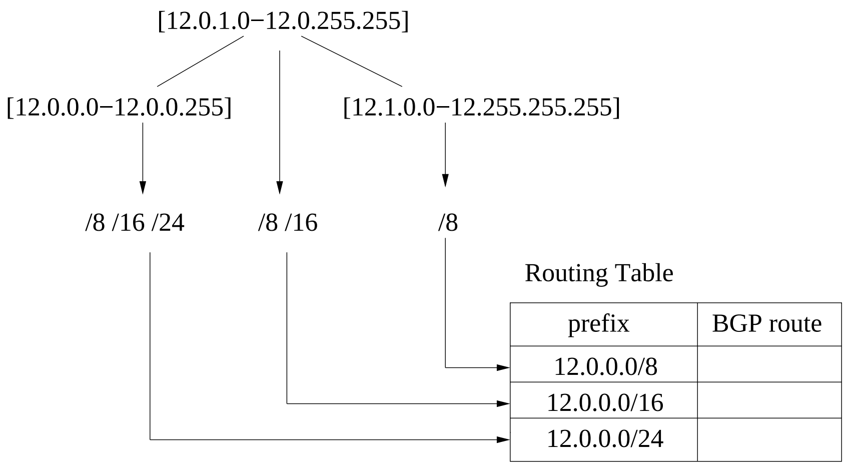


Figure 2.2: Storing address ranges and prefix sets for prefixes 12/8, 12/16, and 12/24

order to track prefix-match changes over time, we need to store information about changes to all prefixes covering the IP address. We refer to the collection of all matching prefixes for a given IP address as its *prefix set*; packet forwarding is driven by the longest-matching prefix in the set at any time. For example, suppose a BGP routing table contains prefixes 12.0.0.0/8 and 12.0.0.0/16. Then, IP address 12.0.0.0 has the prefix set {/8, /16}. IP address 12.0.0.1 also matches the same prefixes. However, the prefix set for 12.1.0.1 is {/8}.

Rather than tracking the prefix set for each individual IP address, we group contiguous addresses that have the same prefix set into an *address range*. For example, prefixes 12.0.0.0/8 and 12.0.0.0/16 divide the IP address space into two address ranges—[12.0.0.0, 12.0.255.255] with prefix set {/8, /16} and [12.1.0.0, 12.255.255.255] with prefix set {/8}. Note that address ranges differ from prefixes in that the boundaries of an address range are not necessarily powers of two. For instance, no single prefix could represent all IP addresses in the range [12.1.0.0, 12.255.255.255].

As we process BGP update messages, address ranges may be created, subdivided or updated. For ease of searching for the affected address range(s), we store information about address ranges in a binary tree, as shown in Figure 2.2. A binary tree efficiently supports all the operations we need (including inserting a new address range, lookup an address range) in an average time of $O(log\, n)$, where n is the number of address ranges. In comparison, the brute-force solution that simply stores address ranges in arrays would operate in an average time of $O(n)$. Each node in the binary tree contains the left-most address in the address range, and each node keeps a pointer to the size of the address range and the associated prefix set. Each element of the prefix set includes a pointer to the BGP route for that prefix; to save memory, we store a single copy of each BGP route. As illustrated in Figure 2.2, both address ranges [12.0.0.0, 12.0.0.255] and [12.0.1.0, 12.0.255.255] have prefix 12.0.0.0/16 in their prefix sets, and their prefix sets store the pointers to the route entry for 12.0.0.0/16. Note that in the figure, we only show the pointers from the most-specific prefixes to the routing table for illustration.

2.4.3 Algorithm for Tracking Changes to Address Ranges

Next, we present an algorithm that reads BGP table dumps or update messages as input, and tracks the changes to the address ranges and their associated prefix sets. The algorithm first determines the address range(s) covered by the prefix, perhaps creating new address ranges or subdividing existing ones. Then, for each of the associated address ranges, the algorithm modifies the prefix set as needed.

Updating address ranges: A BGP announcement for a new prefix may require creating new address ranges or subdividing existing ones. For example, suppose 18.0.0.0/16 is announced for the first time, and no earlier announcements covered any part of the 18.0.0.0/16 address space; then, our algorithm inserts a new address range [18.0.0.0,18.0.255.255], with a prefix set of $\{/16\}$, into the binary tree. As another example, suppose we have previously seen route announcements only for 12.0.0.0/8 and 12.0.0.0/16; then, the binary tree would contain [12.0.0.0,12.0.255.255] with prefix set $\{/8, /16\}$, and [12.1.0.0,12.255.255.255] with prefix set $\{/8\}$. On processing an announcement for 12.0.0.0/24, our algorithm would subdivide [12.0.0.0,12.0.255.255] into two address ranges—one with prefix set $\{/8, /16, /24\}$ and another with $\{/8, /16\}$, as shown in Figure 2.2. Currently, our algorithm does not delete or merge address ranges after withdrawal messages. We take this lazy approach towards deleting and merging address ranges because withdrawn prefixes are often announced again later, and because we have seen empirically that the number of address ranges increases very slowly over time.

Updating prefix set for address ranges: Continuing with the example in Figure 2.2, suppose the route for 12.0.0.0/16 is withdrawn. Then, the algorithm would determine that both [12.0.0.0-12.0.0.255] and [12.0.1.0,12.0.255.255] have $/16$ removed from the prefix set. For addresses in [12.0.1.0-12.0.255.255], the withdrawal would change the longest matching prefix to the less specific 12.0.0.0/8.

Lastly, we estimate the time and space complexity for this algorithm. The pre-processing time for n routes and m updates from time $t1$ to $t2$ is $n * log(n) + m * log(n) = (n + m) * log(n)$. This is because it takes $O(log(n))$ to perform a single operation on the tree. The storage space for the results of LPM changes is $O(n+m)$, since we need to track all routes and updates at most. As for the actual query processing, it only takes the time of $O(k*(n+m))$, by looking up the results for each query IP address.

Another advantage of the address ranges is that it breaks the dependency among the prefixes. By having the pre-processing records of address ranges, we no longer need to process the updates by time consequentially for the queries; instead, we could parallelize the processing for all the result entries for all the queries. Therefore, the query time could be further reduced to $O(k*(n+m)/c)$, where c stands for the number of CPUs available for parallel processing.

In summary, in this section, we present the idea of using address ranges to track the LPM changes for IP addresses at scale. We also point out another benefit of the address ranges, which is to remove the dependency among the prefixes and facilitate parallelization.

2.5 Route Oracle Design and Implementation

Using the algorithm of Section 2.4 as a basis, we present the overall design and implementation of Route Oracle in this section. We begin with the overview of the design, and follow it with a detailed description of the two main parts of the design: the pre-processing module that converts BGP updates to updates of address range records, and the query module that identifies the routes for IP addresses based on these records.

2.5.1 System Overview

The input to Route Oracle consists of a list of IP addresses, a vantage-point router, and the time period over which route changes are desired. The output is routes for each IP address at the start time, and then changes to them during the specified time period. The route includes the longest matching prefix, egress router and the AS path. The basic way to handle such a query is to process the BGP updates from the specified vantage point and time period using the algorithm we presented in the last section, and output the result. For example, for the query of route changes for IP address 128.112.0.0 from t1 to t2 at vantage point v, the algorithm will have to process BGP updates from vantage point v from t1 to t2. The output could be: at time t1, the longest prefix match is /24, with associated route r1; at time t3 (which is before t2), the IPs change to a less-specific prefix match /16, with route r2; and finally no route changes for the address till t2. However, this basic approach does not scale well with the number of IP addresses due to nesting of

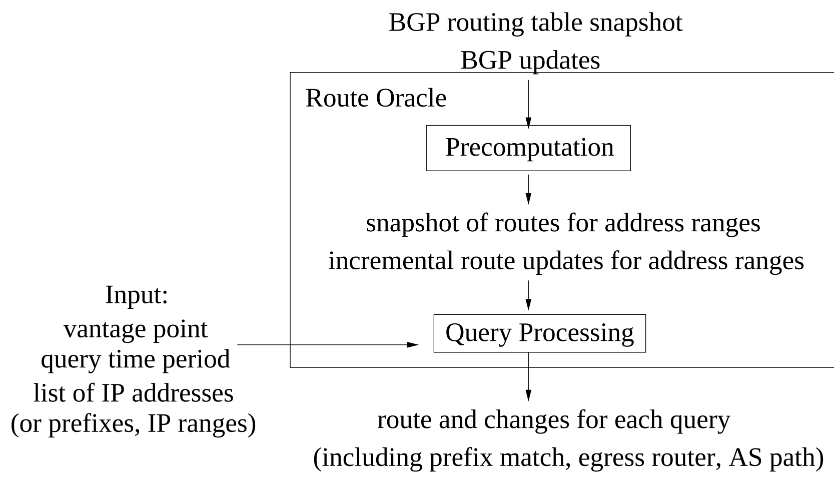

BGP routing table snapshot
BGP updates
Route Oracle
Precomputation
snapshot of routes for address ranges
incremental route updates for address ranges
Input:
vantage point
query time period
list of IP addresses
(or prefixes, IP ranges)
Query Processing
route and changes for each query
(including prefix match, egress router, AS path)

Figure 2.3: Route Oracle System Design

prefixes. The nesting of prefixes means that for each IP address, every update containing a matching prefix needs to be processed to see if the longest prefix match, and hence the route, changed for the address or not.

To overcome this, we convert prefix-based BGP updates to routing changes for address ranges by using the algorithm specified in Section 2.4. Since address ranges do not overlap, answering queries for route changes scales much better for a large number of addresses in the input. In fact, once we convert BGP updates into address ranges, we save the resulting records for future use, and answer all queries using these records, thereby amortizing the cost of conversion across all subsequent queries.

To summarize, our design of Route Oracle consists of two modules as shown in Figure 2.3: A pre-processing module that transforms the BGP update stream into a stream of address ranges and associated route changes; and a query module that determines route changes for desired IP addresses using the address range records. In the following subsections, we provide detailed descriptions of these two modules.

2.5.2 Precomputing Route Updates for Address Ranges

The pre-processing part uses the algorithm presented in the previous section to track the longest prefix match and route changes for all the IP addresses. The inputs to the pre-processing part are BGP routing table snapshots and BGP updates. The output consists of address ranges and route changes to them. Each record contains the following attributes: the address range, time of the route change, the longest prefix match, the egress router, and the AS path. In addition, we also include information on how the longest prefix match was changed compared with the previous route (e.g., change to a more-specific or less-specific prefix), and whether the egress router and AS path were changed compared to the previous route.

Since BGP is an incremental protocol where updates are only sent by a router when its route to a prefix changes, the BGP monitor deployed in the tier-1 ISP generates periodic snapshot of routes for all prefixes. Using the snapshot of prefixes and their routes, the module determines and stores a snapshot of all the address ranges and routes associated with them. It then records changes to address ranges and routes as subsequent BGP updates are received.

We should note that the time interval between snapshots of all address range records has a direct bearing on query processing time since, given a query, we have to process address ranges starting from the latest snapshot before the query time period. Thus, storing snapshots more frequently will reduce query time. However, this will increase the storage space, and hence there is a tradeoff between how often snapshots are stored versus query processing time. At present, we store one snapshot per day to coincide with the BGP routing tables which are generated at the beginning of every day.

We store all the address range records for a given day under the same directory, with the file names indicating the IP address of the vantage point. Directories are named hierarchically according to year, month and day. This facilitates easy searching of files for a given time period. The address range updates are stored in multiple files with each file spanning a fixed interval (currently 15 minutes) of the day. The files are compressed to reduce the storage space. All the records are written in network byte-order for platform compatibility.

2.5.3 Query Processing

The query module uses the address range records (snapshots + updates) to answer queries regarding routes for IP addresses. The input to the query module is a list of IP addresses, the vantage point and the time interval over which routes are desired. The output is routes at the beginning of the time interval for each address range and changes to them over the interval. The input IP addresses can be specified as address ranges or prefixes.

When a query is received, the module first determines the latest address range snapshot prior to the start time of the interval. It then applies updates of address range records to determine the state at the start time, and then continues processing address range records till the end time to determine route changes. For each address range, the module checks if the range overlaps with the input IP address list, and if so, the record is used to update the route for the affected IP addresses. Note that since the address range files are stored hierarchically based on their time stamps, the module can quickly locate files for a specific time interval. For instance, let us assume that the query is for route changes of a single IP address 128.112.0.0 from October 24, 2009 9:00 a.m. to 5:00 p.m.. In this case, the query module will read the address range snapshot at the beginning of the day on October 24, 2009, and then address range updates till 9:00 a.m. to get the latest route for 128.112.0.0 just at the start time of the query time period. Then, the module will read the address range updates from 9:00 a.m. to 5:00 p.m., and output all route changes for the IP address in question.

We have implemented two optimizations to further reduce the query response time. First, if the user inputs a list of IP addresses in a query, we amortize the cost of reading address range records across all the IP addresses. This is because the reading of the address range files dominates the overall query processing time. As we read address ranges sequentially, we compare each record with all the IP address ranges and prefixes in the list. This way, the reading of the result record files is amortized over all IP addresses.

Second, we parallelize the processing of the address range files given the prevalence of multi-core machines. Since address range updates for fixed time intervals are stored in separate files, we parallelize the reading and processing of these files by submitting multiple processes each processing one file at a time. The route changes determined by all these processes are gathered at the end, and sorted in chronological order before generating the output.

2.6 Performance Evaluation

This section presents performance evaluation of the Route Oracle tool. For the pre-processing module, our aim is to verify that it can keep up as BGP updates arrive. For the query module, we evaluate its performance – both response time and resource usage – as a function of the size of the input. We also evaluate the effect of optimizations – parallelization in particular – on the performance of the query module.

Our experiments were run on an standard off-the-shelf SMP server, with two quad-core Xeon X5460 Processors. Each CPU is 3.16 GHz and has 6 MB of cache. The server contains 16 GB of total RAM. We apply our tool to BGP routing table and update messages collected for the month of August 2009 from a top-level route reflector in a tier-1 ISP backbone. The queried IP addresses were randomly selected from the routing table.

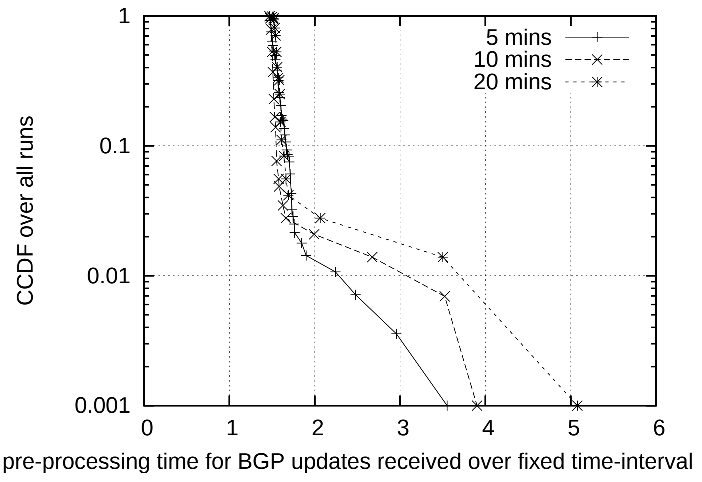


Figure 2.4: CCDF of the time spent by the pre-processing module to convert BGP updates received over fixed time-interval into address range records.

We first evaluate the pre-computation module of the tool, focusing on the time to process BGP updates. Next, we evaluate how the query processing time varies as the query time period grows. Then, we evaluate the scalability of the query processing, as the number of queried IP addresses grow, and show the memory resource consumption. Finally, we demonstrate the benefit of parallelization in query processing.

2.6.1 Pre-processing Time

In this subsection, we evaluate how long it takes to convert BGP updates into address range records. To do this, we consider BGP updates received over fixed time-intervals of 5, 10 and 20 minutes, and compute the time spent by the pre-processing module on each batch of updates. We present results for updates received on August 01, 2009; similar results were observed for other days.

Figure 2.4 shows the pre-processing time for BGP updates received over various fixed time-intervals as a CCDF (complementary cumulative distribution function). As can be seen, of all the BGP updates received and processed in 5 minutes interval, 99% could be processed within 2 seconds, and the maximum time to process the BGP updates received in 20 minutes interval is about 5 seconds. This clearly demonstrates that the pre-processing module is able to handle BGP updates in real time as they arrive.

In order to better understand what factor mainly determines the pre-computation time, we counted the number of BGP updates received over fixed time intervals. For each point in Figure 2.5, the number on the x-axis stands for the number of BGP updates received over the fixed time-interval, while the number

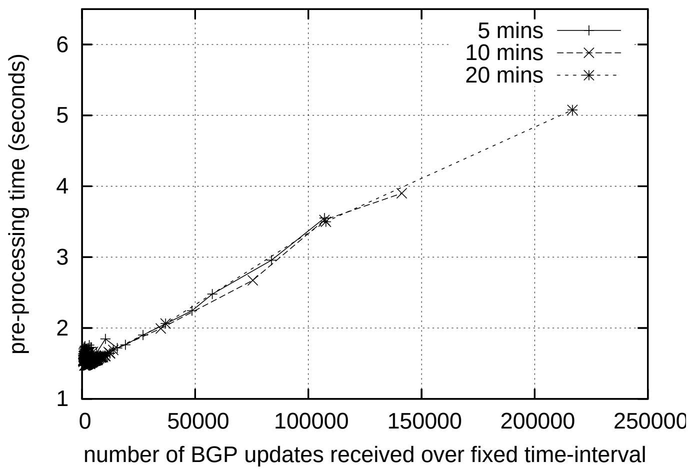


Figure 2.5: Pre-processing time varies according to the number of BGP updates received over the fixed time-interval.

on the y-axis presents the time spent on processing the updates. The figure clearly illustrates the linear relationship between the pre-computation time and the number of BGP updates received over the interval. The start point of each curve shows that it takes approximately 1.5 seconds to finish the fixed steps of the pre-computation part which includes bootstrapping the data structures used by the pre-processing module, and writing the address range records to files at the end. As the number of BGP updates increases, the pre-processing time increases linearly.

2.6.2 Query Processing Time

The query processing time is a function of many parameters, including the length of the query time period, the number of IP addresses queried, and the number of processes run in parallel. In this subsection, we evaluate the query processing time along these three dimensions, and understand how these factors affect the query processing time.

We start by measuring the query processing time for a single IP address, where a single process is used to handle the query. We vary the start and end times of the query time period on August 01, 2009. We first choose a random start time on that day, then choose a random end time between the start time and the end of the day. Figure 2.6 shows the result, where each point stands for one experiment with a randomly chosen IP address to query, random start time and random end time. The x-axis denotes the query end time from the beginning of the day.

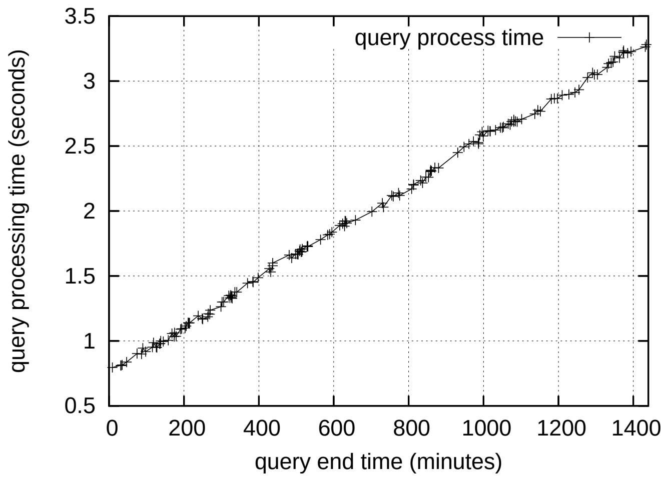


Figure 2.6: Query processing time versus end time.

Figure 2.6 illustrates that the query processing time grows linearly with the end time of the query. This is because irrespective of the start time, the query module has to process all the address range records from the last snapshot (which happens to be the beginning of the day) till the end time since the address range records from the snapshot till the start time are used to determine the routes used at the start time. This explains why the query processing time depends on the length from the beginning of day to the query end time. In addition, note that the time spent on answering the query for a single IP address for a one-day period is no more than 3.5 seconds.

We should also emphasize that the query time can reduced by storing more frequent snapshots at the cost of more storage space. Based on the worst-case time of 3.5 seconds, we believe that storing one snapshot per day provides us with a pretty good performance at a reasonable storage cost (few MBytes per snapshot).

2.6.3 Scalability of Query Processing

Next, we evaluate the scalability of query processing by increasing the number of queried IP addresses. We also show how the amortization of file processing over the IP addresses reduces the query processing time. In addition, we evaluate the memory consumption as the number of queried IP addresses grows. For this experiment, we use the query time period of one hour. We increase the number of number of IP addresses from 1, 2, 4, 8, ... till about 130,000. The IP addresses used for the queries are all randomly chosen. As in

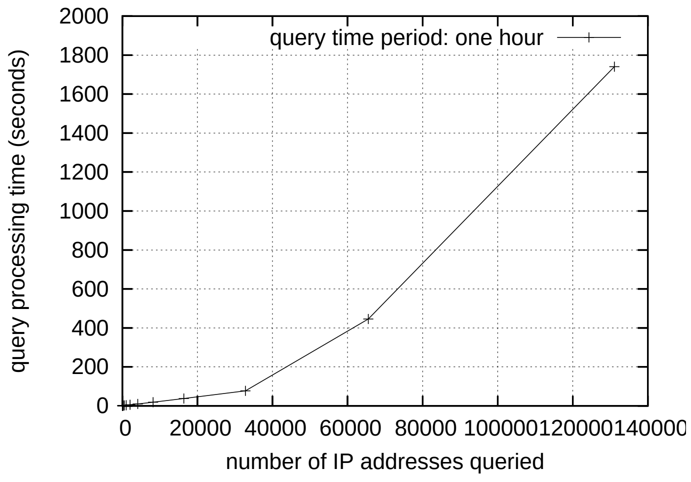


Figure 2.7: Query processing time versus number of IP addresses queried.

the previous section, we use a single process to answer all the queries.

Figure 2.7 shows the query processing time in seconds, as the number of queried IP addresses increases. The figure shows that it takes about half an hour to process 130,000 IP addresses for a time interval of one hour. The query processing time grows linearly as the number of queried IP addresses increases. However, the slope is lower at the beginning of the curve, but increases at some point (in fact two points) as the number of IP addresses increases. We believe this is because when number of IP addresses are lower, the time to read the address range records, which is amortized over all IP addresses, dominates. In contrast, when the number of IP addresses becomes large, the time spent on processing the address range records starts dominating, resulting in a higher slope for the curve.

Figure 2.8 shows the peak virtual memory utilization during the same experiment. As illustrated by the figure, the memory utilization grows as the number of queried IP addresses increases, and the peak virtual memory used for processing about 130,000 IP addresses for an hour is about 480 MB. The increase in memory consumption as the number of queried IP addresses increases is because we store all the matching address range records in memory before outputting them at the end. Another factor affecting the peak virtual memory utilization is the query time period, since the query time period determines the number of address range records that must be stored in memory.

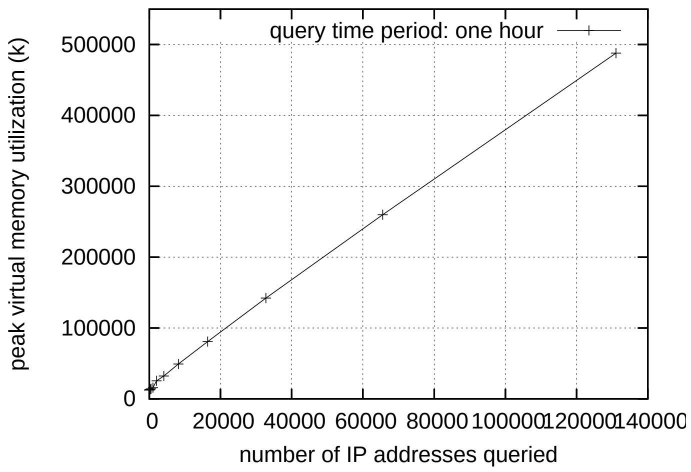


Figure 2.8: Peak virtual memory utilization versus number of IP addresses queried.

#IPs Queried	**Query Processing Time (Secs)**
1	26.1
10	26.1
100	27.7
1000	54.5
10000	279.7

Table 2.4: Query processing time (without parallelization) versus number of IP addresses queried.

2.6.4 Parallelization of Query Processing

Last, we evaluate the benefits of parallelization achieved on multi-core machines. For this experiment, we extend the query time period to 10 days from August 01 to August 10, 2009. We vary the number of IP addresses queried by randomly choosing 1, 10, 100, 1000, and 10000 IP addresses. Recall that the server on which experiments were run had two quad-core CPUs, *i.e.*, 8 processing cores each running at 3.16 GHz.

We start by measuring the query processing time by using only *one* process with no parallelization. These numbers serve as a benchmark against which we compare the processing time with multiple parallel processes. Table 2.4 shows these numbers for varying number of IP addresses over the query time period of 10 days.

Next, for the same set of IP addresses queried, we vary the number of concurrent processes submitted in parallel from 2 to 14. We measure the query processing time, and divide it by the query processing time

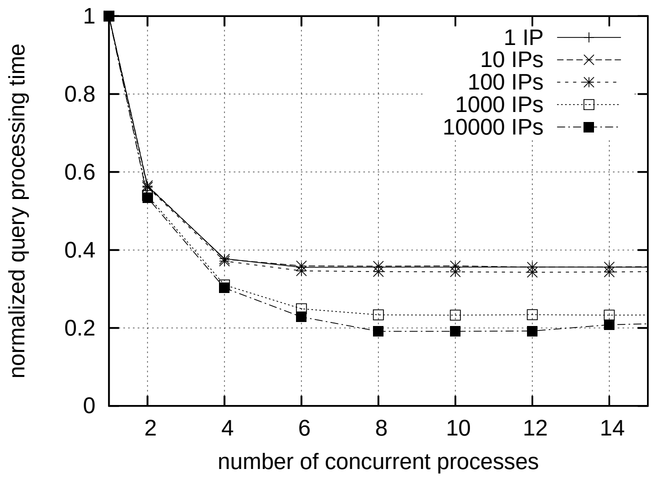


Figure 2.9: normalized query processing time versus number of concurrent processes

without parallelization given by Table 2.4. Figure 2.9 shows the result of this experiment, where the y-axis is the normalized processing time. As illustrated by the figure, the benefits of parallelization increase, as the number of queried IP addresses increases, since more IP addresses mean more time to process the address range records. For example, the query of 10,000 IP addresses takes about 5 minutes to process without parallelization. This number is reduced to 53%, 30%, 23% and 19% of its value when we increase the parallelization to 2, 4, 6 and 8 processes, respectively. No further gain is achieved in processing time by submitting more than eight processes in parallel.

Since the performance levels out when number of processes reach the number of cores, we suspect that the CPU is the performance bottleneck. We confirmed this by tracking the CPU utilization at one minute intervals during the course of the experiment when eight parallel processes were running. As expected, the utilization stayed at 100% during much of the processing, confirming that no further gain is possible beyond eight processes on this particular eight core server.

2.7 Related Work

Understanding route changes is fundamental to network troubleshooting. Packet Design [7] provides Route Explorer to capture the complete stream of routing updates received. In particular, Route Explorer has an animated historical playback feature which lets the operators diagnose problems by quickly rewinding to past routing activities at a specific time. In our work, Route Oracle takes one step further to handle the prefix-match changes, and outputs route changes for IP addresses of interest.

RouteViews [8] and RIPE-NCC [9] provide publicly available passive measurements of BGP route updates. In these projects, the BGP monitors are fed with BGP announcements and withdrawal messages received via an external BGP session with one router in the participating AS. Proposals have been made to pin-point the location and cause of routing changes using these measurement infrastructures [10, 11, 12]. In comparison, Route Oracle allows one to understand how route changes actually affect routes for IP addresses.

Our work also relates to earlier studies that used BGP measurement data to analyze the relationship between IP prefixes [13, 14, 15, 16, 9]. For example, the work on BGP policy atoms [13, 14] showed that groups of related prefixes often have matching AS paths, even when viewed from multiple vantage points; typically, a more-specific prefix had different AS paths than its corresponding less-specific prefix [13]. Other researchers analyzed BGP table dumps to understand the reasons why each prefix appears in the inter-domain routing system, and the reasons include delegation of address space to customers, multi-homing, and load balancing [15, 16]. In contrast, our system focuses on tracking the *changes* in the longest-matching prefix rather than a static analysis of a BGP table dump.

Our work also relates to earlier analysis of BGP routing dynamics [2, 17, 18, 5]. These studies analyze announcement and withdrawal message for each destination prefix, and group related BGP update messages to identify BGP convergence events and route flapping. Whereas these studies treat each IP prefix independently, our analysis of BGP update dynamics focuses on the *relationship between nested prefixes*. Still, we draw on the results in these earlier studies when selecting thresholds for identifying phenomena such as BGP path exploration and route flapping. Our work also relates to measurement studies of prefix hijacking, and particularly *subprefix* hijacking [19, 20] that triggers a change in the longest-matching prefix. Yet, our study considers a wider range of causes of prefix-match changes.

Previous studies have also characterized IP reachability through direct or indirect observations of the underlying data-plane paths used to forward packets [21, 22, 23, 24, 25, 26, 27]. Most of these studies involve active probing (using ping, traceroute, or custom tools) [21, 22, 23, 24], sometimes triggered by passive observations of reachability problems [21, 22]. Other work has focused on analysis of passively collected traffic measurements (such as Netflow data or Web server logs) to detect possible routing changes or reachability problems [25, 26, 27]. In contrast, our work has focused primarily on how the longest-matching prefix, used in packet forwarding, changes over time. That said, these previous studies are quite relevant to our ongoing analysis of the Netflow data to understand the impact of these prefix-match changes on end-to-end reachability.

2.8 Summary

In this chapter, we presented Route Oracle, a scalable system to determine the BGP routes (and thus AS Path and egress router) used by one or more IP addresses from a given router in a network. We believe that the system should form a basis for several network and service management applications. The key component of Route Oracle is an algorithm to track changes to the longest prefix-match for IP addresses as BGP route updates arrive. Our system uses this algorithm to convert BGP updates which are prefix-based into route updates to non-overlapping IP address ranges. This step facilitates queries about route changes for a large number of IP addresses. We also described other optimizations that further reduce the query response time. Our systematic evaluation demonstrated Route Oracle's ability to handle queries at scale and in real time. We have deployed Route Oracle in a large Tier-1 ISP where it is being used to troubleshoot performance impairments for various services. The system optimizations described in the chapter were implemented based on feedback from this user community.

Chapter 3

Diagnosing Wide-Area Latency Increases for CDNs

3.1 Introduction

Content Distribution Networks (CDNs) offer users access to a wide variety of services, running on geographically distributed servers. Many web services are delay-sensitive interactive applications (e.g., search, games, and collaborative editing). CDN administrators go to great lengths to minimize user-perceived latency, by overprovisioning server resources, directing clients to nearby servers, and shifting traffic away from overloaded servers. Yet, CDNs are quite vulnerable to increases in the *wide-area* latency between their servers and the clients, due to interdomain routing changes or congestion in other ASes. The CDN administrators need to detect and diagnose these large increases in round-trip time, and adapt to alleviate the problem (e.g., by directing clients to a different front-end server or adjusting routing policies to select a different path).

To detect and diagnose latency problems, CDNs could deploy a large-scale active-monitoring infrastructure to collect performance measurements from synthetic clients all over the world. Instead, this chapter explores how CDNs can diagnose latency problems based on measurements they can readily and efficiently collect—*passive* measurements of performance [28], traffic [29], and routing from their own networks. Our goal is to maximize the information the CDN can glean from these sources of data. By joining data col-

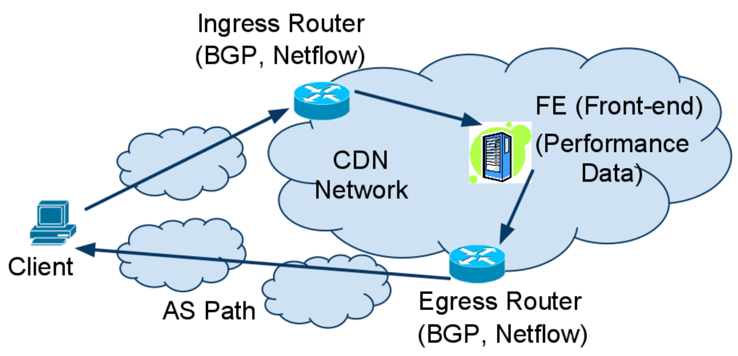


Figure 3.1: CDN architecture and measurements

lected from different locations, the CDN can determine where a client request enters the CDN's network, which front-end server handles the request, and what egress router and interdomain path carry the response traffic, as shown in Figure 3.1. Using this data, we analyze changes in wide-area latency between the clients and the front-end servers. The rest of the user-perceived latency, which is between the front and back-end servers, is under the CDN's direct control and relatively easy to troubleshoot problems.

Finding the *root cause* of latency increases is difficult. Many factors can contribute to higher delays, including internal factors like how the CDN selects servers for the clients, and external factors such as interdomain routing changes. Moreover, separating cause from effect is a major challenge. For example, directing a client to a different front-end server might change where traffic enters and leaves the CDN network and the routes correspondingly. However, in this case, the routing changes is not the original cause of all the changes. Therefore, in order to classify the large increases in latency, our classification must first determine whether client requests shifted to different front-end servers, or the latency to reach the existing servers increased. Only then can we analyze *why* these changes happened. For example, the front-end server may change because the CDN determined that the client is closer to a different server, or because a load-balancing policy needed to shift clients away from an overloaded server. Similarly, if the round-trip time to a specific server increases, routing changes along the forward or reverse path (or both!) could be responsible.

The *scale* of large CDNs also introduces challenges. To measure and control communication with hundreds of millions of users, CDNs typically group clients by an IP prefix or a geographic region. For example, a CDN may collect round-trip times and traffic volumes by IP prefix, or direct clients to front-end servers by region. During any measurement interval, a group of clients may send requests to *multiple* front-end servers, and the traffic may enter and leave the CDN network at *multiple* ingress and egress routers. Thus, in order to analyze the latency increases for groups of requests, we need to define the metrics to distinguish the changes from an individual router or server.

In designing our methodology for classifying large latency increases, we make the following contributions. Our general methodology could be applied to other CDNs, by collecting and using similar data sets.

- **Decision tree for separating cause from effect**: A key contribution of this chapter is that we determine the causal relationship among various factors that lead to latency increases. We propose a *decision tree* for separating the causes of latency changes from their effects, and identify the data sets needed for each step in the analysis. We analyze the measurement data to identify suitable thresholds to identify large latency changes and to distinguish one possible cause from another.

- **Metrics to analyze over *sets* of servers and routers:** Our analysis methodology can analyze latency increases and traffic shifts over *sets* of servers and routers. We propose metrics to quantify the extent of the latency increases caused by various potential causes. For each potential cause, we define the metric to quantify the contribution of latency increases by a single router or server, as well as a way to summarize the contributions across all routers and servers.

- **Latency characterization for Google's CDN:** We apply our methodology to one month of traffic, performance, and routing data from Google's CDN, and identified that nearly 1% of the daily latency changes increase delay by more than 100 msec. Our results show that 73.9% of these large increases in latency were explained (at least in part) by a large increase in latency to reach an existing front-end server, with 42.2% coinciding with a change in the ingress router or egress router (or both!). Around 34.7% of the large increases of latency involved a significant shift of client traffic to different front-end servers, often due to load-balancing decisions or changes in the CDN's own view of the closest server.

- **Case studies to highlight challenges in CDN management:** We present several events in greater detail to highlight the challenges of measuring and managing wide-area performance. These case studies illustrate the difficulty of building an accurate latency-map to direct clients to nearby servers, the extra latency client experience when flash crowds force some requests to distant front-end servers, and the risks of relying on AS path length as an indicator of performance.

The rest of the chapter is organized as follows. Section 3.2 provides an overview of the architecture of the Google CDN, and the datasets we gathered. Section 3.3 describes our decision-tree methodology for characterizing latency increases. Section 3.4 presents a high-level characterization of the latency changes in the Google's CDN, and identifies the large latency events we study. Next, we present the results of our characterization in Section 3.5, followed by several case studies in Section 3.6. Then, we discuss the future research directions in Section 3.7, and present related work in Section 3.8. Finally, we conclude in Section 3.9.

3.2 Measuring the CDN Network

In this section, we first provide a high-level overview of the network architecture of Google's CDN. Then, we describe the measurement dataset we gathered for our analysis in the rest of the chapter.

3.2.1 CDN Architecture

The infrastructure of Google's CDN consists of many servers in the data centers spread across the globe. The client requests are first served at a front-end (FE) server, which provides caching, content assembly, pipelining, request redirection, and proxy functions for the client requests. To have greater control over network performance, CDN administrators typically place front-end servers in managed hosting locations, or ISP points of presence, in geographic regions nearby the clients. The client requests are terminated at the FEs, and (when necessary) served at the backend servers which implement the corresponding application logic. Inside the CDN's internal network, servers are connected by the routers, and IP packets enter and leave the network at edge routers that connect to neighboring ISPs.

Figure 3.1 presents a simplified view of the path of a client request. A client request is directed to an FE, based on proximity and server capacity. Each IP packet enters the CDN network at an *ingress router* and travels to the chosen FE. After receiving responses from the back-end servers, the FE directs response traffic to the client. These packets leave the CDN at an *egress router* and follow an *AS path* through one or

Data Set	Collection Point	Logged Information
Performance	front ends (FEs)	(client /24 prefix, country, RPD, average RTT)
Traffic	ingress routers egress routers	(client IP address, FE IP address, bytes-in) (FE IP address, client IP address, bytes-out)
Routing	egress routers	(client IP prefix, AS path)
Joint data	(client IP prefix, FE, RPD, RTT, {ingress, bytes-in}, {egress, AS path, bytes-out})	

Table 3.1: Measurements of wide-area performance, traffic, and routing

more Autonomous Systems (ASes) en route to the client. The user-perceived latency is affected by several factors: the location of the servers, the path from the client to the ingress router, and the path from the egress router back to the client. From the CDN's perspective, the visible factors are: the ingress router, the selection of the servers, the egress router, and the AS path.

Like many CDNs, Google uses DNS to direct clients to front-end servers, based first on a *latency map* (preferring the FE with the smallest network latency) and second on a *load-balancing* policy (that selects another nearby FE if the closest FE is overloaded) [30]. To periodically construct the latency map, the CDN collects round-trip statistics by passively monitoring TCP transfers to a subset of the IP prefixes. In responding to a DNS request, the CDN identifies the IP prefix associated with the DNS resolver and returns the IP address of the selected FE, under the assumption that end users are relatively close to their local DNS servers. Changes in the latency map can lead to shifts in traffic to different FEs. The latency between the front-end and back-end servers is a known and predictable quantity, and so our study focuses on the network latency—specifically, the round-trip time—between the FEs and the clients.

3.2.2 Passive Measurements of the CDN

The measurement data sets, which are routinely collected at the servers and routers, are summarized in Table 3.1. The three main datasets—performance, traffic, and routing measurements—are collected by different systems. The measurement data gathered is composed of latency sensitive traffic, and specifically excludes video traffic, as that is latency insensitive.

Client performance (at the FEs): The FEs monitor the round-trip time (RTT) for a subset of the TCP connections by measuring the time between sending the SYN-ACK and receiving an ACK from the client. If this SYN-ACK RTT is larger than the RTT for data transfers in the same TCP connection, then the servers log the shorter RTT value instead (which is the RTT for data transfer in this case). These measurements capture the propagation and queuing delays along both the forward and reverse paths to the clients. Each FE also counts the number of requests, producing a daily summary of the round-trip time (RTT) and the requests per day (RPD) for each /24 IP prefix. Each /24 prefix is associated with a specific country, using the IP address-to-Geo mapping. We use it to group prefixes in nearby geographical regions for our study.

Netflow traffic (at edge routers): The edge routers collect traffic measurements using Netflow [29]. The client is the source address for incoming traffic and the destination address for outgoing traffic; similarly, the FE is the destination for incoming traffic, and the source for outgoing traffic. Netflow performs packet sampling, so the traffic volumes are estimates after correcting for the sampling rate. This leads to records that summarize traffic in fifteen-minute intervals, indicating the client IP address, front-end server address, and traffic volume. Traffic for a single client address may be associated with multiple routers or FEs during the interval.

BGP routing (at egress routers): The edge routers also collect BGP routing updates that indicate the sequence of Autonomous Systems (ASes) along the path to each client IP prefix. (Because BGP routing is destination based, the routers cannot collect similar information about the forward path from clients to the FEs.) A dump of the BGP routing table every fifteen minutes, aligned with the measurement interval for the Netflow data, indicates the AS-PATH of the BGP route used to reach each IP prefix from each egress router.

Joint data set: The joint data set used in our analysis combines the performance, traffic, and routing data, using the client IP prefix and FE IP address as keys in the join process. First, the traffic and routing data at the egress routers are joined by matching the client IP address from the Netflow data with the longest-matching prefix in the routing data. Second, the combined traffic and routing data are aggregated into summaries and joined with the performance data, by matching the /24 prefix in the performance data with the longest-matching prefix from the routing data. Note that the /24 prefix is the finest granularity in our measurement study. The resulting joint data set captures the traffic, routing, and performance for each client IP prefix and front-end server, as summarized in Table 3.1. The data set is aggregated to prefix level. In addition, the data do not contain any user-identifiable information (such as packet payloads, timings of individual requests, etc.) The data set we study is based on a sample, and does not cover all of the CDN network.

The data have some unavoidable limitations, imposed by the systems that collect the measurements: the performance data does not indicate which ingress and egress router were used to carry the traffic, since the front-end servers do not have access to this information. This explains why the joint data set has a *set* of ingress and egress routers. Fortunately, the Netflow measurements allow us to estimate the request rate for the individual ingress routers, egress routers, and AS paths from the observed traffic volumes; however, we cannot directly observe how the RTT varies based on the choice of ingress and egress routers. Still, the joint data set provides a wealth of information that can shed light on the causes of large latency increases.

Latency map, and front-end server capacity and demand: In addition to the joint data set, we analyze changes to the latency map used to drive DNS-based server selection, as discussed in more detail in Section 3.3.2. We also collect logs of server capacity and demand at all front-end servers. We use the logs to determine whether specific FE is overloaded at a given time (when the demand exceeded capacity, and requests were load balanced to other front-end servers).

3.3 Methodology

Analyzing wide-area latency increases is difficult, because multiple inter-related factors can lead to higher round-trip times. As the choice of ingress and egress routers depends on the front-end server, our classification starts by determining whether or not the front-end server changes, as shown in the first branch in Figure 3.2. Our analysis accounts for the fact that clients may direct traffic to *multiple* front ends, either because the front-end server changes or because different clients in the same region use different front-end servers.

In this section, we present the methodology we use to analyze latency increases, as illustrated in Figure 3.2. First, we propose metrics for distinguishing FE changes from latency changes that affect individual FEs. Then, we describe the techniques to identify the cause of FE changes (the latency map, or load balancing). Lastly, we present the method to correlate the latency increases that affect individual FEs with routing changes. Table 3.2 summarizes the notation used in this chapter.

3.3.1 Front-End Server and Latency Metrics

Analyzing latency increases is relatively easy if all clients use the same FE. In this case, the average round-trip time could increase for one of two main reasons: front-end server changes, or latency increases at specific FEs. It is worth mention that each front-end server represents one data center, which has hundreds to thousands of servers. Because of the high redundancy in the data centers, there is no category for FE

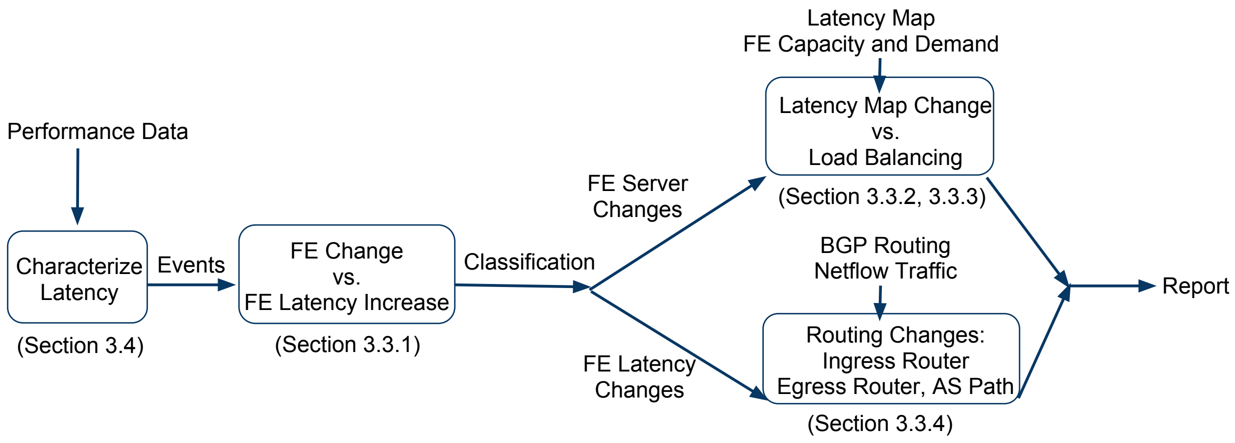


Figure 3.2: Classification of large latency changes using passive measurements

failures (which means the entire data center is shut down).

- **Front-end server changes (ΔFE):** The clients switch from one front-end server to another, where the new server used has a higher RTT. This change could be caused by an FE failure or a change in the CDN's server-selection policies, as shown in the upper branch of Figure 3.2.

- **Front-end latency changes (ΔLat):** The clients could continue using the same FE, but have a higher RTT for reaching that server. The increased latency could stem from changes along the forward or reverse path to the client, as shown in the lower branch of Figure 3.2.

The analysis is more difficult if clients contact multiple front-end servers, and the RTT and RPD for each server changes. Correctly distinguishing all of these factors requires grappling with *sets* of front-ends and *weighing* the RTT measurements appropriately.

The average round-trip time experienced by the clients is the average over the requests sent to multiple front-ends, each with its own average round-trip time. For example, consider a region of clients experiencing an average round-trip time of RTT_1 at time 1, with a request rate of RPD_{1i} and round-trip time RTT_{1i} for each front-end server i. Then,

$$RTT_1 = \sum_i RTT_{1i} * \frac{RPD_{1i}}{RPD_1}$$

Symbol	Meaning
RTT_1, RTT_2	round-trip time for a client region at time 1 and time 2
ΔRTT	change in RTT from time 1 to time 2 (i.e., $RTT_2 - RTT_1$)
RTT_{1i}, RTT_{2i}	round-trip time for requests to FE_i at time 1 and time 2
RPD_1, RPD_2	requests for a client region at time 1 and time 2
RPD_{1i}, RPD_{2i}	requests to FE_i at time 1 and time 2
ΔFE_i	latency change contribution from traffic shifts at FE_i
ΔLat_i	latency change contribution from latency changes at FE_i
ΔFE	latency change contribution from traffic shifts at all FEs
ΔLat	latency change contribution from latency changes at all FEs
r_{1i}, r_{2i}	fraction of requests served at FE_i predicted by the latency map at time 1 and time 2
$\Delta LatMap$	fraction of requests shifting FEs predicted by the latency map
$\Delta FEDist$	actual fraction of requests shifting FEs
$LoadBalance_1$	fraction of requests shifting FEs by the load balancer at time 1
$\Delta LoadBal$	difference of the fraction of requests shifting FEs by the load balancer from time 1 to time 2
$\Delta Ingress$	fraction of the traffic shifting ingress router at a specific FE
$\Delta EgressASPath$	fraction of the traffic shifting (egress router, AS path) at a specific FE

Table 3.2: Summary of key notation

where $RPD_1 = \sum_i RPD_{1i}$ is the total number of requests from that region, across all front-end servers, for time period 1. A similar equation holds for the second time period, with the subscripts changed to consider round-trip times and request rates at time 2.

The increase in average round-trip time from time 1 to time 2 (i.e., $\Delta RTT = RTT_2 - RTT_1$) is, then,

$$\Delta RTT = \sum_i \left(RTT_{2i} * \frac{RPD_{2i}}{RPD_2} - RTT_{1i} * \frac{RPD_{1i}}{RPD_1} \right)$$

The equation shows how the latency increases could come either from a higher round-trip time for the same server (i.e., $RTT_{2i} > RTT_{1i}$) or a shift in the fraction of requests directed to each FE (i.e., RPD_{2i}/RPD_2 vs. RPD_{1i}/RPD_1), or both. Note that in cases when the set of FE server changes, this equation still holds.

To tease these two factors apart, consider one FE i, and the term inside the summation. We can split the term into two parts that sum to the same expression, where the first captures the impact on the round-trip time from traffic shifting toward front-end server i:

$$\Delta FE_i = RTT_{2i} * \left(\frac{RPD_{2i}}{RPD_2} - \frac{RPD_{1i}}{RPD_1} \right)$$

where ΔFE_i is high if the fraction of traffic directed to front-end server i increases, or if the round-trip time is high at time 2. The second term captures the impact of the latency to front-end server i increasing:

$$\Delta Lat_i = (RTT_{2i} - RTT_{1i}) * \frac{RPD_{1i}}{RPD_1}$$

where the latency is weighted by the fraction of requests directed to front-end server i, to capture the relative impact of this FE on the total increase in latency. Through simple algebraic manipulation, we can show that

$$\Delta RTT = \sum_i (\Delta FE_i + \Delta Lat_i).$$

As such, we can quantify the contribution to the latency change that comes from shifts between FEs:

$$\Delta FE = \sum_i \Delta FE_i / \Delta RTT$$

and latency changes for individual front-end servers

$$\Delta Lat = \sum_i \Delta Lat_i / \Delta RTT$$

where the factors sum to 1. For example, if the FE change contributes 0.85 and the latency change contributes 0.15, we can conclude that the latency increase was primarily caused by a traffic shift between front-end servers. If the FE change contributes -0.1 and the latency change contributes 1.1, we can conclude that the latency increase was due to an increase in latency to reach the front-end servers rather than a traffic shift; if anything, the -0.1 suggests that some traffic shifted to FEs with *lower* latency, but this effect was dwarfed by one or more FEs experiencing an increase in latency.

In the following subsections, we present the method to identify the causes of the FE changes: the latency map and load balancing.

3.3.2 Latency Map: Closest Front-End Server

Google CDN periodically constructs a latency map to direct clients to the closest front-end server. The CDN constructs the latency map by measuring the round-trip time for each /24 prefix to different front-end servers, resulting in a list mapping each /24 prefix to a single, closest FE. From the latency map, we can compute the target distribution of requests over the front-end servers for groups of co-located clients in two time intervals. To combine this information across all /24 prefixes in the same region, we weigh by the requests per day (RPD) for each /24 prefix. This results in a distribution of the fraction of requests r_{1i} from the client region directed to front-end server i, at time 1.

As the latency map and the request rates change, the region may have a different distribution $\{r_{2i}\}$ at time 2. To analyze changes in the latency map, we consider the fraction of requests that should shift to different front-end servers:

$$\Delta LatMap = \sum_i |r_{2i} - r_{1i}|/2$$

Note that we divide the difference by two, to avoid double counting the fraction of requests that move away from one FE (i.e., $r_{2i} - r_{1i}$ decreasing for one front-end server i) and towards another (i.e., $r_{2i} - r_{1i}$ increasing for some other front-end server).

3.3.3 Load Balancing: Avoiding Busy Servers

In practice, the actual distribution of requests to front-end servers does not necessarily follow the latency map. Some FEs may be overloaded, or unavailable due to maintenance. To understand how the traffic distribution changes in practice, we quantify the changes in front-end servers as follows:

$$\Delta FEDist = \sum_i \left| \frac{RPD_{2i}}{RPD_2} - \frac{RPD_{1i}}{RPD_1} \right| /2$$

That is, we calculate the fraction of requests to FE i at time 1 and time 2, and compute the difference, summing over all front-end servers. As with the equation for $\Delta LatMap$, we divide the sum by two to avoid double counting shifts away from one front-end server and shifts toward another.

The differences are caused by the CDN's own load-balancing policy, which directs traffic away from busy front-end servers. This may be necessary during planned maintenance. For example, an FE may consist of a cluster of computers; if some of these machines go down for maintenance, the aggregate server capacity decreases temporarily. In other cases, a surge in client demand may temporarily overload the closest front-end server. In both cases, directing some clients to an alternate front-end server is important for preventing degradation in performance. A slight increase in round-trip time for some clients is preferable to all clients experiencing slower downloads due to congestion.

To estimate the fraction of requests shifted by the load balancer, we identify front-end servers that handle a *lower* fraction of requests than suggested by the latency map. The latency map indicates that front-end server i should handle a fraction r_{1i} of the requests for the clients at time 1. In reality, the server handles RPD_{1i}/RPD_1.

$$LoadBalance_1 = \sum_i \left[r_{1i} - \frac{RPD_{1i}}{RPD_1} \right]^+$$

where $[]^+$ indicates that the sum only includes the positive values, with the target request load in excess of the actual load. Similarly, we define the fraction of queries load balanced at time 2 as $LoadBalance_2$.

If much more requests are load balanced on the second day, then more requests are directed to alternative FEs that are further away, leading to higher round-trip times. Thus, we use the difference of the load balancer metric to capture more load balancing traffic at time 2:

$$\Delta LoadBal = LoadBalance_2 - LoadBalance_1$$

We expect the load-balancing policy to routinely trigger some small shifts in traffic.

3.3.4 Identifying Routing Changes

Next, our analysis focuses on events where the RTT jumps significantly for specific FEs. Note that multiple FEs might share the same ingress or egress routers. These increases in round-trip time at a specific FE could be caused by routing changes, or by congestion along the paths to and from the client. Since the CDN does not have direct visibility into congestion outside its own network, we correlate the RTT increases only with the routing changes visible to the CDN—changes of the ingress router where client traffic enters the CDN network, and changes of the egress router and the AS path used to reach the client.

Recall that the latency metric ΔLat can be broken down to the sum of latency metrics at individual FEs (i.e., ΔLat_i). We focus our attention on the FE with the highest value of ΔLat_i, because the latency change for requests to this FE has the most impact on the latency increase seen by the clients. Then, we define metrics to capture what fraction of the traffic destined to this FE experiences a visible routing change. Focusing on the front-end server i with the highest latency increase, we consider where the traffic enters the network. Given all the traffic from the client region to the front-end server, we can compute the fractions f_{1j} and f_{2j} entering at ingress router j at time 1 and time 2, respectively. Note that we compute these fractions from the "bytes-in" statistics from the Netflow data, since the front-end server cannot differentiate the requests per day (RPD) by which ingress router carried the traffic.

To quantify how traffic shifted to different ingress routers, we compute:

$$\Delta Ingress = \sum_j |f_{2j} - f_{1j}|/2$$

Note that the difference between the fractions is divided by two, to avoid double counting traffic that shifts away from one ingress router and toward another. Similarly, we define a metric to measure the fraction of traffic to a FE that switches to a different egress router or AS path. Suppose the fraction of traffic to (egress router, AS path) k is g_{1k} at time 1 and g_{2k} at time 2. Then,

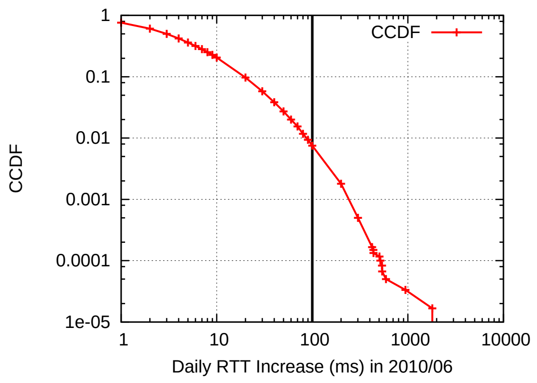
$$\Delta EgressASPath = \sum_k |g_{2k} - g_{1k}|/2$$

similar to the equation for analyzing the ingress routers. These metrics allow us to correlate large increases in latency to server i with observable routing changes. Note that the analysis can only establish a correlation between latency increases and routing changes, rather than definitively "blaming" the routing change for the higher delay, since the performance measurements cannot distinguish RTT by which ingress or egress router carried the traffic.

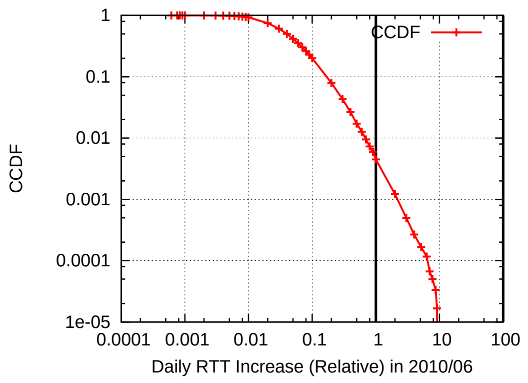
3.4 Distribution of Latency Changes

In the rest of the chapter, we apply our methodology to measurement data from Google's CDN. The BGP and Netflow data are collected and joined on a 15-minute timescale; the performance data is collected daily, and joined with the routing and traffic data to form a joint data set for each day in June 2010. For our analysis, we group clients by "region," combining all IP addresses with the same origin AS *and* located in the same country. In this section, we describe how we preprocess the data, and characterize the distribution of daily increases in latency to identify the most significant events.

As our datasets are proprietary, we are not able to reveal the exact number of regions, events, or threshold of RPD. Instead, we report percentages in our tables and graphs. We believe percentages are more meaningful, since the exact number of events and regions naturally differ from one CDN to another. In addition, the granularity of the data, both spatially (i.e., by region) and temporally (i.e., by day) are beyond our control; these choices are not fundamental to our methodology, which could easily be applied to



(a) Absolute RTT increase



(b) Relative RTT increase

Figure 3.3: Distribution of daily RTT increase

finer-grain measurement data.

3.4.1 Aggregating Measurements by Region

Our joint dataset has traffic and performance data at the level of BGP prefixes, leading to approximately 250K groups of clients to consider. Many of these prefixes generate very little traffic, making it difficult to distinguish meaningful changes in latency from statistical noise. In addition, CDN administrators understandably prefer to have more concise summaries of significant latency changes that affect many clients, rather than reports for hundreds of thousands of prefixes.

Combining prefixes with the same origin AS seems like a natural way to aggregate the data, because many routing and traffic changes take place at the AS level. Yet, some ASes are quite large in their own right, spanning multiple countries. We combine prefixes that share the same country and origin AS (which we define as a *region*), for our analysis. From the performance measurements, we know the country for each /24 prefix, allowing us to identify the country (or set of countries) associated with each BGP prefix. A prefix spanning multiple countries could have large variations in average RTT simply due to differences in the locations of the active clients. As such, we filter the small number of BGP prefixes spanning multiple countries. This filters approximately 2K prefixes, which contribute 3.2% of client requests and 3.3% of the traffic volume.

After aggregating clients by region, some regions still contribute very little traffic. For each region, we calculate the minimum number of requests per day (RPD) over the month of June 2010. The distribution of monthly minimum RPD over all the regions (not shown) reveals that most requests come from a small fraction of the regions. We choose a threshold for the minimum RPD to filter the regions with very low client demand. This process improves statistical accuracy, because it makes sure that we have enough samples of requests for the regions we study. This also helps focus our attention on regions with many clients, and reduces the volume of the measurement data we analyze. This process excludes 85.8% of the regions, but only 6% of the traffic.

Hence, for the rest of our analysis, we focus on clients aggregated by region (i.e., by country and origin AS), and regions generating a significant number of requests per day. Note that our analysis methodology could be applied equally well to alternate ways of aggregating the clients and filtering the data.

3.4.2 Identifying Large Latency Increases

To gain an initial understanding of latency changes, we first characterize the differences in latency from one day to the next throughout the month of June 2010, across all the client regions we selected. For each region, we calculate the average daily RTT for our study. The average RTT is calculated by aggregating over all the prefixes, weighted by the RPD for each prefix. In the rest of the chapter, we refer RTT to the daily average RTT by regions.

We consider both the *absolute* changes (i.e., $RTT_2 - RTT_1$) and the *relative* change (i.e., $(RTT_2 - RTT_1)/RTT_1$), as shown in Figures 3.3(a) and 3.3(b), respectively. The graphs plot only the *increases* in latency, because the distributions of daily increases and decreases are symmetric.

The two graphs are plotted as complementary cumulative distributions, with a logarithmic scale on both axes, to highlight the large outliers. Figure 3.3(a) shows that latency increases less than 10msec for 79.4% of the time. Yet, nearly 1% of the latency increases exceed 100 msec, and every so often latency increases by more than one second. Figure 3.3(b) shows that the RTT increases by less than 10% in 80.0% of cases. Yet, the daily RTT at least doubles (i.e., a relative increase of 1 or more) for 0.45% of the time, and we see occasional increases by a factor of ten.

We define an *event* to be a daily RTT increase over a threshold for a specific region. Table 3.3 summarizes the events we selected to characterize the latency increase. We choose the threshold of absolute RTT increase as 100 ms and the threshold of relative RTT increase as 1, leading to a combined list of hundreds of events corresponding to the most significant increases in latency: with 76.9% of the events over the absolute RTT increase threshold; 35.6% of the events over the relative RTT increase threshold; and 12.5%

% Events	Category
76.9%	Absolute RTT Increase $\geq$ 100 ms
35.6%	Relative RTT Increase $\geq$ 1

Table 3.3: Events with a large daily RTT increase

of the events over both thresholds.

3.5 Characterization

In this section, we characterize the events of large regional latency increases, which are identified in the previous section. By applying the methodology in Section Section 3.3, we first classify them into FE changes and latency increases at individual FEs. Then, we further classify the events of FE changes according to the causes of the latency map and load balancing; classify the events of FE latency increases according to the causes of inter-domain routing changes.

Our high-level results in this section are summarized in Table 3.4. Nearly three-quarters of these events were explained (at least in part) by a large increase in latency to reach an existing front-end server. These latency increases often coincided with a change in the ingress router or egress router (or both!); still, many had no visible interdomain routing change and were presumably caused by BGP routing changes on the forward path or by congestion or intradomain routing changes. Around one-third of the events involved a significant shift of client traffic to different front-end servers, often due to load-balancing decisions or changes in CDN's own view of the closest server. Nearly 9% of events involved both an "FE latency increase" *and* an "FE server change," which is why they sum to more than 100%.

3.5.1 FE Change vs. Latency Increase

Applying our methodology to each event identified in the last section, we see that large increases in latency to reach existing servers (i.e., ΔLat) are responsible for more than two-thirds of the events with a large increase in round-trip time. To identify the cause of latency increases, we first show the CDF of ΔFE (traffic shift) and ΔLat (latency increase) for the events we study in Figure 3.4. The distributions are a reflection of each other (on both the x and y axes), because ΔFE and ΔLat sum to 1 for each event.

The graph shows that about half of the events have ΔFE below 0.1, implying that shifts in traffic from one FE to another are not the major cause of large-latency events. Still, traffic shifts are responsible

Category	**% Events**
FE latency increase	*73.9%*
Ingress router	10.3%
(Egress, AS Path)	14.5%
Both	17.4%
Unknown	31.5%
FE server change	*34.7%*
Latency map	14.2%
Load balancing	2.9%
Both	9.3%
Unknown	8.4%
Total	*100.0%*

Table 3.4: Classification of large latency increases (where latency more than doubles, or increases by more than 100 msec), relative to previous day. Note that nearly 9% of events involve both FE latency increases and FE server changes.

	Threshold		
	0.3	**0.4**	**0.5**
ΔLat	61%	65%	71%
ΔFE	23%	26%	29%
Both	16%	9%	0%

Table 3.5: Events classified by ΔLat and ΔFE

for *some* of the latency increases—one event has a ΔFE of 5.83! (Note that we do not show the very few points with extreme ΔFE or ΔLat values, so we can illustrate the majority of the distribution more clearly in the graph). In comparison, ΔLat is often fairly high—in fact, more than 70% of these events have a ΔLat higher than 0.5.

To classify these events, we apply a threshold to both distributions and identify whether ΔFE or ΔLat (or both) exceeds the threshold. Table 3.5 summarizes the results for thresholds 0.3, 0.4, and 0.5. These results show that, for a range of thresholds, around two-thirds of the events are explained primarily by an increase in latency between the clients and the FEs. For example, using a threshold of 0.4 for both distributions, 65% of events have a large ΔLat and another 9% of events have large values for both metrics, resulting in nearly three-quarters of the events caused (in large part) by increases in RTTs to select front-end servers. In the rest of the chapter, we apply a threshold of 0.4 to distinguish events into the three categories in Table 3.5. This is because the threshold of 0.5 separates the two categories apart; the threshold of 0.3 (where one factor contributes to 30% of the latency increases) is not as significant as 0.4.

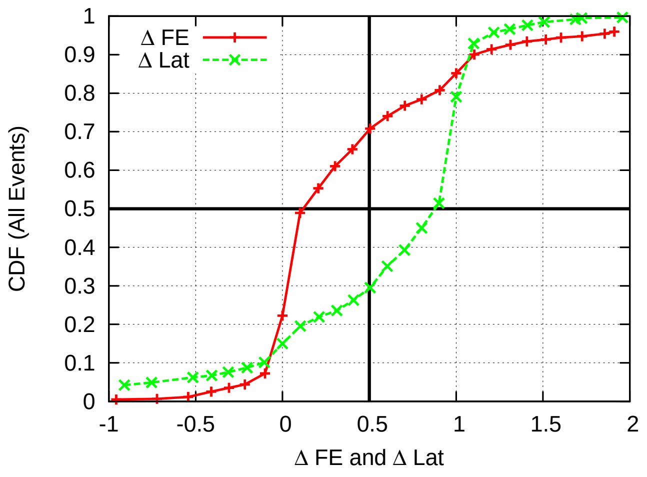


Figure 3.4: ΔFE and ΔLat for large events

3.5.2 Normal Front-End Changes

To understand the normal distribution of latency-map changes, we calculate $\Delta LatMap$ for *all* of the regions—whether or not they experience a large increase in latency—on two consecutive days in June 2010. Figure 3.5 shows the results. For 76.9% of the regions, less than 10% of the requests change FEs because of changes to the latency map. For 85.7% of regions, less than 30% of traffic shifts to different front-end servers. Less than 10% of the regions see more than half of the requests changing front-end servers. Often, these changes involve shifts to another front-end server in a nearby geographic region.

However, note that the distribution of $\Delta LatMap$ has a long tail, with some regions having 80% to 90% of the requests shifting FEs. For these regions, changes in the measured latency lead to changes in the latency map which, in turn, lead to shifts in traffic to different front-end servers. These outliers are not necessarily a problem, though, since the FEs on the second day may be very close to the FEs on the first day. To understand the impact of these traffic shifts, we need to consider the resulting latency experienced by the clients.

Figure 3.5 also shows the resulting distribution of $\Delta FEDist$ (i.e., the actual FE changes) for all client regions for one pair of consecutive days in June 2010. As expected, the distribution matches relatively closely with the distribution for $\Delta LatMap$, though some significant differences exist. Sometimes the traffic shifts even though the latency map does not change. This is evident in the lower left part of the graph, where most client regions see little or no change to the latency map, but a higher fraction experience

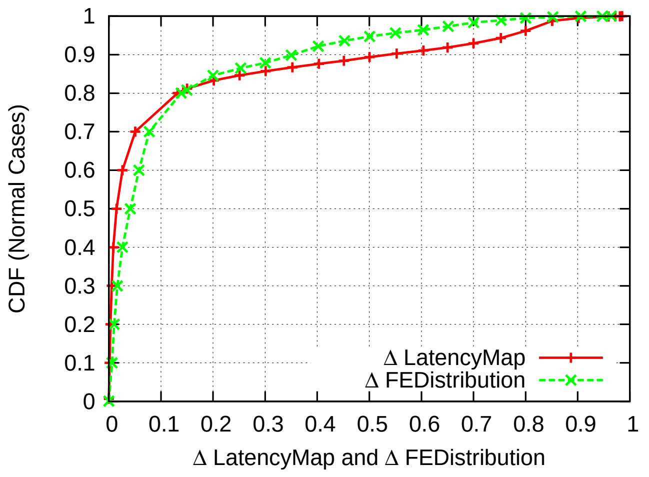


Figure 3.5: Distribution of $\Delta LatMap$ and $\Delta FEDist$ across all client regions for one day in June 2010

as much as a 5% shift in traffic.

We expect the load-balancing policy to routinely trigger some small shifts in traffic. Figure 3.6 plots the distribution of $\Delta LoadBal$ for all client regions for a single day in June 2010, as shown in the "Normal Cases" curve. As expected, most clients are directed to the closest front-end server, as indicated by the clustering of the distribution around $\Delta LoadBal = 0$. In the next subsection, we show that the large latency events coincide with larger shifts in traffic, as illustrated by the "ΔFE Events" curve in Figure 3.6.

3.5.3 Front-End Changes during Events

To understand the influence of traffic shifts during the events, we analyze the large-latency events where front-end changes are a significant contributor to the increase in latency (i.e., $\Delta FE \geq 0.4$); 35% of the events fall into this category, as shown earlier in Table 3.5. Figure 3.7 plots the distributions of $\Delta LatMap$ and $\Delta FEDist$ for these events. For these events, the FE distribution still mostly agrees with the latency map. Compared with the curves in Figure 3.5, the events which experienced large latency increases have a stronger correlation with FE changes. According to the latency map, only 14% of events have fewer than 10% of requests changing FEs; 46% of the events have more than half of queries shifting FEs. Note that FE changes (i.e., in nearby geographical locations) do not necessarily lead to large latency increases, and may even improve user-perceived throughput by avoiding busy servers. That said, these FE changes can cause increases in round-trip time, so we need to understand how and why they happen.

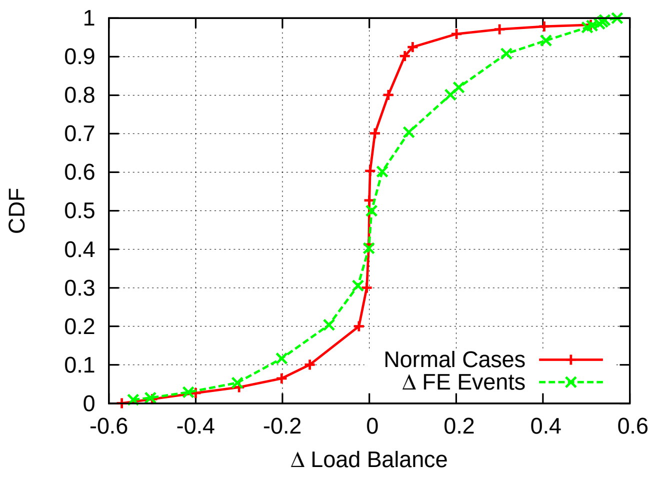


Figure 3.6: Distribution of $\Delta LoadBalance$ for Normal Cases and Events $\Delta FE \geq 0.4$

We then calculate the $\Delta LoadBal$, the difference of fraction of traffic directed by the load balancer from one day to the next. Figure 3.6 shows the distribution of $\Delta LoadBal$ for these events and for all client regions. As illustrated in the figure, 92.5% of the normal cases have less than 10% of requests shifted away from the closest front-end server. In contrast, for the ΔFE events, 27.7% of the events have a $\Delta LoadBal$ value greater than 10%; more than 9% of the events have a $\Delta LoadBal$ in excess of 0.3, suggesting that the load-balancing policy is responsible for many of the large increases in latency.

Based on the $\Delta LatMap$ and $\Delta LoadBal$ metrics, we classify the events into four categories: (i) correlated only with latency map changes, (ii) correlated only with load balancing changes, (iii) correlated with both latency-map changes and load balancing; and (iv) unknown. We choose the 85th-percentile and 90th-percentile in the distribution for the normal cases as the thresholds for $\Delta LatMap$ and $\Delta LoadBal$. Table 3.6 summarizes the results: 26.7% of the events are correlated with both changes to the latency map and load balancing; 40.8% of the events only with changes in the latency map; 8.3% of the events only with load balancing; and 24.3% of the events fall into the unknown category. The table also shows results for the 90th-percentile thresholds.

Note that in the “unknown” category, although the fraction of traffic shifting FEs is low, this does not mean that the FE change is not responsible for the latency increases. This is because: what matters is the latency difference between the FEs, not only the fraction of traffic shifting FEs. For example, even if the fraction of traffic shifting is small, the absolute increase in latency may be high because the clients are directed to FEs with really large RTTs. Completing this analysis is part of our ongoing work.

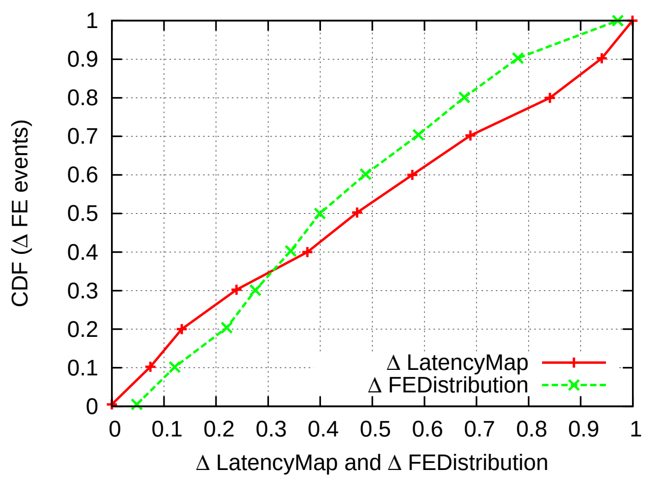


Figure 3.7: Distribution of $\Delta LatencyMap$ and $\Delta FEDistribution$ for Events $\Delta FE \geq 0.4$

Threshold (Percentile)	**(0.27, 0.06) 85th**	**(0.53, 0.08) 90th**
Latency Map	40.8%	23.8%
Load Balancing	8.3%	12.6%
Both	26.7%	18.4%
Unknown	24.3%	45.1%

Table 3.6: Classification of Events with $\Delta FE \geq 0.4$

3.5.4 Inter-domain Routing Changes

In this subsection, we study the events where the round-trip time increases to existing front-end servers. We characterize the events based on these metrics, and classify events based on changes to the ingress router, the egress router and AS path, or both.

For better insight into whether routing changes are responsible for latency increases, we first consider the prevalence of routing changes for *all* client regions—when latency does *not* necessarily increase significantly—for a pair of consecutive days in June 2010. Figure 3.8 shows the CCDF, with the y-axis cropped at 0.45, to highlight the tail of the distribution where clients experience a large shift in ingress routers. Significant routing changes are relatively rare for the "Normal Cases." In fact, 76.6% of the client regions experience *no* change in the distribution of traffic across ingress routers. Less than 7% of the regions experience a shift of more than 10%. As such, we see that shifts in where traffic enters Google's CDN network do not occur often, and usually affect a relatively small fraction of the traffic.

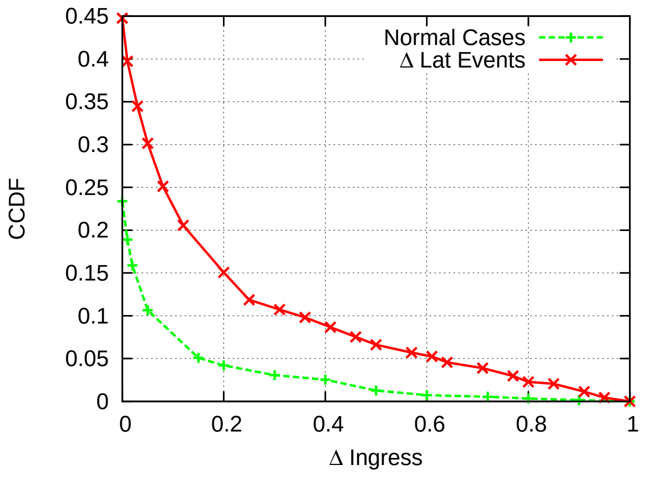


Figure 3.8: Ingress router shifts ($\Delta Ingress$)

However, large shifts in ingress routers are more common for the events where the round-trip time to a front-end server increases significantly (i.e., $\Delta Lat \geq 0.4$), as shown by the "ΔLat Events" curve in Figure 3.8. The events we study have a much stronger correlation with changes in the ingress routers, compared with the normal cases. Though 55% of these events do not experience *any* change in ingress routers, 22.2% of events see more than a 10% shift, and 6.7% of the events see more than *half* of the traffic shifting ingress routers.

Similarly, we calculate $\Delta EgressASPath$ for both the normal cases and the ΔLat events, as illustrated in Figure 3.9. Compared with ingress changes, we see more egress and AS path changes, in part because we can distinguish routing changes at a finer level of detail since we see the AS path. For the normal cases, 63% of the client regions see no change in the egress router or the AS path; 91% see less than 10% of the traffic shifting egress router or AS path. In comparison, for the "ΔLat Events," only 39% of the events see no changes in the egress routers and AS paths; 32% of the events see more than 10% of the traffic changing egress router or AS path, and 10% of the events see more than *half* of the traffic shifting egress routers and/or AS paths.

Based on both of the routing indicators, we classify the events into four categories: (i) correlated only with ingress router changes, (ii) correlated only with changes in the egress router and AS path, (iii) correlated with both ingress changes and egress/AS-path changes, and (iv) unknown. To identify significant shifts, we look to the distributions for "Normal Cases" and consider the 85th and 95th percentiles for shifts in both $\Delta Ingress$ and $\Delta EgressASPath$. Table 3.7 summarizes the results. Based on the 85th-percentile

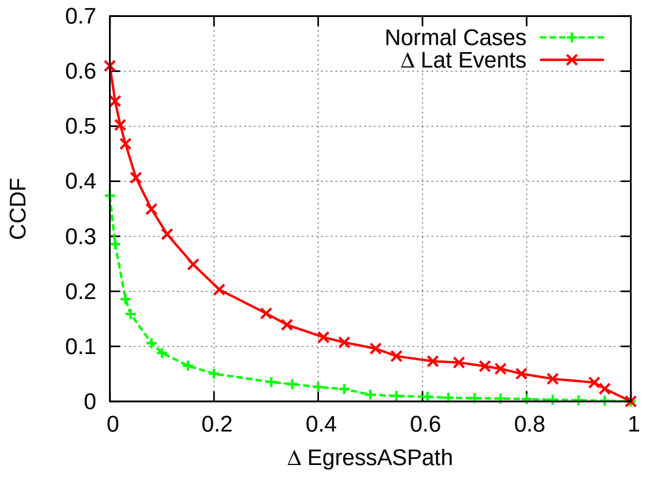


Figure 3.9: Egress router and AS path shifts ($\Delta EgressASpath$)

Thresholds (Percentile)	**(0.025, 0.05) 85th**	**(0.06, 0.09) 90th**
Ingress	13.9%	12.6%
Egress/AS-path	19.6%	17.4%
Both	23.7%	17.6%
Unknown	42.7%	52.5%

Table 3.7: Classification of Events with $\Delta Lat \geq 0.4$

thresholds, 23.7% of the events are associated with large shifts in both the ingress routers and the egress/AS-path; 13.9% of the events are associated with ingress-router shifts; 19.6% of the events are associated with shifts in the egress router and AS path; and 42.7% of the events fall into the unknown category. We also show results using the 90th-percentile thresholds.

Note that around half of the events fall into the unknown category, where we could not correlate latency increases with large, visible changes to interdomain routing. Potential explanations include AS-level routing changes on the forward path (from the client to the front-end server) that do not affect where traffic enters Google's CDN network. Intradomain routing changes in individual ASes could also cause increases in round-trip time without changing the ingress router, egress router, or AS path seen by the CDN. Finally, congestion along either the forward or reverse path could be responsible. These results suggest that CDNs should supplement BGP and traffic data with finer-grain measurements of the IP-level forwarding path (e.g., using traceroute and reverse traceroute [31]) both for better accuracy in diagnosing latency increases and to drive new BGP path-selection techniques that make routing decisions based on direct observations of performance.

3.6 Case Studies

For a better understanding of large latency increases, we explore several events in greater detail. These case studies illustrate the general challenges CDNs face in minimizing wide-area latency and point to directions for future work.

3.6.1 Latency-Map Inaccuracies

During one day in June 2010, an ISP in the United States (which we defined as a region in the previous sections) saw the average round-trip time increase by 111 msec. Our analysis shows that the RTT increased because of a shift of traffic to different front-end servers; in particular, ΔFE was 1.01. These shifts were triggered primarily by a change in the latency map; in particular, $\Delta LatMap$ was 0.90. Looking at the latency map in more detail revealed the reason for the change. On the first day, 78% of client requests were directed to front-end servers in the United States, and 22% were directed to servers in Europe. In contrast, on the second day, all requests were directed to front-end servers in Europe. Hence, the average latency increased because the clients were directed to servers that were further away. The situation was temporary, and the clients were soon directed to closer front-end servers.

This case study points to the challenges of identifying the closest servers and using DNS to direct clients to servers—topics explored by several other research studies [32, 30, 33, 34, 35]. Clients do not necessarily reside near their local DNS servers, especially with the increasing use of services like GoogleDNS and OpenDNS. Similarly, client IP addresses do not necessarily fall in the same IP prefix as their local DNS server. Further, DNS caching causes the local DNS server to return the same IP address to many clients over a period of time. All of these limitations of DNS make it difficult for a CDN to exert fine-grain control over server selection. Recent work at the IETF proposes extensions to DNS so requests from local DNS servers include the client's IP address [36], which should go a long way toward addressing this problem. Still, further research on efficient measurement techniques and efficient, fine-grain control over server selection would be very useful.

3.6.2 Flash Crowd Leads to Load Balancing to Distant Front-End Servers

As another example, we saw the average round-trip time double for an ISP in Malaysia. The RTT increase was caused by a traffic shift to different front-end servers; in particular, ΔFE was 0.979. To understand why, we looked at the metrics for front-end server changes. First, we noticed that $\Delta LatMap$ was 0.005, suggesting that changes in the latency map were not responsible. Second, we observed that $\Delta FEDist = 0.34$ and $\Delta LoadBal = 0.323$, suggesting that load balancing was responsible for the shift in traffic. Looking at the client request rate, we noticed that the requests per day jumped significantly from the first day to the second; in particular, $RPD_2/RPD_1 = 2.5$. On the first day, all requests were served from front-end servers close to the clients; however, on the second day, 40% of requests were directed to alternate front-end servers that were further way. This led to a large increase in the average round-trip time for the whole region.

This case study points to a general limitation of relying on round-trip times as a measure of client performance. If, on the second day, Google's CDN had directed all client requests to the closest front-end server, the user-perceived performance would likely have been *worse*. Sending more requests to an already-overloaded server would lead to slow downloads for a very large number of clients. Directing some requests to another server—even one that is further away—can result in higher throughput for the clients, including the clients using the remote front-end server. Understanding these effects requires more detailed measurements of download performance, and accurate ways to predict the impact of alternate load-balancing strategies of client performance. We believe these are exciting avenues for future work, to enable CDNs to handle flash crowds and other shifts in user demand as effectively as possible.

3.6.3 Shift to Ingress Router Further Away from the Front-End Server

On day in June 2010, an ISP in Iran experienced an increase of 387 msec in the average RTT. We first determined that the RTT was mainly caused by a large increase in latency to reach a particular front-end server in western Europe. This front-end server handled 65% of the requests on both days. However, ΔLat_i for this server was 0.73, meaning 73% of the increase in RTT was caused by an increase in latency to reach this front-end server. Looking at the routing changes, we saw a $\Delta Ingress$ of 0.38. Analyzing the traffic by ingress router, we found that, on the first day, all of the traffic to this front-end server entered the CDN's network at a nearby ingress router in western Europe. However, on the second day, nearly 40% of the traffic entered at different locations that were further away—21% in eastern Europe and 17% of traffic in the United States. Thus, the increase in RTT was likely caused by extra latency between the ingress router and the front-end server, and perhaps also by changes in latency for the clients to reach these ingress routers.

This case study points to a larger difficulty in controlling *inbound* traffic using BGP. To balance load over the ingress routers, and generally reduce latency, a large AS typically announces its prefixes at many locations. This allows other ASes to select interdomain routes with short AS paths and nearby peering locations. However, an AS has relatively little control over whether other ASes can (and do) make good decisions. In some cases, a CDN may be able to use the Multiple Exit Discriminator (MED) attribute in BGP to control how individual neighbor ASes direct traffic, or perform selective AS prepending or selective prefix announcements to make some entry points more attractive than others. Still, this is an area that is ripe for future research, to give CDNs more control over how clients reach their services.

3.6.4 Shorter AS Paths Not Always Better

On another day in June 2010, an ISP in Mauritius experienced a 113 msec increase in the average round-trip time. On both days, most client requests were handled by a front-end server in Asia—60% on the first day and 74% on the second day. However, on the second day, the latency to reach this front-end server increased substantially. Looking at the routing data, we see that traffic shifted to a different egress router and AS path. On the first day, 56% of the traffic left Google's CDN's network in Asia. On the second day, this number dropped to 10%, and nearly two-thirds of the traffic left the network in Europe over a *shorter* AS path. Presumably, upon learning a BGP route with a shorter AS path, the routers preferred this route over the "longer" path through Asia. However, AS-path length is (at best) loosely correlated with round-trip time, and in this case the "shorter" path had a much higher latency.

This case study points to a larger problem with today's interdomain routing system—routing decisions do not consider performance. The BGP decision process uses AS-path length as a (very) crude measure of performance, rather than considering measurements of actual performance along the end-to-end paths. Future work could explore lightweight techniques for measuring the performance along different interdomain paths, including the paths not currently selected for carrying traffic to clients. For example, recent work [37] introduces a "route injection" mechanism for sampling the performance on alternative paths. Once path performance is known, CDNs can optimize interdomain path selection based on performance, load, and cost. However, large CDNs with their own backbone network introduce two interesting twists on the problem of intelligent route control. First, the CDN selects interdomain routes at *multiple* egress points, rather than a single location. Second, the CDN can *jointly* control server selection and route selection for much greater flexibility in directing traffic.

3.7 Future Research Directions

In this section, we briefly discuss several natural directions for future work on diagnosing wide-area latency increases for CDNs.

Direct extensions of our measurement study: First, we plan to extend our methodology in Section 3.3 to distinguish between routing changes that affect the egress router from those that only change the AS path. Second, as discussed at the end of Section 3.5.3, we plan to further explore the unexplained shifts in traffic from one front-end server to another. We suspect that some of these shifts are caused by a relatively small fraction of traffic shifting to a much further away front-end server. To analyze this further, we plan to incorporate the RTT differences between front-end servers as part of our metrics for studying FE changes. Third, our case studies in Section 3.6 required manual exploration, after automatically computing the various metrics. We plan to conduct more case studies and automate the analysis to generate reports for the network operators.

More accurate diagnosis: First, we plan to work with the groups that collect the measurement data to provide the data on a smaller timescale (to enable finer-grain analysis) and in real time (to enable real-time analysis). Second, we plan to explore better ways to track the performance data (including RTT and RPD) separately for each ingress router and egress/AS-path. Currently, the choice of ingress and egress routers are not visible to the front-end servers, where the performance data are collected. Third, we will explore techniques for correlating across latency increases affecting multiple customer regions. For example, correlating across interdomain routing changes that affect the AS paths for multiple client prefixes may enable us to better identify the root cause [11].

Incorporating additional data sets: We plan to investigate techniques for improving the visibility of the routing and performance changes from outside the CDN network. For example, active measurements—such as performance probes and traceroute (including both forward and reverse traceroute [31])—would help explain the "unknown" category for the ΔLat events, which we could not correlate with visible routing changes. In addition, measurements from the front-end servers could help estimate the performance of alternate paths, to drive changes to the CDN's routing decisions to avoid interdomain paths offering poor performance.

3.8 Related Work

CDNs have been widely deployed to serve Web content. In these systems, clients are directed to different servers to reduce latency and balance load. Our characterization reveals the main causes of high latency between the clients and the servers.

An early work in [32] studied the effectiveness of DNS redirection and URL rewriting in improving client performance. This work characterizes the size and the number of the web objects CDNs served, the number of distinct IP addresses used in DNS redirection, and content download time, and compared the performance for a number of CDN networks. Recent work in [38] evaluated the performance of two large-scale CDNs—Akamai and LimeLight. Instead of measuring CDNs from end hosts, we design and evaluate techniques for a CDN to diagnose wide-area latency problems, using readily-available traffic, performance, and routing data.

WhyHigh [39] combines active measurements with routing and traffic data to identify causes of persistent performance problems for some CDN clients. For example, WhyHigh identifies configuration problems and side-effects of traffic engineering that lead some clients to much higher latency than others in the same region. In contrast, our work focuses on detecting and diagnosing large *changes* in performance over time, and also considers several causes of traffic shifts from one front-end server to another. WISE [40] predicts the effects of possible configuration and deployment changes in the CDN. Our work is complementary in that, instead of studying planned maintenance and operations, we study how to detect and diagnose unplanned increases in latency.

PlanetSeer [21] uses passive monitoring to detect network path anomalies in the wide-area, and correlates active probes to characterize these anomalies (temporal vs. persistent, loops, routing changes). The focus of our work is different in that, instead of characterizing the end-to-end *effects* of performance anomalies, we study how to classify them according to the *causes*.

3.9 Summary

The Internet is increasingly a platform for users to access online services hosted on servers distributed throughout the world. Today, ensuring good user-perceived performance is a challenging task for the operators of large Content Distribution Networks (CDNs). In this chapter, we presented techniques for automatically classifying large changes in wide-area latency for CDNs, and the results from applying our methodology to traffic, routing, and performance data from Google. Our techniques enable network operators to learn quickly about significant changes in user-perceived performance for accessing their services, and adjust their routing and server-selection policies to alleviate the problem.

Using only measurement data readily available to the CDN, we can automatically trace latency changes to shifts in traffic to different front-end servers (due to load-balancing policies or changes in the CDN's own view of the closest server) and changes in the interdomain paths (to and from the clients). Our analysis and case studies suggest exciting avenues for future research to make the Internet a better platform for accessing and managing online services.

Chapter 4

Next-Hop Routing: Routing Protocol Designed for Better Performance

4.1 Introduction

The Border Gateway Protocol (BGP) stitches a single global Internet together out of smaller networks with diverse policy objectives. BGP is plagued by poor network performance, as well as security vulnerabilities, configuration errors, software bugs, and more. BGP does not react to serious performance and reachability problems in the data plane, which are, in fact, often utterly invisible to the routing protocol. BGP route fluctuations can lead to high packet loss, intermittent loss of connectivity, and increased path latency [2]. Remarkably, almost half of the performance problems with VoIP are the result of BGP route fluctuations [41].

The research and standards communities have proposed numerous ways to fix BGP. These proposals generally fall into two categories: (1) enhancements to BGP (*e.g.*, to improve convergence [42, 43] and security [44, 45]); and (2) alternative architectures that take a "clean-slate approach" to interdomain routing (see [46, 47, 48, 49]). However, these proposals face serious practical obstacles. On the one hand, enhancements to BGP add mechanisms to an already complex protocol, which can introduce new problems (*e.g.*, configuration errors, protocol convergence issues, and new attack vectors). On the other hand, alternative architectures are difficult, often even impossible, to deploy incrementally.

We present a new way to fix interdomain routing. We introduce a novel routing architecture called "next-hop routing", which is based on two guiding principles: (1) Rather than extending or replacing BGP, *simplifying* the routing protocol by *constraining* how routes are selected and exported; and (2) Handling issues such as data-plane performance and security where they belong—*outside* the routing protocol. We argue that next-hop routing is sufficiently expressive to realize network operators' goals. Our simulations show that next-hop routing significantly improves performance, without compromising other global objectives.

4.1.1 What's Wrong with BGP?

BGP gives network operators great expressiveness in specifying routing policies. Unfortunately, this expressiveness is not free. Today's routing system faces numerous serious problems, including:

- **Slow convergence and divergence.** BGP's convergence to a "stable" routing state can be slow, resulting in poor network performance. Route fluctuations during convergence can lead to high packet loss, intermittent loss of connectivity, and increased path latency [2]. (Remarkably, almost half of the performance problems of VoIP are the result of slow BGP convergence [41].) Worse yet, BGP dynamics can even enter inescapable oscillations, never reaching stability [50, 51].

- **Poor data-plane performance.** BGP does not select paths based on end-to-end performance, but rather based on routing policies and path length. In fact, BGP does not react to serious performance and reachability problems in the data plane, since these are not visible to the routing protocol. In addition, a BGP-speaking router selects a single best path for each destination prefix, rather than capitalizing on the path diversity in the Internet topology.

- **Vulnerability to attacks.** BGP is notoriously vulnerable to malicious [52, 45] or economically-driven [53, 54] attacks. Indeed, BGP makes it extremely easy for an AS to manipulate the routing announcements so as to attract traffic to and through its network [55]. This weakness of BGP can be exploited for eavesdropping, tampering, packet dropping, and also for economic reasons (*e.g.*, increasing an AS's revenue from transiting traffic). Every few years, a major incident exposes just how incredibly vulnerable BGP is (*e.g.*, Pakistan Telecom hijacking YouTube traffic [6]).

- **Configuration errors and software bugs.** Today's routers have a bewildering array of configuration options for selecting and exporting routes, making configuration errors [56, 57] and software bugs [58, 59, 60] common phenomena. In addition, techniques for reducing convergence delay introduce additional timers and configuration options to the router software, exacerbating an already complex system. Over the past several years, we have seen quite a few serious Internet outages caused by programmer and operator errors [60, 61, 62].

4.1.2 Simplicity through Next-Hop Routing

Many of BGP's performance problems relate to how ASes base routing decisions on the sequence of ASes along a path. To avoid the "count-to-infinity" problems that plagued earlier distance-vector protocols, BGP was designed as a *path-vector* protocol that includes the AS-PATH attribute in each route announcement. This allows an AS to quickly detect and avoid paths that contain loops.

However, the AS-PATH is increasingly used for far more than loop detection and, in particular, network operators apply complex regular expressions to the AS-PATH to express local preferences over routes. Unfortunately, routing policies that consider the entire AS-PATH can lead to long convergence time, and can even result in overall protocol divergence.

We argue that such reliance on the AS-PATH is more a hindrance than a help in achieving good network performance. Instead, we advocate next-hop routing—that an AS rank and export paths based *solely* on the next-hop AS en route to each destination prefix, and apply a simple "consistent filtering" rule [63] when exporting routes. That is, we relegate the AS-PATH to its traditional role in loop detection. We show that next-hop routing achieves significantly better network performance than traditional path-based routing in two important respects:

Better convergence. We show that next-hop routing converges much more quickly to a "stable" routing configuration than conventional BGP. In addition, the convergence process involves much fewer update messages and forwarding changes. Our simulation results establish that next-hop routing is especially effective at preventing the most serious BGP convergence problems.

More amenable to multipath routing. As mentioned earlier, today's BGP-speaking routers select a single path for each destination, rather than capitalizing on the path diversity in the Internet. We show that next-hop routing naturally supports multipath routing, leading to many benefits, *e.g.*, improved availability, better recovery from failures, load balancing, customized route selection.

4.1.3 Path-quality Monitoring and Security

Today's BGP provides network operators with very unsatisfactory means for handling data-plane performance issues and security threats, as reflected in poor network performance and in serious security breaches. We argue that such important objectives should be handled *outside* the routing protocol [64], and propose specific solutions in two key areas:

Performance-driven routing. Today's BGP-speaking routers prefer shorter AS-PATHs as an indirect way to improve end-to-end performance. However, AS-PATH length only loosely correlates with end-to-end propagation delay and does not reflect other performance metrics of interest, *e.g.*, throughput, latency, and loss [65]. Consequently, BGP is unable to cope with serious performance and reachability problems in the data plane. We argue that, to achieve good network performance, route selection should be based on direct observations of path quality, rather than the length of the AS-PATH.

We propose leveraging two mechanisms: end-to-end monitoring and multipath routing (see above). Neither of these mechanisms is provided by today's BGP. Instead, ASes should adapt next-hop rankings, as well as how traffic is split between multiple "next hops," based on path-quality monitoring in the data plane. We describe techniques for stub ASes, online service providers, and transit ISPs to monitor path performance.

Security. We believe that relying on BGP for data-plane security is misguided. Instead, these guarantees should be assured in other (end-to-end) ways, such as encryption and authentication. We show that next-hop routing significantly reduces BGP's attack surface. Specifically, next-hop routing makes ASes immune to virtually all AS-PATH-based attacks (*e.g.*, path-shortening attacks [55]). Next-hop routing also removes incentives for rational ASes to manipulate the routes.

4.1.4 Roadmap

We present the details of next-hop routing in Section 4.2. Since next-hop routing is a broad routing architecture, our evaluation takes many forms, including (i) simulation experiments (to measure convergence time), (ii) protocol design (to illustrate simple ways to support multipath routing and path-quality monitoring), and (iii) qualitative arguments (about the reduced attack surface and techniques for traffic engineering).

Section 4.3 shows that next-hop routing significantly reduces convergence time and router overhead. Section 4.4 shows how next-hop routing greatly simplifies support for multipath routing. Section 4.5 shows that next-hop routing reduces the attack surface for BGP and provides incentives for rational ASes to participate honestly in the protocol. We present techniques for traffic management (i.e., path-quality monitoring and traffic engineering) in Section 4.6. We present related work in Section 4.7 and conclude in Section 4.8.

4.2 Next-Hop Routing Architecture

We now present our next-hop routing architecture, which consists of two components: (1) three simple rules that constrain how routes are selected and exported in the BGP decision process; and (2) external mechanisms for performance-driven routing based on end-to-end data-plane monitoring, and for security. Our focus in this section is on the first component—the next-hop routing rules. We discuss security and performance-driven routing in Sections 4.5 and 4.6, respectively.

In today's Internet, the bilateral business contracts ASes sign with their *immediate* neighbors play a crucial role in determining their routing policies (see [66] and Appendix 4.9.2): ASes tend to prefer revenue-generating routes through customers over routes (for which they pay) through their providers; ASes avoid carrying traffic through their providers. Under next-hop routing, ASes rank and export paths *solely* based on the next-hop AS en route to each destination prefix, but an AS has full control over which immediate neighbors carry its traffic and direct traffic through it. Next-hop routing thus allows ASes sufficient expressiveness to realize their business policies. (Indeed, these common routing practices can be realized with next-hop routing.)

4.2.1 The Existing BGP Decision Process

BGP-speaking routers typically receive multiple routes to the same destination prefix. There are three steps a BGP router uses to process route advertisements:

1. an **import policy** determines which routes should be filtered (and hence eliminated from consideration). After the filtering process, a "local preference" is assigned to each route based on the import policy;
2. a **decision process** selects the most desirable route;
3. an **export policy** determines which of the neighbors learn the chosen route.

1. Highest local preference
2. Lowest AS path length
3. Lowest origin type
4. Lowest MED (with *same* next-hop AS)
5. eBGP-learned over iBGP-learned
6. Lowest IGP path cost to egress router
7. Prioritize older routes
8. Lowest router ID of BGP speaker

Table 4.1: Steps in the BGP decision process

The decision process consists of an ordered list of attributes across which routes are compared. Here, we focus on the following important steps in the BGP decision process (see Table 4.1 for more information, and [1] for more details about the BGP decision process, including a full list of decision steps):

- **Prefer higher local preference (LocalPref)**. The BGP LocalPref attribute enables operators to express rich, engineering- and business-motivated preferences over routes, thus potentially choosing a longer AS-PATH over a shorter route;
- **Prefer shorter routes.** Ties between routes that share the highest LocalPref are broken in favor of routes with the lowest AS-PATH length;
- **Prioritize old routes over new ones.** This decision step is implemented in some routers [67] but not always enabled by default.

4.2.2 Next-Hop Routing Rules

We now specify our three next-hop routing rules.

Rule I: Use next-hop rankings. Configure rankings of routes based *only* on the immediate next-hop AS en route to each destination (*e.g.*, to prefer customer-learned routes over provider-learned routes). Ties in the rankings of next-hops are permitted.

Rule II: Prioritize current route. To minimize path exploration, when faced with a choice between the "old" (current) route and an equally-good (in terms of next-hop) new one, re-select the old route.

Rule III: Consistently export [63]. If a route P is exportable to a neighboring AS i, then so must be all routes that are more highly ranked than P. Intuitively, Consistent Export prevents undesirable phenomena

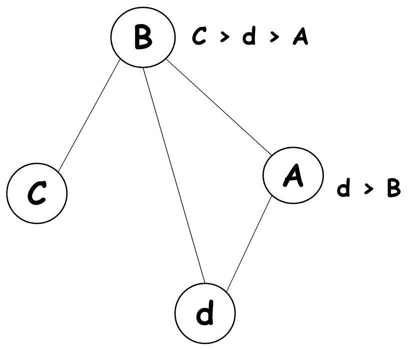


Figure 4.1: ASes A, B and C wish to send traffic to AS d. All three ASes have next-hop rankings, *e.g.*, A prefers sending traffic directly to d over sending traffic through B. AS B is willing to export route BAd, but not the more preferred route Bd, to C. Hence, if B changes its route from BAd to the Bd, it will stop announcing its route to C and thus disconnect C from the destination d.

as in Figure 4.1, where an AS disconnects a neighbor from a destination by selecting a better route for itself.

4.2.3 Implementing the Next-Hop Routing Rules

Next-hop routing can be deployed incrementally by individual ASes, without changing the BGP protocol. In addition, next-hop routing greatly simplifies the protocol configuration interface and router software implementation. We envision three main deployment scenarios that offer increasing benefits in this respect.

Configure today's routers to obey the next-hop rules.

Network operators can switch from path-based routing to next-hop routing by locally configuring their existing routers in accordance with the next-hop routing rules. This is achieved via the following three actions: (i) applying only next-hop rankings when assigning LocalPref, (ii) disabling the AS-path length step in the BGP decision process (as supported on most commercial routers), and (iii) selecting eBGP export policies that obey consistent filtering. The advantage of this approach is that an AS can readily deploy next-hop routing without any changes to the underlying equipment and without any support from its neighbors[1].

[1]Peering contracts usually requires an AS to export routes with the same AS-PATH length across all peering points with a given neighbor, which allows the neighbor to employ hot-potato routing. However, the neighbor can take advantage of the BGP decision process by sending routes with different lengths to influence how traffic enters the network. With next-hop routing, this is not possible since the step of using shorter AS-PATH length is disabled in the decision process, and all routes are considered *equally good*

Create a simpler configuration interface.

The simplicity of next-hop routing enables the design of a simpler BGP configuration interface, that *enforces* next-hop routing, in which: (i) the path ranking is configured simply by specifying an order (that need not be necessarily strict) over neighboring ASes (routers) and applied to a set of destination prefixes; (ii) the AS-PATH length step in the BGP decision process is disabled by default, and (iii) for each neighboring AS A (router), the local export policy with respect to A is configured simply by specifying a *single* neighboring AS B; only routes that have B, or some more preferred neighbor, as a next hop are exportable to A. (If an AS wishes to make all routes exportable to its neighbor it can do so by specifying a unique symbol.)

This simple configuration interface requires less training for network operators and would lead to fewer configuration errors. In addition, the router software would be simpler (and, hence, have fewer bugs) since the policy configuration would be easier to parse; this is important, as many bugs in routing software lie in the complex configuration parsing code [58].

Build router software that (only) supports next-hop routing.

Router software that *only* supports next-hop routing would not only have much fewer configuration options, but also much fewer execution paths (for applying routing policy) and a simpler/shorter decision process. We later show that next-hop routing has much better convergence properties than path-based routing. Hence, some complex mechanisms for reducing convergence time (such as the min-route-advertisement timer and route-flap damping) may not be needed. Instead, router software could be extended in newer, more fruitful, directions, such as support for multipath routing.

4.3 Better Routing Convergence

Intuitively, BGP can converge slowly for two main reasons: (1) small and faraway routing changes can lead an AS to select a new next-hop, thus leading to a chain reaction of subsequent routing changes; and (2) inconsistencies between path rankings and route export policies can lead an AS to disconnect other ASes from a destination when selecting a better route for itself, pushing them to seek alternate routes (see Figure 4.1).

irrespective of their AS-PATH length.

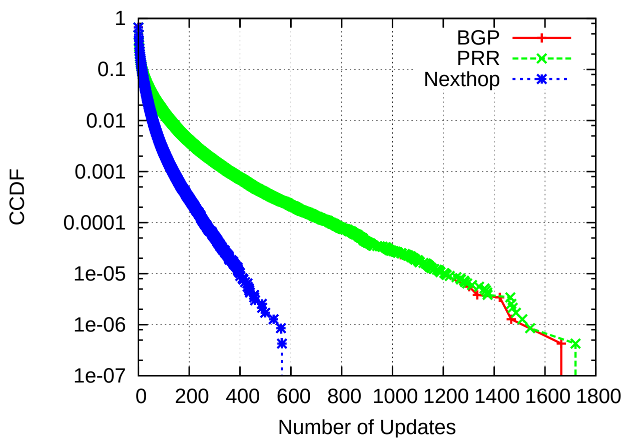


Figure 4.2: Fraction of non-stub ASes experiencing more than x update messages after a link failure.

Our next-hop routing rules prevent these scenarios; next-hop rankings guarantee that remote routing changes do not drive an AS to change its next hop; Consistent Export guarantees that when selecting a better route, an AS *never* disconnects other ASes from a destination. We now present experimental evidence that next-hop routing converges *quickly* to a "stable" routing configuration.

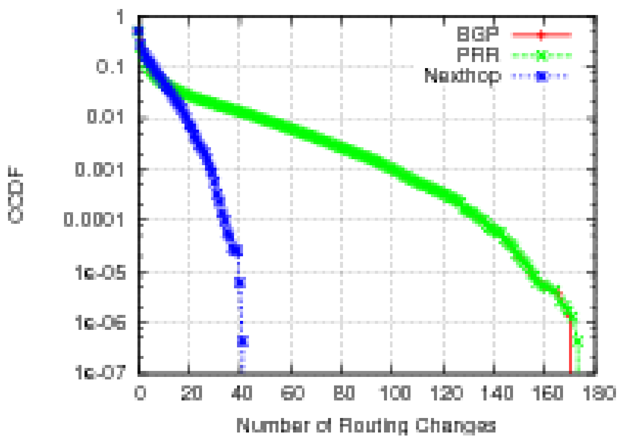
Metrics. We focus on three metrics for measuring performance during the convergence process:

- **# forwarding changes**, that is, the number of times ASes change their choices of next hop during the convergence process (summed across all vantage point ASes);
- **# routing changes**, that is, the number of times ASes' routes change during the convergence process (summed across all vantage point ASes);
- **# BGP updates** transmitted during the convergence process (summed across all vantage point ASes).

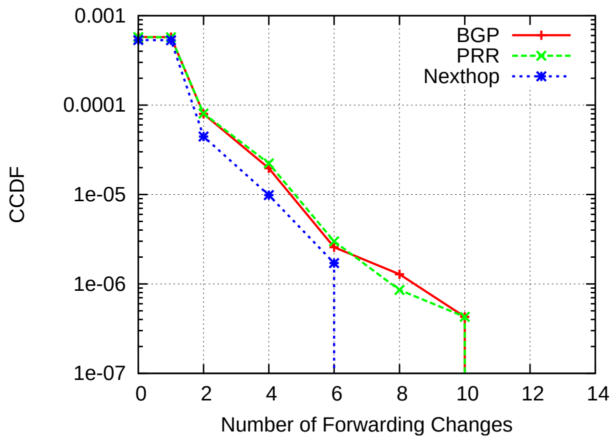
Our experiments show that next-hop routing significantly reduces the number of update messages, routing changes, and forwarding changes, under various network events and vantage points.

4.3.1 Simulation Setup

Topology. We use the Cyclops [68] AS-level Internet topology on Jan 1, 2010. Each AS is represented by one router, and the links between ASes are annotated with business relationship. The topology contains a total of 33,976 ASes, of which 4,670 are non-stubs, that is, ASes that have customers of their own, and the rest are stubs. The topology contains 54,786 customer-provider links and 43,888 peer-peer links.



(a) # routing changes



(b) # forwarding changes

Figure 4.3: Fraction of non-stub ASes experiencing more than x routing changes/forwarding changes after a link failure

Protocols. We evaluate BGP (standard decision process), PRR (Prefer Recent Route) [67] where the final tie-breaking step prefers the current best route over new routes, and next-hop routing. For all these protocols, we follow the Gao-Rexford import and export conditions [66] (see Appendix 4.9.2), where ASes prioritize customer-learned routes over peer-learned routes over provider-learned routes. BGP and PRR prefer shorter routes to longer ones within each category (customer-/peer-/provider-learned). All three protocols obey the Consistent Export requirement. Comparing the three protocols allows us to quantify the importance of next-hop ranking (only in next-hop routing) and prioritizing the current route (in both next-hop ranking and PRR).

Simulator. We use the C-BGP [69] simulator. Note that C-BGP does not support the MRAI (Minimum Route Advertisement Interval) timer, which rate-limits the updates sent on each session. Although the BGP RFC recommends the default MRAI of 30 seconds [1], router vendors [70] and the IETF [71] advocate using much smaller MRAIs (e.g., a few seconds) or removing the timer entirely, due to the performance requirements of interactive applications. Thus, our simulation results without MRAI settings should be a reasonable estimation of forwarding changes, routing changes, and BGP update messages as the Internet moves away from large MRAI timers. In fact, our experimental and theoretical results below suggest that next-hop routing could allow ASes to remove the MRAI timer entirely. Since C-BGP does not produce accurate estimates of convergence *time*, we do not present results for this metric.

Events. We consider three events: prefix announcement, link failure, and link recovery. We first inject a prefix from a randomly selected multi-homed stub AS, next randomly fail a link between a stub AS and one of its providers, and then recover the failed link. We wait for the routing protocol to converge after each step, before moving on to the next step. For each experiment, we compute all the metrics at selected vantage points (see below). We repeat this experiment for 500 randomly-chosen multi-homed stub ASes.

Vantage points. We observe the number of update messages, routing changes, and forwarding changes from all 4,670 non-stub ASes and from randomly chosen 5,000 stub ASes as vantage points.

4.3.2 Convergence Results

We now present our results for the failure of a link connecting an AS to the rest of the Internet. Results for the other events we evaluate are similar.

Comparison: BGP updates at non-stubs. Figure 4.2 plots the distribution of the number of update messages seen at non-stub ASes. Around one-third of the ASes on the Internet see no routing changes after a link failure. For the ASes which experience a routing changes, the average number of updates received at each AS is 11.7 for BGP, 11.8 for PRR, and 8.0 for next-hop BGP. Since many ASes see little or no effects after any event, we plot the complementary cumulative distribution function (CCDF) to focus on ASes that experience many update messages.

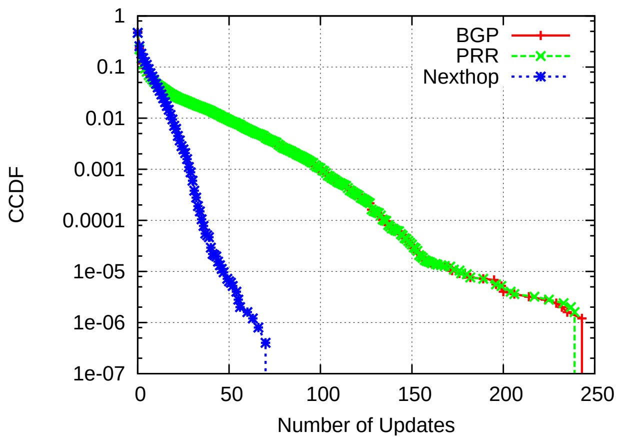
Under BGP (upper curve in Figure 4.2), some non-stub ASes receive thousands of update messages. Interestingly, the larger, better-connected ASes tend to receive more update messages, presumably because they have many more neighbors exporting routes to them. The most-affected non-stub AS receives 1698 update messages. PRR slightly reduces this number (middle curve in Figure 4.2, which mostly overlaps with the BGP curve, except at the tail), by avoiding unnecessary path exploration. Next-hop routing leads to significant improvement (bottom curve). The much smaller number of update messages suggests that next-hop routing would allow ASes to remove the MRAI timer (as is the case in our experiments) without introducing a large number of messages.

Recall that next-hop routing protocol is composed of three parts: next-hop ranking, prefer old route, and consistent export. While the step of consistent export is applied in all protocols of BGP, PRR, and next-hop, the gap between the curves of BGP and PRR in Figure 4.2 shows the minor benefits of the PRR step in next-hop routing. By contrast, the gap between the curves of PRR and next-hop in the figure clearly demonstrates the improvement by using next-hop ranking.

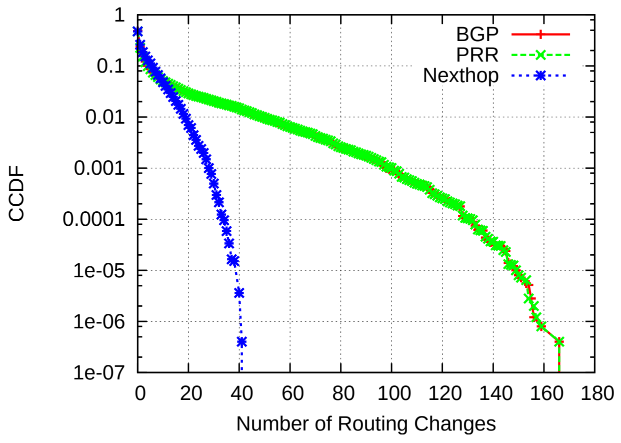
Comparison: routing/forwarding changes at non-stubs. Next-hop routing also greatly reduces the number of routing and forwarding changes for non-stub ASes (see Figure 4.3). For the ASes which experience changes, the average number of routing changes is 4.99 for BGP, 5.03 for PRR, and only 3.95 for next-hop BGP. We get similar results for the average number of forwarding changes: 2.36 for BGP, 2.37 for PRR, and 2.20 for next-hop routing. In fact, none of the non-stub ASes experience more than *six* forwarding changes under next-hop routing for these experiments. In contrast, ASes experience as many as *ten* forwarding changes when preferring a shorter AS-PATH over staying with the same next-hop AS, as in PRR.

Comparison: stubs. Figure 4.4 plots the distribution of the number of update messages and routing changes as seen at stub ASes. Observe that next-hop routing once again leads to significant improvement over both BGP and PRR.

The 0.1% position. The CCDF plots, while useful for illustrating the diverse experiences of ASes during convergence, are unwieldy for head-to-head comparisons of the protocols across different events and convergence metrics. Yet, with such a skewed distribution, the mean and the median are not especially meaningful. So, we focus on the experience of the AS at the 0.1% position in the CCDF plots, allowing us to focus on the (non-stub) ASes that are affected the most link failures without allowing one outlier AS to bias the results. We summarize the data in bar charts in Figure 4.5 and Figure 4.6. The bars for "Link Failure" correspond to the $y = 0.001$ position in Figures 4.3(a) and 4.4(a), respectively. We do not plot the number of forwarding changes since, for most experiments, the 0.1th-percentile AS experienced at most one forwarding change. We get similar results by comparing the number of update messages and routing



(a) # updates

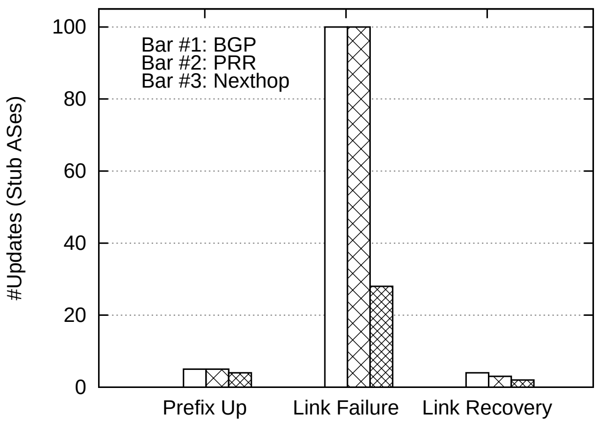


(b) # routing changes

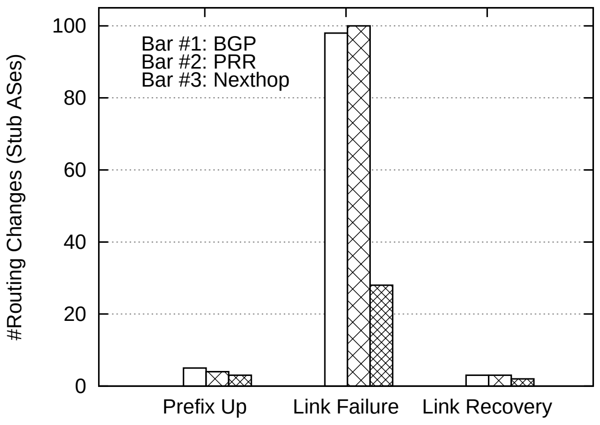
Figure 4.4: Fraction of stub ASes experiencing more than x update messages/routing changes after a link failure

changes during various events for the stub ASes.

Conclusions. Our results show that next-hop routing offers reductions in the number of update messages and routing changes across a range of network events. These results also illustrate how next-hop routing is especially effective at preventing the most serious convergence problems—where an AS experiences hundreds of routing changes and tens of forwarding changes. This not only reduces the performance degradation experienced by the data traffic, but also significantly reduces the overhead for disseminating BGP update messages.



(a) # update messages



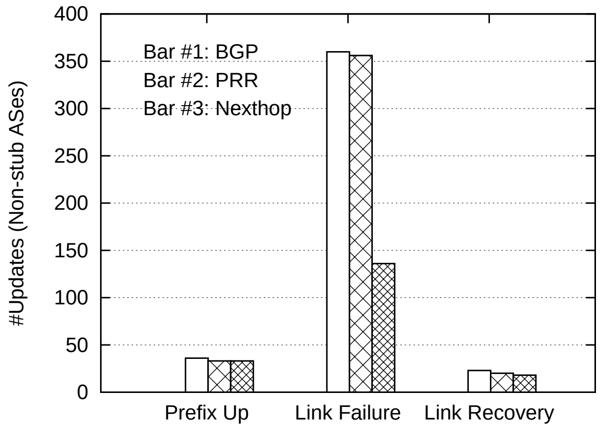
(b) # routing changes

Figure 4.5: Number of update messages and routing changes for stub ASes

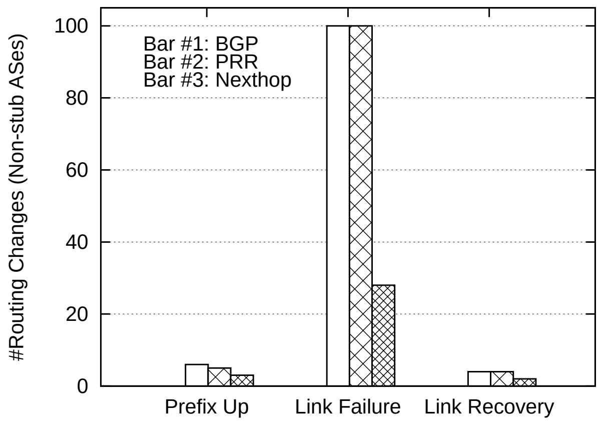
4.4 Multipath Interdomain Routing

Recently, there has been a surge of interest in multipath interdomain routing (see, *e.g.*, [72, 48, 42, 73, 74]). We now discuss how next-hop routing can make multipath routing more scalable and incrementally deployable.

Naive BGP-based multipath routing schemes can be unscalable, due to the need to disseminate and store multiple routes. For the topology in Figure 4.7, consider the naive multipath protocol where nodes announce all available routes to neighboring nodes. Observe that ASes A and B, that each have two routes to the destination d, will both announce two routes to AS C. AS C will then announce four routes to AS D, and so on. This can easily result in state explosion and in excessive transmission of update messages.



(a) # update messages



(b) # routing changes

Figure 4.6: Number of update messages and routing changes for non-stub ASes

We show that next-hop routing is more amenable to multipath than path-based routing. The key observation is that under next-hop routing, a node need not learn a neighboring node's multiple paths, but merely learn enough to avoid loops. If AS C in Figure 4.7 has a next-hop ranking of routes then, to enable C to detect loops, AS A (or B) can merely send C an (unordered) *list* of all the ASes its (multiple) routes traverse. BGP allows the aggregation of routes into one such AS-SET [75], that summarizes the AS-PATH attributes of all the individual routes. Thus, BGP route aggregation, used to keep BGP routing tables manageable in other contexts, can also be used to greatly mitigate the cost of multipath next-hop routing.

Hence, next-hop routing lowers the barrier for making multipath routing a reality. Capitalizing on multipath routing can yield the following benefits:

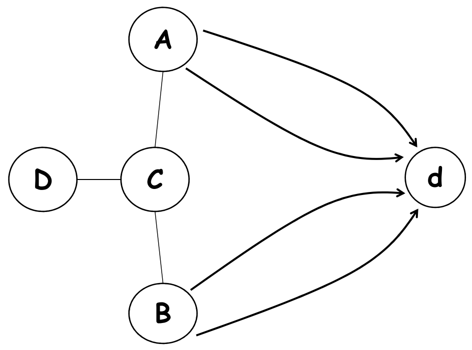


Figure 4.7: ASes A and B both announce two routes to AS C, and C then announces four routes to AS D, and so on. Clearly, this can result in state explosion.

Availability. Multipath routing increases the likelihood that an AS has at least one working path.

Failure recovery. An AS with multiple next-hops can react *immediately* to a failure in one outgoing link by sending traffic along another (and not wait for the protocol to re-converge).

Load balancing. An AS could have multiple next-hops (with equal local preference) and decide whether and how much traffic to direct through each next-hop (*e.g.*, based on data-plane monitoring). Conventional techniques, such as hashing on fields in the IP header, can ensure that packets of the same flow traverse the same path, to prevent out-of-order packet delivery.

4.5 Security

We now present the advantages of next-hop routing over path-based routing with BGP in terms of security: (1) significantly reduced attack surface; and (2) incentive compatibility. We then explain how additional security mechanisms can be used to secure the routing system from the remaining forms of attacks.

4.5.1 Reduced Attack Surface

Some of the most dangerous attacks on BGP are based on "lying" about the length of an AS's path. Indeed, an AS can attract much traffic by announcing a shorter route than that it really has [55], thus launching so called blackhole or interception attacks for tampering, dropping packets, eavesdropping, *etc.*

Under next-hop routing, ASes do not consider the AS-PATH when making routing decisions—beyond the first hop, which cannot be forged!—and so an AS no longer benefits from such attacks. Thus, next-hop routing significantly reduces BGP's attack surface. In fact, the only effective AS-PATH-based attacks on next-hop routing are attacks that trigger BGP's loop detection mechanism (that are clearly also effective against path-based routing with BGP).

4.5.2 Incentive Compatible

BGP is not incentive compatible, in the sense that ASes might have incentive not to participate honestly in the routing protocol [53, 76, 54]. In contrast, [53] shows that next-hop routing is incentive-compatible. Specifically, in the Gao-Rexford framework (see Appendix 4.9.2), an AS cannot get a "better" next-hop to a destination by "deviating" from BGP (*e.g.*, announcing bogus routes, reporting inconsistent information to neighboring ASes, *etc.*).

To illustrate the incentive compatibility of next-hop routing, we revisit the example in Figure 4.1. Observe that in the unique stable routing state ASes A, B, and C send traffic along the routes Ad, Bd, and CBd, respectively. Hence, BGP is guaranteed to converge to a routing state where each AS directs traffic via its most preferred *feasible* next-hop AS. (Observe that, while B would prefer using C as a next-hop, no route from C to d that does not traverse B exists, and hence B cannot hope to send traffic through C.) [53] shows that this is true for general network topologies (see also [63]).

4.5.3 End-to-end Security Mechanisms

Today, an AS can use BGP policies to avoid paths that travel through undesirable ASes. For example, the U.S. government may want to avoid directing their traffic through other countries. Similarly, an AS may wish to avoid ASes known to perform censorship, conduct wiretapping, or offer poor performance. This is achieved by applying regular expressions to the AS-PATH to assign lower preference to routes that contain the undesirable ASes, or filtering these routes entirely. Under next-hop routing, an AS can no longer specify BGP routing policies that avoid remote undesirable ASes or countries rendering such "AS-avoiding policies" impossible.

While next-hop routing renders "AS-avoiding policies" impossible, we point out that these kinds of policies come with no guarantees; the AS-PATH lists the sequence of ASes that propagated the BGP announcement, not the path the *data packets* necessarily traverse (and these can differ even for benign reasons). We believe that relying on BGP for data-plane security is *misguided*. It is precisely when issues like confidentiality and integrity are involved that the matter should not be left to chance, or misplaced trust. Instead, we argue that these guarantees should be assured in other (end-to-end) ways, such as encryption and authentication, *e.g.*, using the mechanisms in [64].

4.5.4 Discussion

We have shown that next-hop routing renders some dangerous attacks against BGP (almost all AS-PATH-based attacks, *e.g.*, path-shortening attacks) ineffective, while making other attacks less harmful, or non-beneficial for the attacker. We have also shown how end-to-end mechanisms, which can be incrementally deployed, can be used to improve data-plane security. Combining these measures will create a routing system that is significantly more secure than today's BGP. However, next-hop routing alone does not provide full protection against all attacks. Specifically, next-hop routing is still vulnerable to "prefix hijacking" attacks, where the attacker announces an IP prefix which it does not own, as well as to attacks that trigger BGP's loop-detection mechanism.

Over the past decade the standards and research communities have devoted much effort to securing BGP against prefix hijacking and more sophisticated attacks. We are finally witnessing the initial deployment of the Resource Public Key Infrastructure (RPKI)—a cryptographic root-of-trust for Internet routing that authoritatively maps ASes to their IP prefixes and public keys. RPKI will enable an AS to authenticate that the origin of a route announcement indeed owns the announced prefix—a property called "origin authentication". In addition, there is much debate about the deployment of mechanisms for AS-level path validation (*e.g.*, S-BGP, soBGP, BGPsec), which will enable an AS to verify that an announced route actually exists (and was announced to the announcer), thus also preventing attacks against BGP's loop-detection mechanism.

4.6 Traffic Management

Today's routers prefer shorter AS-PATHs as an indirect way to improve end-to-end performance and avoid selecting backup routes. In this section, we discuss how network operators can more naturally achieve their traffic-management goals *without* relying on the AS-PATH. First, we discuss how to rank next-hop ASes based on measurements of end-to-end path performance. Then, we discuss how operators can balance load

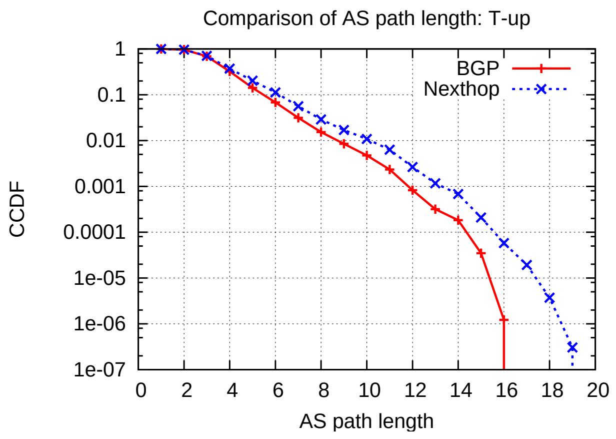


Figure 4.8: ASes's path lengths after a prefix injection.

by ranking next-hop ASes on a per-prefix basis and tagging backup paths with a BGP community attribute.

4.6.1 Performance-driven Routing

Next-hop routing can lead to longer paths than conventional BGP. In practice, AS-PATH length is not a good indicator of performance[2], but using shorter paths is generally better (all other things being equal).

To understand the impact on AS-PATH length, we analyze the differences in path lengths for different ASes across a range of events. Specifically, we evaluate next-hop routing and regular BGP, across prefix injection ("T-up", Figure 4.8), link failure ("T-long", Figure 4.9), and link recovery ("T-short", Figure 4.10). Our experiments show that most ASes (68.7%–89.9%, depending on the event) have the same AS-PATH length under BGP and next-hop routing, and most other ASes experience just one extra hop. Still, a non-trivial fraction of ASes see even longer paths. While these paths may perform reasonably well, some ASes may indeed experience worse performance. As a result, we believe that a static next-hop ranking should *not* be the only factor in routing decisions.

Given AS-PATH length only loosely correlates with path performance, we argue that routers should make decisions based on measurements of path quality. Routers could adjust (i) the ranking of the next-hop ASes and (ii) the splitting of traffic over multiple next-hop ASes with the same rank, based on the performance metrics (e.g., throughput, latency, or loss) of interest. Different kinds of ASes may select

[2]AS-PATH length is (at best) loosely correlated with end-to-end propagation delay—a route with short AS-PATH can have many *router*-hops, or be long in terms of *physical* distance. Moreover, in practice, shorter AS path also does not necessarily mean larger throughput.

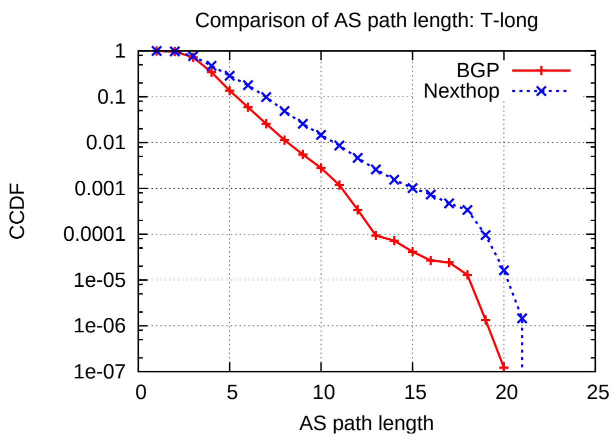


Figure 4.9: ASes's path lengths after a link failure.

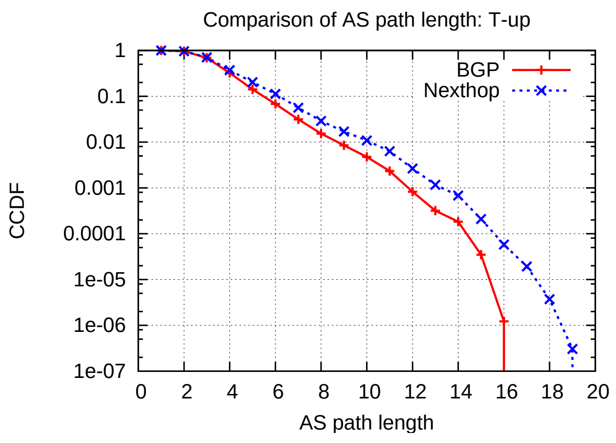


Figure 4.10: ASes's path lengths after a link recovery.

different techniques for monitoring path performance, such as:

Multihomed stub ASes: Multi-homed stub ASes can apply existing measurement techniques for intelligent route control supported in commercial routers (e.g., [77]). Round-trip performance measurements are relatively easy to collect, since the stub AS sees traffic in both the forward and reverse directions.

Online service provider: Online service providers, such as Google and Microsoft, can easily monitor end-to-end performance for their clients (e.g., by logging TCP-level RTT statistics at their servers [28]).

Transit providers: Performance monitoring is more difficult for transit providers, since most traffic does not start or end in their networks, and asymmetric routing may cause them to see one direction of traffic but not the other. Still, passive flow measurements can be used to infer performance [78]. Transit providers can also measure performance directly by using hash-based sampling [79] to sample both directions of the traffic—for the subset of flows that traverse the AS in both directions. This technique also ensures that AS measures just the downstream portion of the path to the destination, rather the part between the source and the ISP.

In all three cases, ASes can measure the performance of multiple paths by using “route injection” [80] to direct a small portion of traffic on each alternate path to continuously track performance. In addition, ASes can use active probing to monitor alternate paths, rather than directing “real” customer traffic over these paths.

Based on the performance measurements, an AS can adapt how it directs traffic over multiple next-hops. Designing and analyzing effective adaptive load-balancing techniques is a rich research topic in it own right. Recent work suggests that it is indeed possible to design stable and efficient controllers for adapting the flow of traffic over multiple paths [81,]. We plan to study this issue in greater depth as part of future work.

4.6.2 Interdomain Traffic Engineering

In addition to selecting paths with good performance, operators rely on interdomain path selection to balance load on their network links. Operators perform traffic engineering by measuring traffic volumes and selecting paths that optimize the flow of traffic over different links. Today, some operators use the AS-PATH to aid in traffic engineering. We believe other approaches (that do not rely on the AS-PATH) are more appropriate:

Outbound traffic engineering: Operators balance load on edge links to other ASes by adjusting the policies that assign “local preference” to BGP routes. This is still possible under next-hop routing, with the restriction that the preferences depend only the immediate neighbor rather than subsequent hops in the AS-PATH. As such, network operators cannot use “regular expressions” on the AS-PATH to (say) direct some traffic through an alternate egress point based on whether the *second* AS in the path is even or odd [82]. Policies based on regular expressions are arguably quite clumsy, and may not be widely used in practice. Instead, network operators could select different next-hop rankings for different (groups of) destination prefixes as a way to direct some traffic over other paths.

Inbound traffic engineering: Controlling the flow of *incoming* traffic is notoriously difficult, since Internet routing is destination-based. Today, some ASes use “AS prepending”—artificially adding extra hops to make the AS-PATH look longer—to make routes through them look less desirable to others. This is often used as a (somewhat clumsy) way to signify a "backup" route. Observe that next-hop routing, by removing AS-PATH length from the decision process, makes AS prepending ineffective. Instead, network operators could use BGP communities to signal backup routes to neighboring ASes [83]; if these ASes export the "signal" to their neighbors, other ASes can also give lower ranking to backup paths. Perhaps more importantly, path-quality monitoring allows ASes to make decisions based on performance, leading naturally to routing decisions that avoid placing excessive load on lower-bandwidth backup links. Still, just as with today's BGP, inbound traffic engineering remains somewhat of a "black art."

In addition, traffic engineering greatly benefits from multipath routing—something next-hop routing supports much more naturally than conventional BGP, as discussed earlier in Section 4.4. An AS can easily split traffic over *multiple* next-hops leading to the same destination; adjusting the fraction of traffic assigned to each next-hop is a much finer-grain approach to traffic engineering than selecting a single path for each destination prefix.

4.7 Related Work

Interdomain routing has been an active research area, ever since early work identified thorny problems with BGP [50, 84, 51]. Since then, many papers have characterized BGP's behavior (both theoretically [66, 76, 53], and via measurement [2, 56]), and designed techniques for managing BGP (to detect configuration mistakes [57] and automate traffic engineering [82]). Other research has proposed extensions to BGP (particularly to improve security [44, 45] and convergence [42, 43], or support multipath routing [72, 48, 42, 73, 74, 64]), or designed new interdomain routing architectures [46, 47, 48, 49]. We do not attempt to provide a comprehensive overview.

Our work is inspired by recent theoretical work [53, 54, 63] on BGP policy restrictions that lead to desirable global properties. We explore whether we can make *reality* look more like those models.

4.8 Summary

BGP suffers from many serious problems. We argued that *simplifying* BGP is an attractive alternative to extending or replacing BGP. We proposed next-hop routing—three simple constraints on how routes are selected and exported, combined with external mechanisms for data-plane monitoring and security. Next-hop routing allows ASes sufficient expressiveness to realize their business and engineering objectives, while sidestepping some of BGP's major problems (*e.g.*, slow convergence, large attack surface, incentives to "lie", difficulty in supporting multipath routing, and more).

Our work leaves several interesting directions for future research. We plan to explore how to remove the AS-PATH attribute entirely, that is, how not to rely on the AS-PATH even for loop detection purposes (*e.g.*, by detecting forwarding loops through data-plane monitoring). We also plan to further investigate the stability and efficiency of adaptive interdomain routing based on measurements of data-plane performance.

4.9 Appendix

4.9.1 Modeling BGP Dynamics

We use the standard model for analyzing BGP dynamics put forth in [51]. The reader is referred to [51] for further details.

Network and policies. The network is defined by an *AS graph* $G = (N, L)$, where N represents the set of ASes, and L represents physical communication links between ASes. N consists of n *source-nodes* $\{1, \ldots, n\}$ and a unique *destination node* d. P^i denotes the set of "permitted" simple (noncyclic) routes from i to d in G. Each source-node i has a *ranking function* $\leq_i$, that defines an order over P^i. We allow ties between two routes in P^i only if they share the same next hop. The *routing policy* of each node i consists of $\leq_i$ and of i's *import* and *export policies*.[3]

Protocol dynamics. Under BGP at routing tree to the destination d is built, hop-by-hop, as knowledge about how to reach d propagates through the network. The process is initialized when d announces itself to its neighbors by sending update messages. From this moment forth, every active node establishes a route to d by repeatedly choosing the best route that is announced to it by its neighbors (according to $<_i$) and announcing this route to its neighbors (according to its export policy). The network is assumed to be *asynchronous*; ASes can act at different times and BGP updates can be arbitrarily delayed.

Stable states. Informally, a stable state is a global configuration that once reached remains unchanged.

[3]The import and export policies can be folded into the routing policies, by modifying the preferences so that paths that are filtered out have the lowest possible value. Thus, we do not explicitly model these.

Formally, a **stable state** is an n-tuple of routes in G, $R_1, ..., R_n$, such that: (1) If for two nodes $i \neq j$ it holds that j is on R_i, then it must hold that $R_j \subset R_i$ (that is, R_j is a suffix of R_i). (2) If there is a link $(i, j) \in L$, and $R_i \neq (i, j)R_j$, then $(i, j)R_j <_i R_i$. It is easy to show [51] that a stable state is always in the form of a tree rooted in d.

4.9.2 Gao-Rexford Framework

In the Gao-Rexford framework, neighboring ASes have one of two business relationships: *customer-provider* and *peering*. [66] presents three conditions that are naturally induced by the business relationships between ASes and proves that these conditions imply guaranteed BGP convergence to a stable state. The three Gao-Rexford conditions are the following:

- **Topology Condition:** there should be no customer-provider cycles in the AS hierarchy digraph.
- **Preference Condition:** an AS should prioritize customer-learned routes over peer- and provider-learned routes.
- **Export Condition:** an AS should not export peer-/provider-learned routes to other peers and providers.

If the Gao-Rexford conditions hold for a network, then BGP convergence to a stable state is guaranteed [66].

Chapter 5

Conclusion

Today's Internet routing was not designed with management and performance challenges in mind. Therefore, the network operators and researchers face a number of major challenges in minimizing performance disruptions in the network, including: (1) ISP's challenge in providing good transit for IP packets, (2) CDN's challenge in maximizing performance for services, and (3) research community's challenge in improving the design of the routing protocol for better data-plane performance. This dissertation addresses these challenges from three different perspectives.

In this dissertation, we identified two network troubleshooting challenges faced by ISPs and CDNs today, and presented solutions. We then took the protocol designer's perspective, and proposed a BGP variant designed for performance-driven route selection.

First, we identified that many network-management tasks in large ISP networks need to track route changes scalably and in real time. We formulated the problem as: given an IP packet which entered the network at a particular place and time, where does it leave the network, and what is the subsequent path to the destination. To answer this question, we need to keep track of the changes to the longest prefix match (LPM). We explored the option to track LPM changes using the forwarding table of the IP routers today, and pointed out the scaling challenges of this approach. Instead, we proposed the idea of grouping IP addresses into contiguous ranges which match the same set of prefixes. The concept of the address range not only makes the tracking of LPM changes scale, but also facilitates parallelization. We designed and implemented the Route Oracle tool based on this idea. We presented performance evaluation of the tool, with the following optimizations: pre-processing of address range records and parallelization of query processing. We demonstrated the ability of the Route Oracle tool to handle large number of queries at scale and in real time.

In our second case study, we focused on one of the major challenges that CDN networks face today: minimizing network latency for the clients accessing the CDN services. The problem is challenging, because latency may increase for many reasons, such as inter-domain routing changes and the CDN's own load-balancing policies. We first separated the many causes from the effects by using a *decision tree* to classify latency changes. Another challenge is that network operators group clients according to their ISP and geographic region to reduce measurement and control overhead, but the clients in a region may use multiple servers and paths during a measurement interval. To solve the problem, we proposed metrics that quantify the latency contributions across *sets* of servers and routers. Our analysis over a month of data from Google's CDN showed that nearly 1% of the daily latency changes increase delay by more than 100 msec. More than 40% of these increases coincide with inter-domain routing changes, and more than one-third involve a shift in traffic to different servers. Furthermore, we explored case studies involving individual events, and identified research challenges for measuring and managing wide-area latency for CDNs.

Finally, we took a broader perspective on improving the inter-domain routing system. The BGP protocol today does not perform well, because it converges slowly, selects paths without regard for performance, does not support multipath routing, and has numerous security vulnerabilities. Rather than adding mechanisms to an already complex protocol, or redesigning inter-domain routing from scratch, we proposed making BGP *simpler*, and handling issues such as data-plane performance and security where they belong—*outside* the routing protocol. We proposed a transition from today's *path-based routing* (where routing decisions depend on the entire AS-PATH) to *next-hop routing*—a solution that selects and exports routes based *only* on the neighboring domain. Based on experimental results, we showed that next-hop routing leads to significantly better network performance than path-based routing, alongside other advantages (including being more amenable to multipath routing and a reduced attack surface).

Collectively, the contributions of this dissertation provided solutions to help automate the network diagnosis process for ISPs and CDNs, with the proposal to revise the routing protocol for better performance.

Bibliography

[1] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)." RFC 4271, January 2006.

[2] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed Internet routing convergence," *IEEE/ACM Trans. on Networking*, vol. 9, pp. 293–306, June 2001.

[3] Y. Zhu, J. Rexford, S. Sen, and A. Shaikh, "Impact of Prefix-Match Changes on IP Reachability," in *Proc. Internet Measurement Conference*, 2009.

[4] B. Zhang, V. Kambhampati, M. Lad, D. Massey, and L. Zhang, "Identifying BGP routing table transfers," in *Proc. ACM SIGCOMM Workshop on Mining Network Data (MineNet)*, August 2005.

[5] 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 *Proc. USENIX/ACM NSDI*, May 2005.

[6] Rensys Blog, "Pakistan hijacks YouTube." `http://www.renesys.com/blog/2008/02/pakistan_hijacks_youtube_1.shtml`.

[7] "Packet Design Route Explorer." `http://www.packetdesign.com`.

[8] "University of Oregon Route Views Project." `http://www.routeviews.org`.

[9] RIPE, "RIPE Routing Information Service (RIS)." `http://www.ripe.net/ris`.

[10] M. Caesar, L. Subramanian, and R. H. Katz, "Towards localizing root causes of BGP dynamics." Technical Report, U.C. Berkeley UCB/CSD-04-1302, 2003.

[11] A. Feldmann, O. Maennel, Z. M. Mao, A. Berger, and B. Maggs, "Locating Internet Routing Instabilities," in *Proc. ACM SIGCOMM*, 2004.

[12] R. Teixeira and J. Rexford, “A Measurement Framework for Pin-pointing Routing Changes,” in *ACM SIGCOMM Network Troubleshooting Workshop*, 2004.

[13] A. Broido and k. claffy, “Analysis of RouteViews BGP data: Policy atoms,” in *Proc. Network Resource Data Management Workshop*, 2001.

[14] Y. Afek, O. Ben-Shalom, and A. Bremler-Barr, “On the structure and application of BGP policy atoms,” in *Proc. Internet Measurement Workshop*, pp. 209–214, 2002.

[15] T. Bu, L. Gao, and D. Towsley, “On characterizing BGP routing table growth,” *Computer Networks*, vol. 45, pp. 45–54, May 2004.

[16] X. Meng, Z. Xu, B. Zhang, G. Huston, S. Lu, and L. Zhang, “IPv4 address allocation and BGP routing table evolution,” *ACM Computer Communication Review*, January 2005.

[17] R. Mahajan, D. Wetherall, and T. Anderson, “Understanding BGP misconfiguration,” in *Proc. ACM SIGCOMM*, August 2002.

[18] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang, “BGP routing stability of popular destinations,” in *Proc. Internet Measurement Workshop*, November 2002.

[19] J. Karlin, S. Forrest, and J. Rexford, “Autonomous security for autonomous systems,” *Computer Networks*, October 2008.

[20] M. Lad, D. Massey, D. Pei, Y. Wu, B. Zhang, and L. Zhang, “PHAS: A prefix hijack alert system,” in *Proc. USENIX Security Symposium*, 2006.

[21] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang, “PlanetSeer: Internet path failure monitoring and characterization in wide-area services,” in *Proc. OSDI*, 2004.

[22] N. Feamster, D. Andersen, H. Balakrishnan, and M. F. Kaashoek, “Measuring the effects of Internet path faults on reactive routing,” *Proc. ACM SIGMETRICS*, 2003.

[23] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, “User-level path diagnosis,” in *Proc. ACM SOSP*, October 2003.

[24] Z. M. Mao, J. Rexford, J. Wang, and R. H. Katz, “Towards an accurate AS-level traceroute tool,” in *Proc. ACM SIGCOMM*, August 2003.

[25] F. Wang, L. Gao, O. Spatscheck, and J. Wang, "STRID: Scalable trigger-based route incidence diagnosis," in *Proc. IEEE International Conference on Computer Communications and Networks*, August 2008.

[26] P. Huang, A. Feldmann, and W. Willinger, "A non-intrusive, wavelet-based approach to detecting network performance problems," in *Proc. Internet Measurement Workshop*, November 2001.

[27] V. N. Padmanabhan, L. Qiu, and H. Wang, "Server-based inference of Internet performance," in *Proc. IEEE INFOCOM*, March 2003.

[28] M. Szymaniak, D. Presotto, G. Pierre, and M. V. Steen, "Practical large-scale latency estimation," *Computer Networks*, 2008.

[29] Cisco NetFlow, `http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html`.

[30] Z. M. Mao, C. Cranor, F. Douglis, M. Rabinovich, O. Spatscheck, and J. Wang, "A precise and efficient evaluation of the proximity between web clients and their local DNS servers," in *Proc. USENIX Annual Technical Conference*, 2002.

[31] E. Katz-Bassett, H. V. Madhyastha, V. K. Adhikari, C. Scott, J. Sherry, P. van Wesep, T. Anderson, and A. Krishnamurthy, "Reverse traceroute," in *Proc. Networked Systems Design and Implementation*, April 2010.

[32] B. Krishnamurthy, C. Wills, and Y. Zhang, "On the use and performance of Content Distribution Networks," in *Proc. Internet Measurement Workshop*, November 2001.

[33] A. Shaikh, R. Tewari, and M. Agarwal, "On the effectiveness of DNS-based server selection," in *Proc. IEEE INFOCOM*, August 2002.

[34] J. Pang, A. Akella, A. Shaikh, B. Krishnamurthy, and S. Seshan, "On the responsiveness of DNS-based network control," in *Proc. Internet Measurement Conference*, October 2004.

[35] B. Ager, W. Muehlbauer, G. Smaragdakis, and S. Uhlig, "Comparing DNS resolvers in the wild," in *Proc. Internet Measurement Conference*, November 2010.

[36] C. Contavalli, W. van der Gaast, S. Leach, and D. Rodden, "Client IP information in DNS requests," May 2010. Expired Internet Draft, draft-vandergaast-edns-client-ip-01.

[37] Z. Zhang, M. Zhang, A. Greenberg, Y. C. Hu, R. Mahajan, and B. Christian, "Optimizing cost and performance in online service provider networks," in *Proc. Networked Systems Design and Implementation*, April 2010.

[38] C. Huang, A. Wang, J. Li, and K. W. Ross, "Measuring and evaluating large-scale CDNs," in *Proc. Internet Measurement Conference*, 2008.

[39] R. Krishnan, H. V. Madhyastha, S. Srinivasan, and S. Jain, "Moving beyond end-to-end path information to optimize CDN performance," *Proc. Internet Measurement Conference*, 2009.

[40] M. B. Tariq, A. Zeitoun, V. Valancius, N. Feamster, and M. Ammar, "Answering 'what-if' deployment and configuration questions with WISE," *Proc. ACM SIGCOMM*, 2008.

[41] N. Kushman, S. Kandula, and D. Katabi, "Can you hear me now?! it must be BGP," *ACM SIGCOMM Computer Communications Review*, April 2007.

[42] N. Kushman, S. Kandula, D. Katabi, and B. Maggs, "R-BGP: Staying connected in a connected world," in *Proc. USENIX NSDI*, 2007.

[43] P. B. Godfrey, M. Caesar, I. Haken, Y. Singer, S. Shenker, and I. Stoica, "Stabilizing route selection in BGP," tech. rep., University of Illinois at Urbana-Champaign, January 2010. `http://hdl.handle.net/2142/14841`.

[44] S. Kent, C. Lynn, and K. Seo, "Secure border gateway protocol (S-BGP)," *Journal on Selected Areas in Communications*, no. 18(4), pp. 582–592, 2000.

[45] K. Butler, T. Farley, P. McDaniel, and J. Rexford, "A survey of BGP security issues and solutions," *Proceedings of the IEEE*, January 2010.

[46] L. Subramanian, M. Caesar, C. T. Ee, M. Handley, M. Mao, S. Shenker, and I. Stoica, "HLP: A next generation inter-domain routing protocol," in *Proc. ACM SIGCOMM*, pp. 13–24, ACM, 2005.

[47] X. Yang, D. Clark, and A. W. Berger, "NIRA: A New Inter-domain Routing Architecture," *IEEE/ACM Trans. Networking*, vol. 15, no. 4, pp. 775–788, 2007.

[48] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica, "Pathlet routing," in *Proc. ACM SIGCOMM*, 2009.

[49] J. P. John, E. Katz-Bassett, A. Krishnamurthy, T. Anderson, and A. Venkatarmani, "Consensus routing: The Internet as a distributed system," in *Proc. Networked Systems Design and Implementation*, April 2008.

[50] K. Varadhan, R. Govindan, and D. Estrin, "Persistent route oscillations in inter-domain routing," *Computer Networks*, vol. 32, pp. 1–16, March 2000.

[51] T. G. Griffin, F. B. Shepherd, and G. Wilfong, "The stable paths problem and interdomain routing," *IEEE/ACM Trans. Netw.*, vol. 10, no. 2, pp. 232–243, 2002.

[52] O. Nordstrom and C. Dovrolis, "Beware of BGP attacks," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 1–8, 2004.

[53] J. Feigenbaum, M. Schapira, and S. Shenker, "Distributed algorithmic mechanism design," in *Algorithmic Game Theory* (N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, eds.), ch. 14, Cambridge University Press, 2007.

[54] S. Goldberg, S. Halevi, A. D. Jaggard, V. Ramachandran, and R. N. Wright, "Rationality and traffic attraction: Incentives for honest path announcements in BGP," in *Proc. ACM SIGCOMM*, pp. 267–278, 2008.

[55] S. Goldberg, M. Schapira, P. Hummon, and J. Rexford, "How secure are secure interdomain routing protocols?," in *Proc. ACM SIGCOMM*, 2010.

[56] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP misconfiguration," in *Proc. ACM SIGCOMM*, 2002.

[57] N. Feamster and H. Balakrishnan, "Detecting BGP configuration faults with static analysis," in *Proc. Networked Systems Design and Implementation*, 2005.

[58] Z. Yin, M. Caesar, and Y. Zhou, "Towards understanding bugs in open source router software," *ACM SIGCOMM CCR*, 2010.

[59] Renesys, “Longer is not always better.” http://www.renesys.com/blog/2009/02/longer-is-not-better.shtml.

[60] Renesys, “AfNOG takes byte out of Internet.” http://www.renesys.com/blog/2009/05/byte-me.shtml.

[61] “Research experiment disrupts Internet, for some,” http://www.computerworld.com/s/article/9182558/Research_experiment_disruptsInternet_for_some.

[62] “Reckless Driving on the Internet,” http://www.renesys.com/blog/2009/02/the-flap-heard-around-the-worl.shtml.

[63] J. Feigenbaum, V. Ramachandran, and M. Schapira, “Incentive-compatible interdomain routing,” in *Proc. ACM Electronic Commerce*, pp. 130–139, 2006.

[64] D. Wendlandt, I. Avramopoulos, D. Andersen, and J. Rexford, “Don’t secure routing protocols, secure data delivery,” in *Proc. Hotnets V*, 2006.

[65] P. R. McManus, “A passive system for server selection within mirrored resource environments using AS path length heuristics,” April 1999. http://www.ducksong.com/patrick/proximate.pdf.

[66] L. Gao and J. Rexford, “Stable Internet routing without global coordination,” *IEEE/ACM Trans. on Networking*, vol. 9, no. 6, pp. 681–692, 2001.

[67] “BGP best path selection algorithm.” http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080094431.shtml.

[68] B. Zhang, R. Liu, D. Massey, and L. Zhang, “Collecting the internet AS-level topology,” *ACM SIGCOMM CCR, special issue on Internet Vital Statistics*, 2005.

[69] B. Quoitin and S. Uhlig, “Modeling the routing of an autonomous system with C-BGP,” *IEEE Network*, vol. 19, no. 6, 2005.

[70] Juniper, “Out-delay.” https://www.juniper.net/techpubs/software/junos/junos57/swconfig57-routing/html/bgp-summary32.html.

[71] P. Jakma, “Revisions to the BGP ’Minimum Route Advertisement Interval’.” Internet Draft draft-ietf-idr-mrai-dep-03, October 2010.

[72] W. Xu and J. Rexford, “MIRO: Multi-path interdomain routing,” in *Proc. ACM SIGCOMM*, pp. 171–182, 2006.

[73] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala, “Path Splicing,” in *Proc. ACM SIGCOMM*, August 2008.

[74] Y. Wang, M. Schapira, and J. Rexford, “Neighbor-Specific BGP: More flexible routing policies while improving global stability,” in *Proc. ACM SIGMETRICS*, 2009.

[75] “Understanding route aggregation in BGP.” `http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080094826.shtml`.

[76] H. Levin, M. Schapira, and A. Zohar, “Interdomain routing and games,” in *Proc. ACM STOC*, pp. 57–66, 2008.

[77] Cisco, “Cisco IOS Optimized Edge Routing Overview.” `http://www.cisco.com/en/US/docs/ios/oer/configuration/guide/oer-overview.html`, 2007.

[78] A. Gerber, J. Pang, O. Spatscheck, and S. Venkataraman, “Representative speed tests for free: Estimating achievable download speed from passive measurements,” in *Proc. Internet Measurement Conference*, November 2010.

[79] N. G. Duffield and M. Grossglauser, “Trajectory sampling for direct traffic observation,” in *Proc. ACM SIGCOMM*, 2000.

[80] Z. Zhang, M. Zhang, A. Greenberg, Y. C. Hu, R. Mahajan, and B. Christian, “Optimizing cost and performance in online service provider networks,” in *Proc. NSDI*, 2010.

[81] S. Kandula, D. Katabi, B. Davie, and A. Charny, “Walking the tightrope: Responsive yet stable traffic engineering,” in *Proc. ACM SIGCOMM*, pp. 253–264, 2005.

[82] N. Feamster, J. Borkenhagen, and J. Rexford, “Guidelines for interdomain traffic engineering,” *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 5, pp. 19–30, 2003.

[83] E. Chen and T. Bates, “RFC 1998: An application of the BGP community attribute in multi-home routing,” August 1996.

[84] C. Labovitz, R. Malan, and F. Jahanian, “Internet Routing Instability,” *IEEE/ACM Trans. Networking*, vol. 6, no. 5, pp. 515–526, 1998.